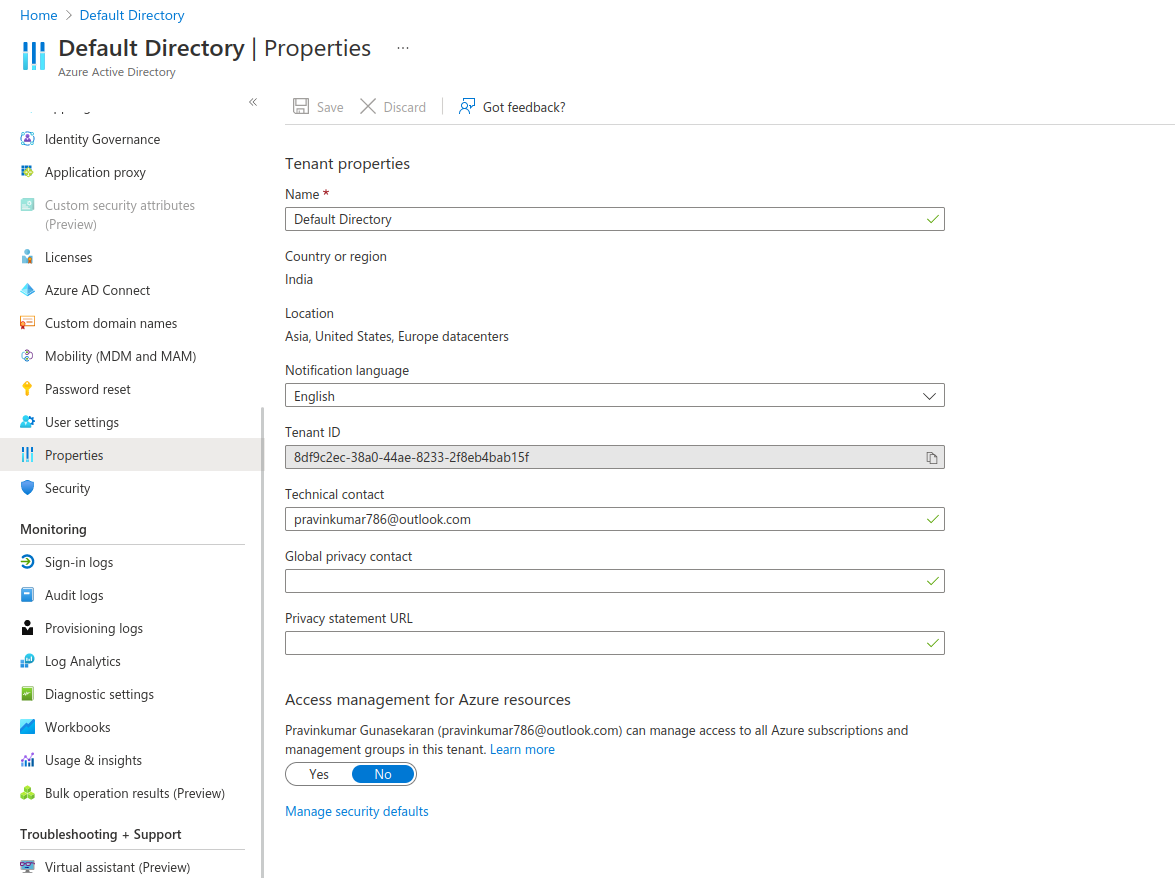
**Section 1**

**Security Defaults**

Security defaults is automatically enabled for all azure accounts by default. You can see this under Active directory -> Properties -> Security defaults



If this option is enabled then while we are loggin in Multi factor authentication is enabled. Means one time password will be send to mobile number or email id.

**Section 2**

**Virtual machine**

When we create virtual machine couple of other resources will be created along with virtual machine

* Virtual machine
* Network interface
* Public IP address (establish connectivity from outside internet)
* OS disk
* Data disk
* Network watcher
* Vnet(virtual network)
* Subnet
* NSG (Network Security Group – it is lightweight firewall which will filter incoming request and outgoing responses)

After create you can Go to resources -> Connect

=> RDP (remote desktop protocol – to connect windows VM)

=> SSH (To connect with Linux vm)

=> Bastion (secure way to connect both windows and Linux)

RDP-> download RDP -> Connect -> Enter credentials -> you can connect to VM

**Install IIS**

IIS is a web server from which outside world computers can request for a webpage and IIS is the one who will serve that request.

Start -> Server Manager -> Dashboard -> configure server -> Add roles and features -> Server roles -> Web server -> Add feature -> Install

After installation go to Tools -> Internet Information Server -> you can see our hosted web application in this IIS.

Copy public IP address and paste it in the browser, now ideally IIS server home page should be displayed. But it is showing “this site cant be reached”.

By default our VM IIS server is listens to port 80 http port. We need to allow this port to make communication from outside world. For this

Add Inbound Rule -> Service -> HTTP -> port 80 -> save. This should allow communication from outside world.

**Deploying a .NET Core app on Windows Server**

1. Assign a DNS name to the VM

2. Add a rule for port 8172 to the NSG

3. Add the role of the management service on the VM

4. Check the configuration of the Management service in IIS

5. Install the .net hosting bundle. This allows .net applications to be hosted on IIS

6. Install web deploy v3.6 tool

1. Go to VM overview -> DNS Name -> Click Not configured link -> enter name -> save

2. Add inbound rule for Allow for the port 8172,

=> service = Custom

=> Action = Allow

=> port = 8172

and save this

3. Server Manager Dashboard -> Add roles and features -> Server roles -> Web server (IIS) -> Management Tools -> Management services -> Add feature -> install

4. Go to Application in IIS -> Click on the application -> Management services -> Click Enable remote connections checkbox (notice this has port number 8172 enabled, that is the reason why we created inbound rule for that port. ) -> Apply

5. Server Manager Dashboard -> Local server -> IE Enhanced security configuration -> Off for sometime -> Download .net version you want to serve your application

We only need hosting bundle

6. Download web deploy tool

From your visual studio you can publish your web project. If you want to add your account then Tools -> Options -> Azure Service Authentication -> Choose an account for publish

We need to login to azure credentials, choose our target VM, subscribtion, Resource group etc and publish. This will create a publish profile

Instead of creating inbound rule for HTTP port 80 after creating the VM, we can also allow inbound port while creating the VM

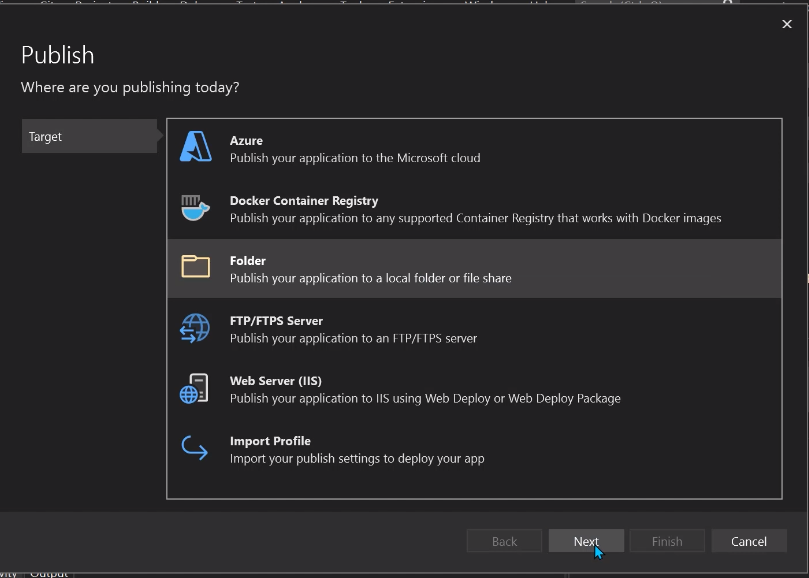
**Deploying .net app on the Linux VM**

We need putty application to connect with Linux VM

1. Host name - Public IP address -> Open -> Yes

Give credentials

2. To publish into Linux -> Create new publish profile -> Folder ( local folder or file share - it will create publish version of the project in bin\release\net6.0\publish) -> Publish



3. We need to copy all artifacts from local to linux server using winSCP

4. Host name , Uname and password -> WinSCP connected to the Linux

Left hand side is your local machine, and right hand side is the linux server -> drag and drop folder to Linux after copy is complete go to the terminal type pwd (present working directory) -> ls (list)

5. We need to install .net runtime on ubuntu also, for that we can see the instruction from Microsoft

Before install .NET, we need to add microsoft package signing key to your list of trusted keys and add the package repository then we can install SDK. Below is the command for linux ubuntu 20.04

wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

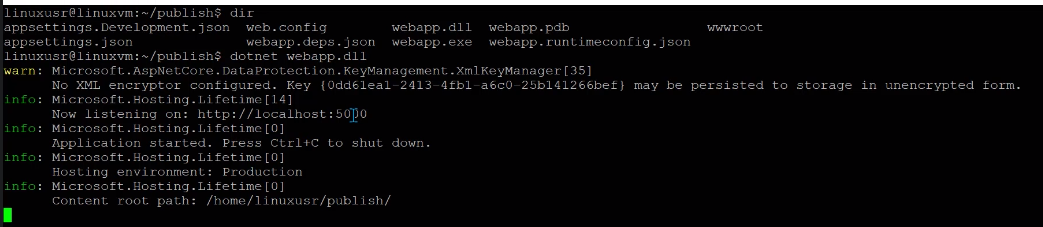
rm packages-microsoft-prod.deb

install SDK

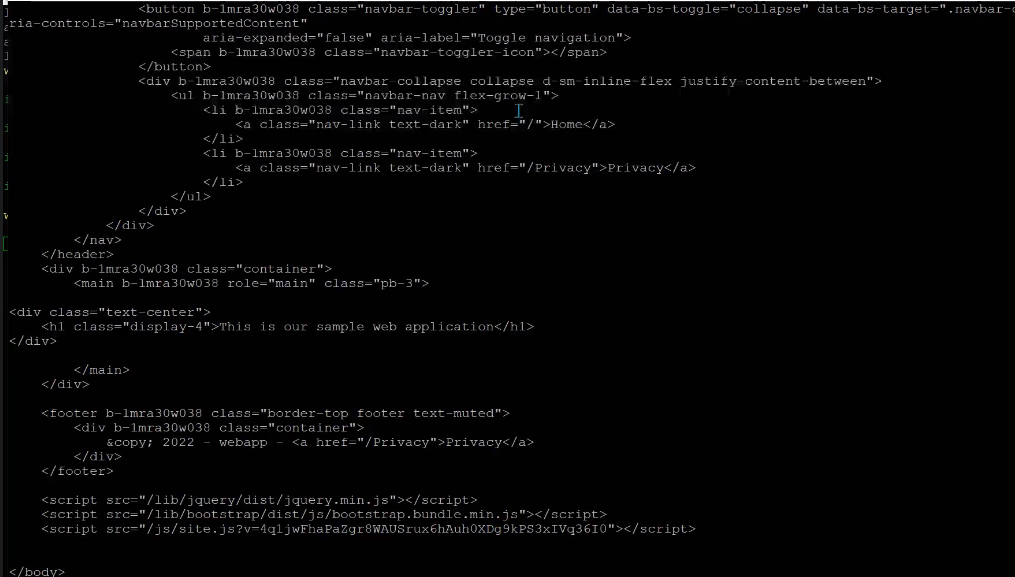
sudo apt-get update && \

sudo apt-get install -y dotnet-sdk-6.0

dotnet webapp.dll ->



Duplicate the session and run "curl http://localhost:5000 we will get the response



nginx is the webserver for Linux, to install it in linux

sudo get-apt install nginx

to do any configuration changes into linux server nginx we need to stop the server using the command sudo service nginx stop

We are inside the Home folder by default click one level up -> etc -> nginx -> sites-available-> default, right click and see the permission for this file, it is 644 we can change its permission to 777 and edit it

command to change permission is sudo chmod 777 default

Replace the location configuration to the below

location / {

proxy\_pass Http://localhost:5000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection Keep-alive;

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

now start the service again

sudo service nginx start -> now click on the public IP address then instead of nginx page our web application page will be displaying

Click on Publish -> Select Azure -> Select Resource group and app service from visual studio

**Section 3**

**App Service**

\* App service is platform as a service offering, as underlying machine is managed by Azure itself. It provides autoscaling, CI/CD ability. If you need to configure underlying virtual machine in a specific way than normal then you can deploy your application on Virtual machine, VM is Infrastructure as a Service offering.

In order to create a web app, we need to create app service plan first, based on this only you will get billed.

Create new resource -> Web app -> Publish -> 1. Code, 2. Docker Container 3. Static Web App

If you are using free credit then you can only use free F1 shared infrastructure , not possible to go for higher offering. With standard tier, you get 60 mins compute per day(for all apps together) . Also autoscaling and additional features are not available

**Azure SQL database**

When we use VM and install SQL server then you need to configure the server, configure for high availability and configure backups etc.

But if you choose Azure SQL database then the above mentioned points are taken care by azure itself.

When we create Azure SQL we will create two resources one is SQL server and one database.

We can configure which authentication we want and then we can choose network connectivity

\* No Access

\* Public endpoint

\* Private endpoint

Fireware rules

Allow Azure services and resources to access this server to yes

Add current client IP to connect with the database

**Building an application that connects to Azure DB**

This is same as ADO.net, connection string, DB name nothing different.

**Publishing from Github**

Create App service -> Deployment -> Deployment Settings -> Continous deployment -> Enable

Github actions -> we can configure when creating app service

Another way Git -> Git repository-> Create and push -> this will create repository in local and then push those changes to git

Web app-> Deployment center -> Settings -> Source -> Github -> Repository , Branch, RunTime save

Go to logs -> you can see log of deployment .

If you change and Git changes -> comments and commit then push push. If you go to the github then changes will be reflected. Go to webapp and see the logs

it will show changes in progress or completed. another log we can find. If we invoke the url new content can be seen.

**Using the Azure Web app connection string**

We can create connecting string under

Webapp -> Configuration -> App settings -> Connecting string -> Value (connecting string ) , Type SQLAzure, Name (connection string name)

in service class we need Iconfiguration interface to invoke this connection string

private readonly Iconfiguartion \_configuration;

public ProductService(Iconfiguartion \_iConfiguartion )

{

\_configuration = \_iconfiguration ;

}

to get the connection string we need

\_configuration.getConnectionString("ConnectionString")

**Azure app service logging**

1. Application logging (app code generated error )

2. Web server logging (Raw HTTP request date)

3. Detailed error message (Copies of .htm error pages)

4. Deployment logging

App service -> app service log -> Application logging (filesystem)

Level

If you dont want to use existing file system you can keep you logs in seperate storage account.

app service -> Log stream -> where you can see live loggin

**Azure Web Apps Autoscaling**

Auto scaling features ensures that our web app serves better when there is a change in demand. If the demand is high then it automatically add more VMs and the other way around.

This autoscaling is manual process and if you need automatic process then you need to go for standard pricing plan. Notice there is no automatic autoscaling in basic and free tiers. We are doing autoscaling on pricing plan not on the app service. Because all app services are priced based on this service plan only.

Scale up -> change pricing plan

Scale out -> auto scaling rule.

We can add our custom rule, If my app service's hosting machine

CPU percentage is above 70%

for the continous 10 mins

add another instances

Maximum you can add 3 instances

after adding instances wait for 5(cool down time) mins to allow the changes to take effect.

This is sample rule, you can also configure scale in rule to reduce the instances as well.

If you see the service plan you can see S1:3. Means now your app service plan is running 3 machines .

Go to Run history -> Resource instance count will show number of machine instances running

Operation name -> autoscaling done log will be there

**Custom domains**

Azure Web apps -> Custom domains

We need to get custom domains from 3rd party vendors like goDaddy.

Go to custom domains in the app service -> Add Custom domain -> enter domain and click validate.

Now Azure will give two types of record "A" and "TXT" records, you need to copy the value from Azure to your 3rd party vendor. This will ensure you own that domain.

After update this, wait for 10 mins to take changes effect. then again go to azure and add custom domain and validate. Now azure should show validated custom domain, and then add custom domain. Now if you give your custom domain then your web app should be shown.

Still we need to add SSL certificates to this custom domain otherwise connection is not secure error will be shown in the browser.

**SSL**

To do this Go to App service -> TLS/SSL settings -> Private key certificates (.pfx) (usually we need to buy certificate, but we are going to use a certificate provided by azure now)

Add custom domain name and certificate friendly name. You cannot download this certificate as this is managed by app service.

Now go back to custom domain -> Add binding -> Choose private certificate thumbprint -> TLS/SSL Type SNI SSL -> add binding

**Deployment slots**

Deployment slots are available in Standard and premium app service plan

With Deployment slots you have ability to validate application changes in the staging deployment slot and you can swap the staging slot with the production slot.

This helps eliminate the downtime for your application during deployment

you can easily swap deployment slots in case of issues.

Deployment slots will have unique URLs just like differents webapps

Add Deployment slot, then we have option to configure how much traffic to go to another deployment slot. While publishing, we need to select app service and expand deployment slot and select particular deployment slot and then publish

We have swap option to use which deployment slot.

When we swap environment we need to wait for warm up time to take effect the swap. We can improve this warmup time by setting up custom warm up

**Deployment slots with Databases**

While creating app config click on "Deployment slot setting" checkbox while create app configuration . If we do that then while doing a deployment slot, this will not be copied to new deployment slot.

**Azure App configuration**

We have seen that we have configured connection string in web application (configuration section ) itself. But we may have multiple configuration that can be saved in centralized place called App Configuration

Create New resource -> App configuration -> After resource created -> Configuration explorer -> Create -> key-value -> give connecting string key value pair and save

To use this we need to install AzureAppConfig nuget package in our project

App configuration -> Access Keys -> we can take primary key or secondary key.

var constr='primary key copied from azure app config'

builder.Host.COnfigureAppCOnfiguration(builder => {

builder.AddAzureAppConfigration(constr);

})

**Feature flag**

This allows us to toggle on and off based on the need.

App COnfiguration -> Feature manager -> Add -> name -> Apply

Install Microsoft.FeatureManagement nuget package.

on builder services in program.cs

builder.Services.AddFeatureManagement();

builder.Host.COnfigureAppCOnfiguration(options => {

options.connect(constr).UseFeatureFlags();

})

in service

private readonly IFeatureManager \_featureManger

**Section 4**

**Azure Function**

Azure Function is serverless , we need to associate Storage account to work with Azure Functions

We can choose Plan type as

\* Consumption based (serverless)

\* App service plan

\* Functions Premium

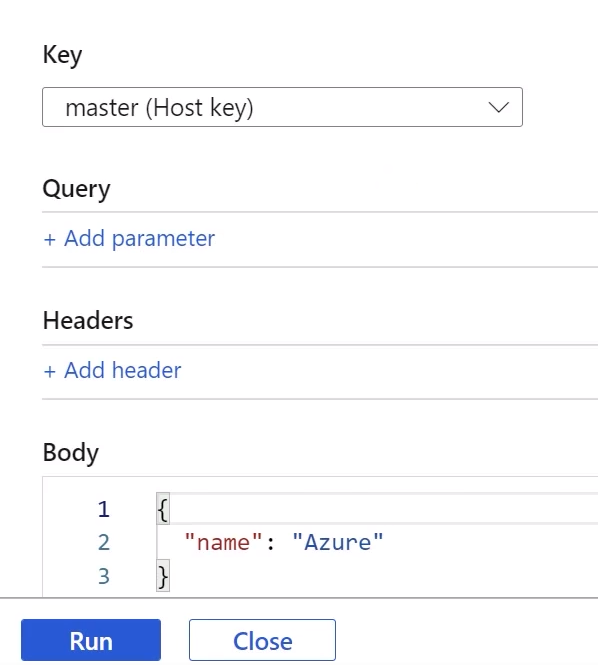
**Create a function**

****

* Function App -> Create function -> Development environment - Develop in portal

Select a template - HTTP trigger

Template Detail (Title section) - Give name for function

* Code + Test -> some template will be already written, that will extract query string from request and display it. Test and Run ->
* 

**Azure Function – Azure SQL database**

1. Create function app -> .net 6 -> HTTP Trigger -> Storage account -> storage emulator -> Just put ado.net code and call that method in Run method. While running you can get the URL of function app, if you invoke the URL then you can get response as per the query.

**Publish Azure function**

1. Publish -> Azure -> Azure Function App(Windows) -> Select function app name -> Finish -> Publish . -> Run from package file (recommended) is checked.

2. Notice existing function on the azure will be deleted and replaced by function from visual studio.

3. As we are deploying as a package, we will get the following warning “your app is currently in read only mode because you are running from a package file. To make any changes udate the content in your zip file and WEBSITE\_RUN\_FROM\_PACKAGE app setting”

4. If we add another method in our function app in visual studio and publish we can see two functions under function app in Azure.

[FunctionName(“getProduct”)]

public static async Task<IactionResult> Run(

[HttpTrigger(AuthorizationLevel.Function, “get”, Route = null )] HttpRequest req, Ilogger log)

{

// our code goes here

}

5. We can write many methods in one cs file, or we can create another cs file.

6. For post we have to take content from body then deserialize and use in it insert statement

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

Product product = JsonConvert.DeserializeObject<Product>(requestBody);

public async Task<bool> IsBeta()

{

return await \_featureManager.IsEnabledAsync("beta");

}

**PostMan tool for post request**

1. Choose Post -> Copy function url and then -> body -> raw -> paste payload to insert

**Connection string**

Function app -> Configuration -> Connection string -> Type =”SQL Azure”, Name=”ConStr”,

to get the connection string inside .net project

string connectionString = Environment.GetEnvironmentVariable(“SQLAZURECONNSTR\_SQLConStr”) // we need to prepend SQLAZURECONNSTR before our connection string. To tell the type of the connection string.

We can use this featureflag to control controllers or actions like below

using Microsoft.FeatureManagement;

[FeatureGate(MyFeatureFlags.FeatureA)]

public IActionResult Index()

{

}

If we create azure function from visual studio and publish to azure then we cannot edit code from azure. we can only edit function.json file.

Same as Webapps, we can also add connection string in function app. Go to FunctionApp -> Configuration -> Connection String ->

Environment.GetEnvironmentVariable("SQLAZURECONNSTR\_SQLConnectionstring") , SQLAZURECONNSTR is mandatory to include before your connectionstring

Call Azure Function from webapp

public async Task <List<Product>> GetProducts()

{

String FunctionURL =””;

using (HttpClient client = new HttpClient())

{

HttpResponseMessage response = await client.GetAsync(FunctionURL);

string content = await.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Product>>(content);

}

}

public void OnGet()

{

Products =\_productService.GetProducts().GetAwaiter().GetResult();

}

**Section 5**

## Benefits of containers

Before we look at the main use cases for Docker containers, it’s a good idea to revisit the characteristics of containers and why they’re used in DevOps.

Understanding the benefits of containers will help to understand why they are great for the different use cases that we’ll look at further below.

Containers offer the following benefits:

• Everything in a single package: A Docker image contains all of an application’s binaries, dependencies and libraries – everything the application needs to run. There’s no need to run an install process.

• Repeatability: A container image will run the same way, wherever it is run. So once my application is built as a Docker image, I can run my container wherever Docker is installed. I can also run multiple instances of my application very easily, and they will all behave in the same way.

• Lightweight: Containers are cheap, and fast to create and destroy1. This makes it easier to upgrade and patch software in containers. You don’t modify or upgrade an app inside a container when it’s running: you just kill the old container, and launch a new one with the upgraded application.

Also, Docker images are comprised of several stacked layers, which saves time when moving images around, as Docker only needs to download the layers which it doesn’t have already.

• Portability: With Docker, portability comes from the Docker image format. It’s like a zip file (actually it’s a TAR archive) which contains a whole application and all of its dependencies. You can download Docker images from public or private registries, or create your own. A container will run the same on a laptop, in the data center, or in the public cloud.

• Isolation: A Docker container offers a certain level of isolation. This means that an application runs in its own sealed environment, and cannot affect other applications when it’s running inside a container, unless you choose to allow it.

## What is Docker?

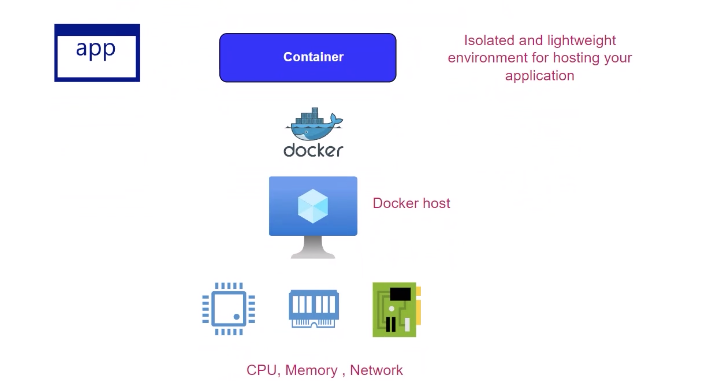
****

Image – This is read only template with instruction that are required to create the Docker container

Container – This is runnable instance of an image.

**Installing Docker on Linux Machine**

<https://docs.docker.com/engine/install/ubuntu/>

#### Set up the repository

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

$ sudo apt-get update

$ sudo apt-get install \

ca-certificates \

curl \

gnupg \

lsb-release

Add Docker’s official GPG key:

$ sudo mkdir -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

Use the following command to set up the repository:

$ echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

#### Install Docker Engine

#### Update the apt package index:

$ sudo apt-get update

To install the latest version, run:

$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

Verify that the Docker Engine installation is successful by running the hello-world image:

$ sudo docker run hello-world

**Running a simple container**

Container is an isolated environment for an application which has all app dependency, to run a container we need an image, either we can create and save it in a registory like docker hub, Azure container Registory or we can get one image from docker hub(repository of images).

1. Sudo docker images - will give list of docker available on our linux machine.
2. sudo docker run --name appngnix -p 80:80 -d nginx (this command first search for the “nginx” image in the local machine, if it is not found then it will fetch that image from Docker hub)
3. Add inbound rule to listen and allow port 80 in the VM, so that request will go from VM to container. Both VM and container listenting to port 80. (first 80 is the port for the container, container will listen to port 80 (by default) and that port is alinged with the Virual machine's port 80)
4. sudo docker ps - this will give running dockers

**Dockerize .Net application**

1. Create publish version of sqlapp application and copy to home/linuxuser via the winscp tool.

2. Create a docker file to build an image (What is docker file -> docker file will have information for the docker runtime on how to build an image.)

3. Dockerfile ensure the exact name for the docker file name. Docker file content will be as below

SAMPLE DOCKER FILE

*FROM mcr.microsoft.com/dotnet/aspnet:6.0*

*WORKDIR /app*

*COPY.. // current directory*

*EXPOSE 80*

*ENTRYPOINT ["dotnet","applicationname.dll"]*

4. Place the docker file in the location where our publish files are placed.

to build docker use below command

5. sudo docker build -t sqlapp . (docker file should be in your current working directory)

If there is any mistake in the container then we need to stop and remove the container and create that container again, below are necessary commands.

to stop and container

6. sudo dockefr stop 9ec (this can be full id of the container or first 3 characters)

7. to remove container from docker runtime

sudo docker rm d90

**Need for registroy**

Container registory will have all the readily available images, we can quickly use it to create containers, for example i need to create .net and mssql based application we can go and find the image that has this dependent softwares and create container and start doing the development. Instead of we create from scratch, we can go for prebuild images. Commonly used softwares like mongoDB, Nginx, mySQL all we have image, only for specific requirement we can create our own image.

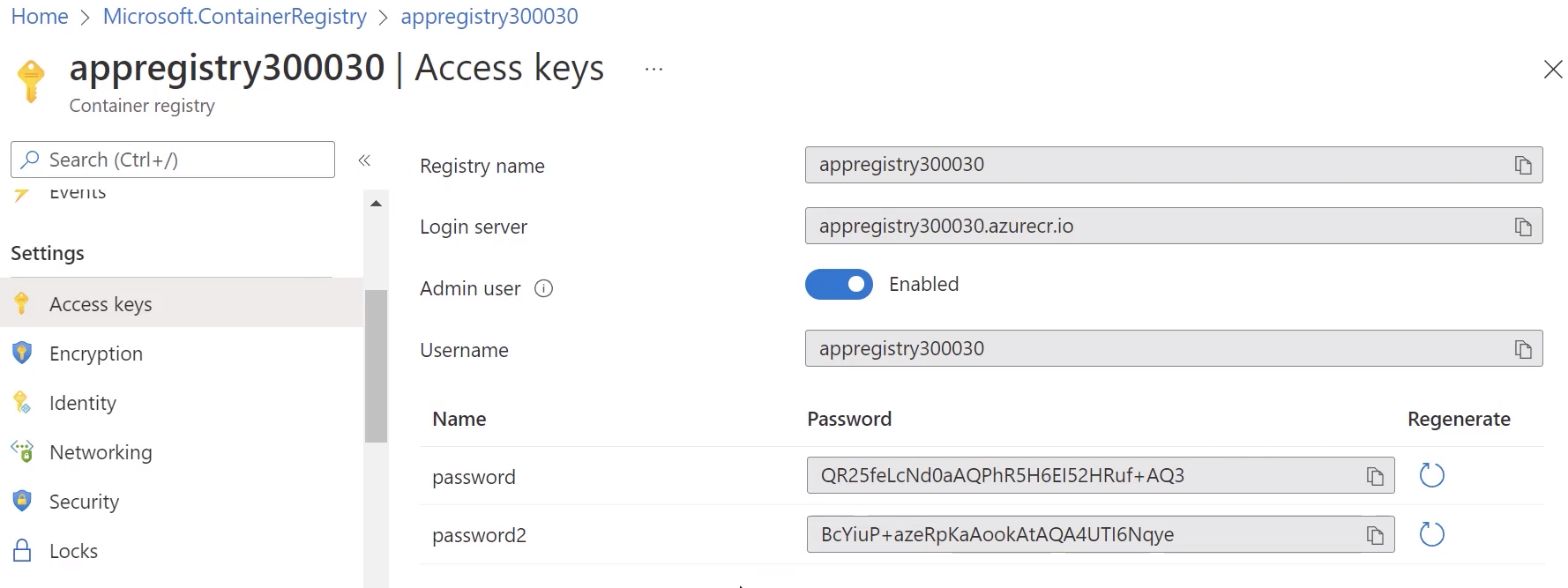
Docker Hub i such image registory, which is public. We have Azure container registroy servers the same purpose, but it is more private.

**What is Azure container registory?**

It is same as docker hub its a private version repositroy.

1. Create new resource -> Azure container registory -> Standard SKU

2. Go to Azure container registory -> Access keys -> Enable Admin user then you can see username and password to connect to container registory



3. We are trying to push our sqlapp image, which is .net and mssql app to Azure container registory.

For that we need to first Install Azure CLI in Linux VM

|  |
| --- |
| If you prefer a step-by-step installation process, complete the following steps to install the Azure CLI.  Get packages needed for the install process:  Bash    sudo apt-get update  sudo apt-get install ca-certificates curl apt-transport-https lsb-release gnupg  Download and install the Microsoft signing key:  Bash    curl -sL https://packages.microsoft.com/keys/microsoft.asc |  gpg --dearmor |  sudo tee /etc/apt/trusted.gpg.d/microsoft.gpg > /dev/null  Add the Azure CLI software repository:  Bash    AZ\_REPO=$(lsb\_release -cs)  echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ $AZ\_REPO main" |  sudo tee /etc/apt/sources.list.d/azure-cli.list  Update repository information and install the azure-cli package:  Bash    sudo apt-get update  sudo apt-get install azure-cli |

4. After CLI installation login to azure container registory

az acr login --name registoryname --username uname --password password

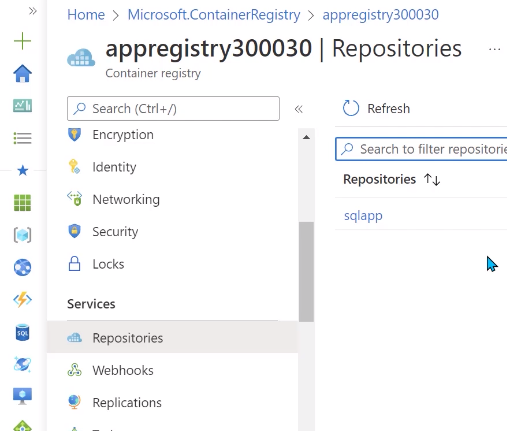
login succeeded

5. to tag our image on this container registory

sudo docker tag sqlapp appregistory300030.azurecr.io/sqlapp(sqlapp is image name)

sudo docker push appregistory300030.azurecr.io/sqlapp

6. Now we can see our image is added in the ContainerRegistroy -> Repositories



**Azure Container instances**

1. Once we create an image then we need to create publish build of the application and copy that publish files into linux machine, then create a docker file end run the docker. (we are creating a container based on the image, remember container is runnable image)

This can be simplified using container instances, once your image is ready map it with an instance then you have your image up and running, underlying infrastructre is managed for you.

1. Provides a fast and easy way to deploy containers

2. This is managed instance, docker installation and configuration everything will be managed for you (underlying infrastructure)

3. ACI can also get their own public IP and DNS name

4. You can also persist data via the use of azure file shares.

Ceate Resource -> Container Instance ->

Image source

\* Quickstart images (Microsoft)

\* Azure Container Registry

\* Other registry (from docker hub)

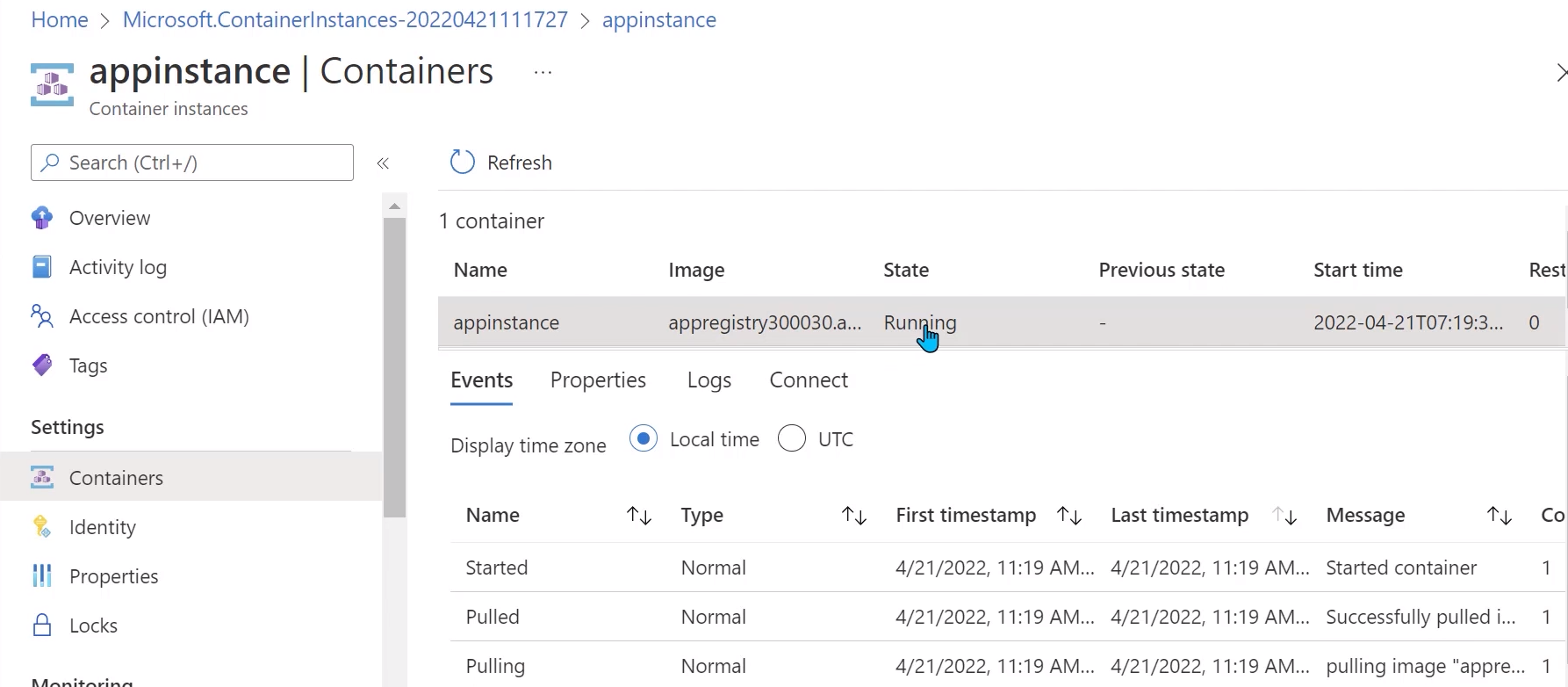
We can create image and its tag, which one we need . You needed to **enable admin user in Azure container registory to access that in Container instances**

Networking Type -> public / Private -> Public

Port -> 80 we are exposing on port 80. , Protocol -> TCP

Review and create will create a container based on the image we gave.

Go to containers we can see state of the container, properties, logs etc.

But with the Azure Container instances, we just need to tell our image and Azure will create runnable container automatically.

**Multi Stage builds**

* Remember, the size of the image is based on the instruction we give to that image. For example in nginx image is lightweight. it will only have bare minimum things to serve as a web server will not have entire linux features.
* Similarly, if you see dotnet images, it will have dotnet runtime so it will be bigger in size.
* Instead of doing build and get publish version on our local and then port that to image we can give this as instruction to docker file

FROM mcr.microsoft/dotnet/sdk:6.0

WORKDIR /source -- we are creating a working directory

COPY \*.csproj ./

COPY ..

RUN dotnet restore

COPY ..

RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:6.0

WORKDIR /app

COPY..

EXPOSE 80

ENTRYPOINT ["dotnet","applicationname.dll"]

1. We will copy entire project file, not the publish version, but the whole source code.

If we run this as it is. this will create layer by layers. we can create stages by doing this if that stage is done then it will remove that layer

FROM mcr.microsoft/dotnet/sdk:6.0 AS build – build is a stage it will be used in the copy highlighted below

WORKDIR /source -- we are creating a working directory

COPY \*.csproj ./

COPY ..

RUN dotnet restore

COPY ..

RUN dotnet publish -c Release -o out -- this step create an out folder without source folder

FROM mcr.microsoft.com/dotnet/aspnet:6.0

WORKDIR /app

COPY --from=build /source/out . -- we are copying out folder

EXPOSE 80

ENTRYPOINT ["dotnet","applicationname.dll"]

2. Go to Linux VM using putty -> sudo docker -build - t sqlapp-new . // this will do all the steps mentioned in the above scripts,

3. sudo docker ps –get all containers and its status, stop any running container, as we are going to run another container, that will also listen to port 80. to stop sudo docker stop 123 //123 first 3 character of the container

4. Now If we access Linux Vms public address then we can see our sqlapp-new running from container.

**Azure container groups**

This is a collection of containers that get scheduled on the same host machine

Example, if we have two containers one for UI and one for backend

The containers then share same lifecycle, resources, local network and storage volumes

The deployment of a container group is done via a Resource Manager template or a YAML file.

you can persist data via the use of Azure file shares.

3306 is port that will expose mysql

1. Create inbound rule to allow access on 3306 port, and service will be MySQL and save

2. sudo docker run --name=mysql-instance -p 3306:3306 --restart on-failure -d -e MYSQL\_ROOT\_PASSWORD=Azure123 mysql (name of my image)

sudo docker exec -it mysql-instance mysql -uroot -p

upon prompt give password. now you can see its connected to MySQL, you can run sql commands from putty itself

**creating a custom image of mysql**

1. Prepare the Dockerfile

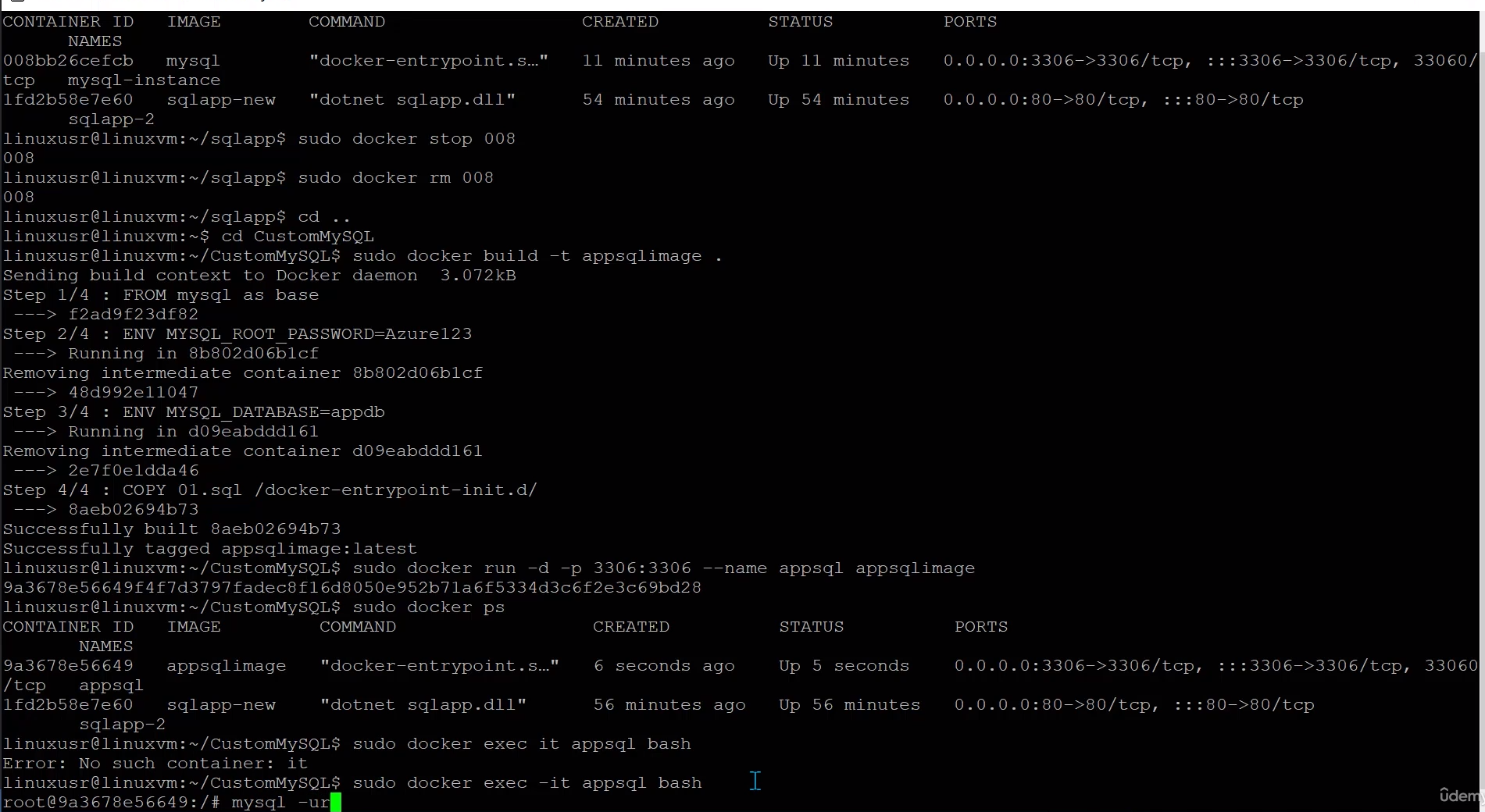
FROM mysql as base

ENV MYSQL\_ROOT\_PASSWORD=Azure123

ENV MYSQL\_DATABASE=appdb

COPY 01.sql /docker-entrypoint-initdb.d/

2. sudo docker build -t appsqlimage .



**Deploying custom Mysql**

1. Tag the custom image

sudo docker tag appsqlimage companyregistry.azurecr.io/appsqlimage

2. Log into the container registry

sudo docker login companyregistry.azurecr.io -u companyregistry -p u0K36bwuW4E85jUFQL6l6d/jEtf7wA71

3. Push the image to the registry

**sudo docker push companyregistry.azurecr.io/appsqlimage**

1. We need to replace connection string server name with localhost for db. Because it will be hosted on same server.

remove containers

i. sudo docker stop containername

ii. sudo docker rm containername

remove images

i. sudo docker images rm imagename

yaml file

apiVersion: 2022-10-10

location: northeurope

name: sqlAppGroup

properties:

containers:

properties:

image: regristoryname/imagename:latest

resources:

requests:

cpu: 1

memoruyInGb: 1.5

ports:

- port: 3306

-name: web

properties:

containers:

properties:

image: regristoryname/image2name:latest

resources:

requests:

cpu: 1

memoruyInGb: 1.5

ports:

- port: 80

osType: Linux

ipaddress:

type:Public

ports:

-protocol: tcp

port: 80

**imageRegistryCredentials**:

- server: companyregistry.azurecr.io

username: companyregistry

password: dxzy

type: Microsoft.containerInstance/containerGroups

**Open Cloud shell**

**Advanced settings**

**File share, storage account - create new**

**Upload yml file in cloud shell..**

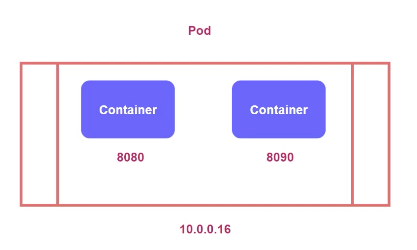
**az container create --resource-group resourcegroupname --file deployment.yml**

based on the above yml file, this container group will have two container instances

**Azure Kuberetes**

* Azure Kubernetes is a managed service of Kubernets service, it is managing containers at scale.
* Kubernetes is used to orchestrate your containers for hosting your applications.

Pod – A pod is used to group one or more containers. The pod gets shared storage and network resources.



Deployment – This is a declarative way to describe the state of the pods and ReplicateSets.

The deployment controller is used to ensure the desired state of the environment is always met.

**Deploying Kubernetes Service**

Create new Kubernetes service -> Cluster details -> Cluster preset configuration

=> Standard

=> Dev/Test

=> Cost-optimized

=> Batch processing

=> Hardened access

Mode Size,

Scale Method => Manual / Autoscale

Node Count =>

Container registry => Select our azure container registry

**Lab - Azure Kubernetes - Deployment - NGINX**

1. Azure Kubernetes service -> Create Workloads -> Create with YAML -> paste below lines.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

2. Azure Kubernetes service -> Create Services and ingresses -> Create with YAML -> paste below lines.

apiVersion: v1

kind: Service

metadata:

name: nginxservice

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

**Lab - Deploying our application on Kubernetes**

**app.yml**

apiVersion: v1

kind: Service

metadata:

name: appservice

spec:

selector:

app: myapp

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp

spec:

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- image: companyregistry.azurecr.io/kubernetesapp:latest

name: myapp

ports:

- containerPort: 80

name: myapp

**mysql.yml**

apiVersion: v1

kind: Service

metadata:

name: mysql

spec:

ports:

- port: 3306

selector:

app: mysql

clusterIP: None

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql

spec:

selector:

matchLabels:

app: mysql

strategy:

type: Recreate

template:

metadata:

labels:

app: mysql

spec:

containers:

- image: companyregistry.azurecr.io/appsqlimage:latest

name: mysql

ports:

- containerPort: 3306

name: mysql

**Section 6 - Tools**

**Powershell**

Powershell is an engine where we can run various commands to work with different services. It can be used on **Windows, Mac and Linux** distributions. Azure powershell is a section which will handle azure based resources.

Powershell provides command line **shell, script language and configuration** network

Powershell has the ability to work with .Net objects.

cmdlets - these are powershell commands.

**Install Azure powershell**

In windows machine, powershell is preinstalled, Powershell is having different module, Azure powershell is module that deal with Azure related functionality.

get version "$PSVersionTable"

Install Azure Az Powershell module.

0. Open Powershell in admin mode

1. Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

2. Install-Module -Name Az -Scope CurrentUser -Repository PSGallery -Force (in case of error use -AllowClobber)

ARM Templates are resource as code

3. After instalation Connect-AzAccount this will prompt credentials in browser upon successful log we can get Account, tenant and Subscription ids in Powershell.

On Ubuntu we can install like below

# Update the list of packages

sudo apt-get update

# Install pre-requisite packages.

sudo apt-get install -y wget apt-transport-https software-properties-common

# Download the Microsoft repository GPG keys

wget -q "https://packages.microsoft.com/config/ubuntu/$(lsb\_release -rs)/packages-microsoft-prod.deb"

# Register the Microsoft repository GPG keys

sudo dpkg -i packages-microsoft-prod.deb

# Update the list of packages after we added packages.microsoft.com

sudo apt-get update

# Install PowerShell

sudo apt-get install -y powershell

# Start PowerShell

pwsh

**Azure Powershell using visual studio code.**

Open VS code -> Extensions -> Install

\* Powershell

\* Azure powershell

Create new file -> Select language (default python, we can change to powershell. As soon as you change Powershell you can notice terminal starts PS /home/project (whatever your current directory )

Create 3 folders Azure web app -> Script.ps1

$ResourceGroupName="powershell-grp"

$Location="North Europe"

$AppServicePlanName="companyplan200"

$WebAppName="companyapp90999"

Connect-AzAccount

New-AzResourceGroup -Name $ResourceGroupName

New-AzAppServicePlan -ResourceGroupName $ResourceGroupName `

-Location $Location -Tier "B1" -NumberofWorkers 1 -Name $AppServicePlanName

New-AzWebApp -ResourceGroupName $ResourceGroupName -Name $WebAppName `

-Location $Location -AppServicePlan $AppServicePlanName

If we run this command, it will ask for credentials and this script itself will create resourcegroup and create webapp for us.

**Integrate Github**

Another folder Configuration with Github

$ResourceGroupName="powershell-grp"

$WebAppName="companyapp90999"

//we dont have to use AzConnect as we just connected with another powershell script in the previous example. otherwise connect to azure account is the first step.

$Properties=@{

repoUrl="url of your Github repository";

branch="master";

isManualIntegration="true";

}

Set-AzResource -ResourceGroupName $ResourceGroupName -Properties $Properties `

-ResourceType Microsoft.Web/sites/sourcecontrols -resourceName $WebAppName/web -ApiVersion 2015-08-01 -Force

**Deployment slots**

$ResourceGroupName="powershell-grp"

$WebAppName="companyapp90999"

$AppServicePlanName="companyplan200"

Set-AzAppServicePlan -Name $AppServicePlanName -ResourceGroupName $ResourceGroupName `

-Tier Standard

$SlotName="Staging"

New-AzWebAppSlot -Name $WebAppName -ResourceGroupName $ResourceGroupName -Slot $SlotName

//we dont have to use AzConnect as we just connected with another powershell script in the previous example. otherwise connect to azure account is the first step.

$Properties=@{

repoUrl="url of your Github repository";

branch="master";

isManualIntegration="true";

}

Set-AzResource -ResourceGroupName $ResourceGroupName -Properties $Properties `

-ResourceType Microsoft.Web/sites/slots/sourcecontrols -resourceName $WebAppName/$SlotName/web -ApiVersion 2015-08-01 -Force

// above code should create a deployment slot named staging. to switch our deployment slots from Production to staging follow the below commands

//As we already executed the above code, need to highlight the below and run it agin

$TargetSlot="production"

Switch-AzWebAppSlot -Name $WebAppName -ResourceGroupName $ResourceGroupName -SourceSlotName $SlotName -DestinationSlotName $TargetSlot

**Azure CLI**

Azure CLI is available in Mac, Linux and Windows

Install Azure cli for your respoective OS and to go command prompt and enter

"az login" -> prompted to give credentials once done your detail will be fetched on the CLI

Few CLI commands

//1. create app service plan

az appservice plan create --name azcliplan100 --resource-group docker-grp --is-linux //--is-linux check is underlying machine is for linux

//2. Create Web app

az webapp create --resource-group docker-grp --plan azcliplan100 --name dockerapp55000 --deployment-container-image-name appregistry30030.azurecr.io/webapp:latest

//3. If you want to turn on container logging

az webapp log config --name dockerapp55000 --resource-group docker-grp --docker-container-logging filesystem

//4. Enable the log stream

az webapp log tail --name dockerapp55000 --resource-group docker-grp

**Azure CLI - Azure Kubernetes**

Azure Cloud shell - this terminal can execute both Powershell scripts and Azure cli commands

Powershell may be prompted to you for admin rights, so if you dont have admin rights then you can go for the Azure cloud shell, it will be operate based on Azure logged in credentials.

Please note it requires Azure storage account.

1. Create resource group

\* az group create --name kubernetes-grp --location northeurope

2. Az aks create --resource-group kubernetes-grp --name appcluster --node-count 1 --enable-addons monitoring --generate-ssh-keys

// aks will create Azure Kubernetes services . Please wait for 5 mins for the step 2 to complete

3. kubectl tool is inbuild with kubernetes (kubectl is included with Azure cloud shell) . To install in your local machine az aks install-cli

4. Get credentials so that kubectl tool can interact with aks

az aks get-credentials --resource-group kubernetes-grp --name appcluster

5. kubectl get notes

6. We can deploy yaml files we created when work with containers. deploy and service.yml files

kubectl apply -f deployment.yml

kubectl apply -f service.yml

7. After this commands, you can go to appcluster -> workloads -> nginx deployment and go to services and ingresses -> nginxservice in place.

8. Delete resource group - az group delete --name kubernetes-grp

**ARM templates**

1. Infrastructure as code.

sample ARM template

"resources": [

{

"type": "Microsoft.Storage/storageAccounts",

"apiVersion": "2019-04-01",

"name": "mystorageaccount",

"location": "westus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {}

}

]

**Create ARM from VS code**

1. Extension -> search arm template ARM tools

2. Select language - > JSON -> type Arm (as we have already installed ARM tools we will get basic template

3. New-AzResourceGroupDeployment -ResourceGroupName app-grp -TemplateFile template.json

4. "copy":{

"name":"storagecopy",

"count":3

}

this command will create 3 instances having same configuration as mentioned in the resources. However we need to give unique name for the storage account. for that we can use the below expression

"name": "[concat(copyIndex(),'appstore444444')]

**ARM Template - Azure Virtual Machine**

Create resource -> Click Virtual machine -> Edit template -> (this will give all information needed, we can edit and use that ARM

location: "[resourceGroup().location]"

we have dependsOn property. It will say particular property is dependsOn another, for example Network interface is depend on PublicIpaddress and virtual networks. This is important azure is not going to follow the order we created json property. So it is important to mention depends on

**Section 7**

**Azure Storage**

There are different types of Storage account. If you create general purpose v2 , we can create

\* Blob (Large objects images, videos etc)

\* Table (store table data)

\* Queue (storign queues, used for sending and receiving messages)

\* File (used for file shares)

**Account types**

1. Standard General purpose v2

This is standard storage for blobs, file shares, queues and tables

2. Premium block blobs

This is supported for block and append blobs. This is when you want fast access to your blobs, high transaction rates

3. Premium page blobs

This used to store virtual harddisk for azure VM . This is when you want to fast access to your blobs high transaction rates

4.Premium file shares

If you want fast access to your files

Different types of blobs

\* Block blobs (blocks of data that can be managed individually)

\* Append Blobs (These block blobs are optimized for append operation, good for logging)

\* Page blobs (Virtual hard drive files )

**Upload a blob**

1. We need to create a container then we can upload any file

2. Upload -> Browse -> Select files ->open -> upload -> Edit (there are selected formats that can open, pdf it will not open)

3. By default it will create Block Blob

4. Upload a folder (this will first create a folder then create files into that

5. Data lake Gen 2 , this will create folder structure in place

**Accessing blob**

1. If you copy paste blob url and execute in the browser, you can see you not authorized error. It is private and not available for ananymous access

2. One way to enable access is(not recommended way) Change access level ->

\* Private (no anonymous access)

\* BLob (anaonymous read access for blobs only)

\* Container (anonymous read access for containers and blobs)

3. If we give option 2 or 3 now you can access blob.

**Different Authorization methods**

1. Access Keys

2. Shared Access Signatures

3. Azure Active Directory

**Using Azure Storage explorer**

Install this free tool from internet -> launch it. If we login, under our subscription we can see storage account and blob container.

If we provide the access key for the storage account then the user is having full access to the storage account.

**Shared Access Signature**

1. Clck on any object in storage account -> Generate SAS ->

\* Signing Key -> Key 1 / Key 2 - its going to use any one key to generate SAS key.

Permission -> Read

Start and end time we can configure

Allowed IP addresses ->

Allowed protocol -> HTTPS only

Generate SAS token and URL

Blob SAS token and

Blob SAS URL is generated

If we copy SAS URL then we can see the azure blob file using SAS token.

**SAS at account level**

We have more option now when we check SAS at the account level

Allowed services

Blob File Queue Table - we can check as required

Allowed resource types

Services container object - as required

Allowed Permissions

Read Write Delete List Add Create Update Process Immutable storage

Blob Versioning permission

Enables Deletion of versions

Allowed blob index permissoin

Read/Write Filter

If we try to access the SAS token we cannot access it, as it will have to access whole storage account, not single file . So instead of browser we need to use Azure Storage Explorer

We can only see what we have access to see. I have given only blob then only blob storage will be visible in storage explorer

**Stored Access Policy**

If SAS is compromised and wrong people get to know it then we need to invalidate. We have access policy

1. Access Policy -> Add policy -> "ReadPolicy -> Name)" save this.

2. Go to Azure Storage explorer -> Storage account -> Container -> Get Shraed Access policies -> Access Policy -> Select our ReadPolicy -> Create -> now you will get URL -> copy

3. Add Account -> Blob container -> SAS -> Paste URL here

4. If you want to revoke the access you can go to the Access policy and remove the permission.

Different types of SAS

1. Account SAS

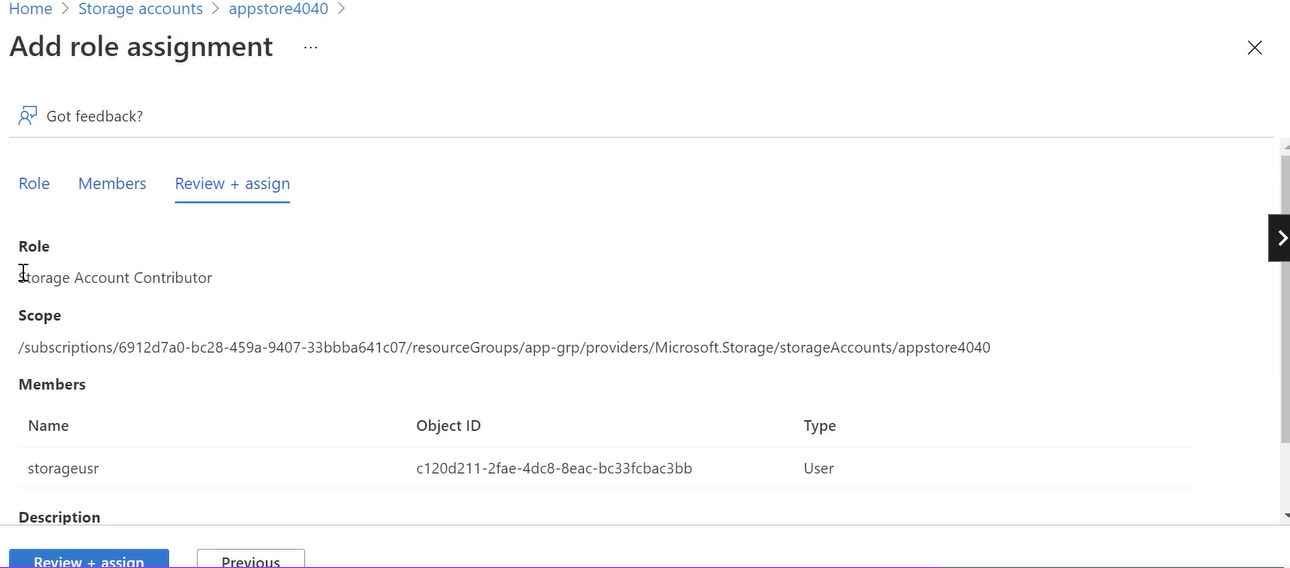
2. Service SAS

3. User delegation SAS

**Storage Accounts - Active Directory Authentication**

1. Create User in AD

2. Storage account -> Access control (IAM) -> Add -> Add role assignment -> storage account contributor -> Select member



Azure Storage account - Pricing Tiers

All prices are per GB per month.

Premium Hot Cool Archive

First 50 terabyte (TB) / month $0.15 per GB $0.018 per GB $0.01 per GB $0.00099 per GB

Next 450 TB / month $0.15 per GB $0.0173 per GB $0.01 per GB $0.00099 per GB

Over 500 TB / month $0.15 per GB $0.0166 per GB $0.01 per GB $0.00099 per GB

Hot is default access tier.

Cool - accessed infrequently stored for at least 30 days

Archive - Rarely accessed and stored for at least 180 days

Hot and cold is enabled for storage account level and archive is enabled for Blob level .

**Hot ,Cool access tiers**

Storage account -> Configuration settings -> Blob access tier -> Can change it to Hot and cool. What is default we choose is coming as inferred

Graphical user interface, text, application, email

Description automatically generated

**Archive access tier**

Setting the access tier to "Archive" will make your blob inaccessible until it is rehydrated back to "Hot" or "Cool" which may take several hours.

Graphical user interface, text, application, email

Description automatically generated

If you change your blob item to Archive, then you cannot view its content or delete that item from storage account. You need to change the access tier again.

There you can set the rehydrate priority to

\* Standard or

\* High (this means faster rehydrate and higher cost)

**Life cycle management policies**

If we feel we are using some objects less frequently then we can change it to Cool or archive access tier, but it is tedious to do that every single item. There is efficient way to do that.

Through some rules we can tell Azure to change access tier for an object or even delete object.

**Life cycle Management**

1. Transition - You can transition blobs from the cool to host access tier to save on storage cost

2. You can transition blobs, blob versions and blob snapshots

3. You can also define rules to delete blobs, blob versions and blob snapshots

4. Rules can be applied at the storage account level or to a subset of blobs.

Rules filter - you can define filters for blobTypes - blockBlob, appendBlob

Rule actions - You have actions such as tierToCool, tierToArchive and delete

Support - Rules are supported for blob and append blobs in general0purpose v2 accounts, Premium Block Blob and Blob Storage accounts.

Regions - this feature is available in all regions.

**How to do that?**

Storage account -> Data management -> Lifecycle management -> Add rule -> => Allow rule to all bobs in your storage account

* Limit Blobs with filters.

Blob type ->

=> Block blobs

=> Append blobs

Blob subtype ->

=> Base blobs

=> snapshots

=> versions

Base blobs were Last modified More than (days ago) -> 30

Then Move

=> to cool storage

=> to Archive storage

=> Delete the blob

**Note on the Rehydrate time**

Rehydration priority

When you rehydrate a blob, you can set the priority for the rehydration operation via the optional x-ms-rehydrate-priority header on a Set Blob Tier or Copy Blob operation. Rehydration priority options include:

Standard priority: The rehydration request will be processed in the order it was received and may take up to 15 hours to complete for objects under 10 GB in size.

High priority: The rehydration request will be prioritized over standard priority requests and may complete in less than one hour for objects under 10 GB in size.

|  |  |
| --- | --- |
| Authorization | Required. Specifies the authorization scheme, account name, and signature. For more information, see [Authorize requests to Azure Storage](https://learn.microsoft.com/en-us/rest/api/storageservices/authorize-requests-to-azure-storage). |
| Date or x-ms-date | Required. Specifies the Coordinated Universal Time (UTC) for the request. For more information, see [Authorize requests to Azure Storage](https://learn.microsoft.com/en-us/rest/api/storageservices/authorize-requests-to-azure-storage). |
| x-ms-version | Required for all authorized requests. For more information, see [Versioning for the Azure Storage services](https://learn.microsoft.com/en-us/rest/api/storageservices/versioning-for-the-azure-storage-services). |
| x-ms-rehydrate-priority | Optional. Indicates the priority with which to rehydrate an archived blob. This header is supported on version 2019-02-02 and later for block blobs. Valid values are High and Standard. You can set the priority on a blob only once. This header will be ignored on subsequent requests to the same blob. Default priority without this header is Standard. |

Any HTTP call we add headers like authorization, we can add optional x-ms-version to tell the rehydration priority.

**Azure Blob versioning**

Versions - Allow to keep previous versions of a blob

Restore - You can then restore an earlier version of the blob

Version Id - Each blob gets an initial version ID. A new version is set whenever the blob is updated

Feature Enable - You can enable or disable this feature at any point of time

Storage Account -> Data management -> Data protection -> Tracking -> Enable versioning for blobs

Graphical user interface, text, application, email

Description automatically generated

**Blob Snapshots**

Note: If we disable versioning for a storage account still the versions of the file will not be removed.

Storage accounts -> Data protections -> Snapshots (Take a copy /snapshot of file at that particular point of time, it’s a manual process)

Click on the snapshot -> Promote snapshot -> this will restore the snapshot as current version.

**Soft Delete**

Retention - You can specify a retention period for deleted objects, this is between 1 - 365 days.

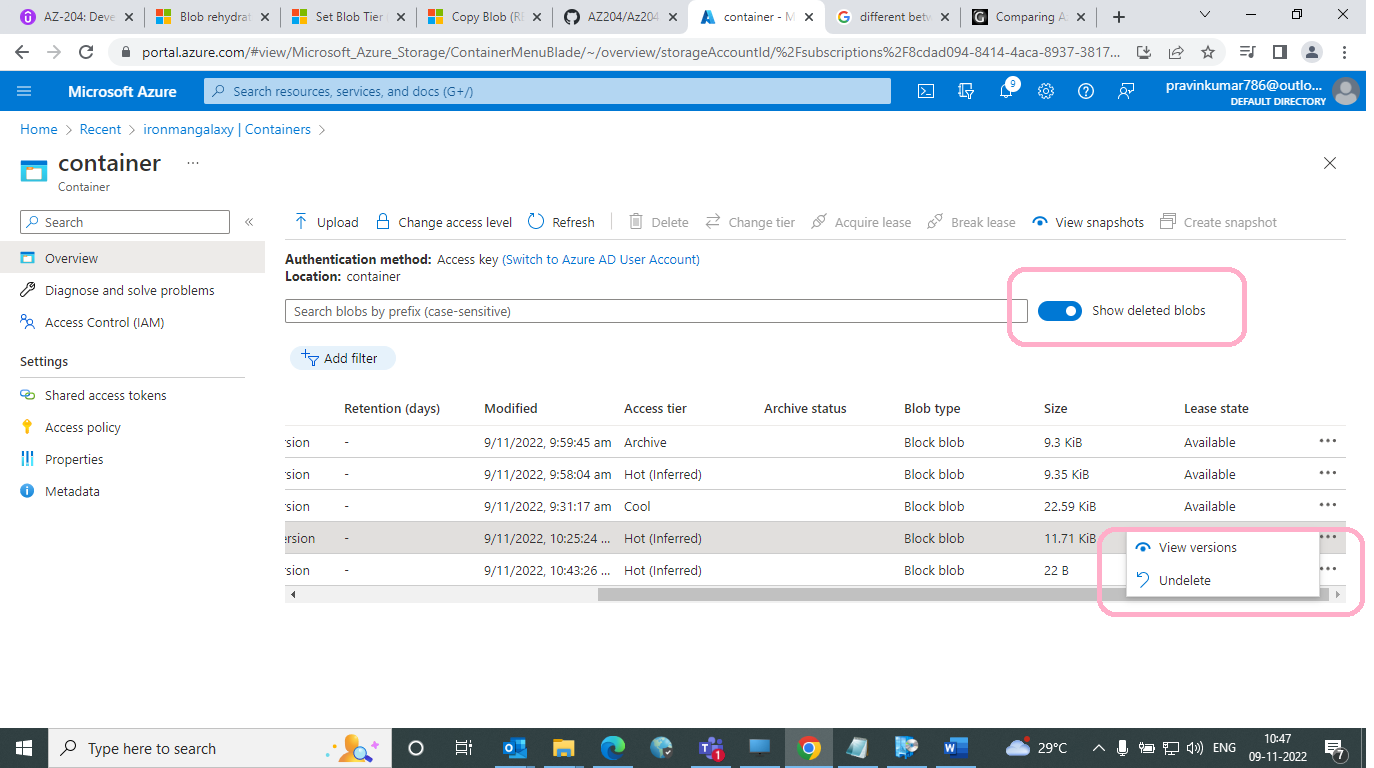
Availability - Depending on the retention period, the data will be made available after it has been deleted or overwritten

Restore - During the retention period, you can restore a deleted blob along with its snapshots

Change - You can change retention any time.

Storage account -> Data protections -> Enable soft delete for blobs -> 7 days is enabled by delete.

Delete the blob file -> Click on the switch "Show delete blobs" -> right most . . . -> Undelete



**Creating containers**

1. Create console project based on .net 6. -> Install Azure.Storage.Blobs ->

2. Program.cs ->

string connectionString ="constrfromAzureStoreAccount";

string containerName ="DesiredContainerName";

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

await blobServiceClient.CreateBlobContainerAsync(containerName); // we can specify second parameter as public access type Azure.Storage.Blobs.Models.PublicAccessType.Blob/BlobContainer/None

Console.WriteLine("Container created");

**.NET -Uploading a blob**

string connectionString ="constrfromAzureStoreAccount";

string containerName ="DesiredContainerName";

string blobName="script.sql ";

string filePath="c:\\temp\\script.sql"

BlobContainerClient blobServiceClient = new BlobContainerClient(connectionString, containerName); // we are directly connecting to Container of storage account

BlobClient blobClient= blobServiceClient.GetBlobClient(blobName);

await BlobClient.UploadAsync(filePath, true); // true is override flag, if it exists already.

Console.WriteLine("Container Updated ");

**.NET -list the blob**

Using Azure.Storage.Blobs;

string connectionString ="constrfromAzureStoreAccount";

string containerName ="DesiredContainerName";

BlobContainerClient blobContainerClient = new BlobContainerClient(connectionString, containerName); // we are directly connecting to Container of storage account

await foreach(BlobItem bi in blobContainerClient.GetBlobsAsync )

{

Console.WriteLine("The Blob Name is {0}", blobItem.Name);

Console.WriteLine("The Blob size is {0}", blobItem.Properties.ContentLength ); // size will ne in bytes

}

**.NET - Download a blob**

Using Azure.Storage.Blobs;

string connectionString ="constrfromAzureStoreAccount";

string containerName ="DesiredContainerName";

string blobName="script.sql ";

string filePath="c:\\temp\\script.sql"

BlobClient blobClient= new BlobClient(connectionString, ContainerName, blobname);

await.blobClient.DownloadToAsync(filePath) ;

Console.WriteLine(" blob downloaded ");

**.NET - Blob metadata**

Metadata -> is a key value pair . Example Department IT.

**How to set metadata from .Net program**

Using Azure.Storage.Blobs;

string connectionString ="constrfromAzureStoreAccount";

string containerName ="DesiredContainerName";

string blobName="script.sql ";

BlobClient bc = new BlobClient(connectionString, ContainerName, blobname);

async Task SetBlobData()

{

BlobClient bc = new BlobClient(connectionString, ContainerName, blobname);

IDictionary<string,string> metaData = new Dictionary<string,string>();

metaData.Add("Department","IT");

metaData.Add("Department","HR");

await bc.SetMetadataAsync(metaData);

Console.WriteLine("metadata added! ");

}

async Task GetMetaData();

{

BlobProperties blobProperties = await blobClient.GetPropertiesAsync();

foreach(var metaData in blobProperties.Metadata)

{

console.WriteLine("The Key is {0}". medataData.Key);

console.WriteLine("The Value is {0}". medataData.Value);

}

.NET - Blob lease

1. If an update is performed on a blob object by a webapp/ any program then that application will acquire a lock so that no other application is allowed to change that blob file.

2. How to acquire an exclusive lock from an application.

async Task AcquireLease()

{

string blobName ="script.sql";

BlobClient blobClient = new BlobCLient(connectionString, containerName, blobName);

BlobLeaseClient blc = BlobClient.GetBlobLeastClient();

TimeSpan leaseTime = new TimeSpan(0,0,1,00);

Response<BlobLease> response = await blobLeaseClient.AcquireAsync(leaseTime);

Console.WriteLine("Lease Id is {0}", response.Value.LeaseId);

}

When we run this program and then go to the blob we can see message "This blob has an active lease and can not be deleted or edited".

**AZ-Copy tool**

1. Download az-copy executable from internet and install it

Note: Authorize AzCopy can be done by Azure AD and SAS.

//To create a container

azcopy make "https://appstore4040.blob.core.windows.net/tmp?SASKeyofAzureStorageAccount" // This is SAS of our storage account

copy paste the above lines in az copy. go to path where azcopy executable is resides. then open commande prompt then paste the above commands

// Upload a blob format is https://storageaccount/containername/filestoupload/SASaccesskey

azcopy copy "script.sql" "https://appstore4040.blob.core.windows.net/tmp?SASKeyofAzureStorageAccount" // This is SAS of our storage account

// Download blob format is https://storageaccount/containername/filestoupload/SASaccesskey "Downloadfilename"

azcopy copy "https://appstore4040.blob.core.windows.net/tmp?SASKeyofAzureStorageAccount" // This is SAS of our storage account "script.sql"

**Moving a storage from one region to another**

1. To move a storage account, create a copy of your storage account in another region. Then, move your data to that account by using AzCopy, or another tool of your choice.

Export a template.

Modify the template by adding the target region and storage account name.

Deploy the template to create the new storage account.

Configure the new storage account.

Move data to the new storage account.

Delete the resources in the source region.

Prerequisites

1. Export template can be found in Storage account -> Automation -> Export template

In the downloaded template we can change the location of the file and execute that in template deployment

**Azure Blob Change feed**

1. purpose - It provides an ordered, guaranteed, durable, immutable read-only log of changes.

2. Operation - Audit log of events to Blob data - Create, Update, Delete

3. The Change feed data is stored ina container name $blobchangefeed (under the same storage account) .

4. Record format - The records are stored in Apache Avro format

How to enable this?

-----------------------------------------------------------------

1. Storage Account -> Data protection -> Tracking -> Enable blob change feed -> (Keep track of create, modification and delete changes to blobs in your account.

\* Keep all logs

\* Delete change feed log after (in days) -- you are now having logs that are going to be stored in the same storage account.

**What is Azure Table storage**

1. This is a service that is used to store non-reational structured data

2. Based on structured NoSQL data

3. You follow a key/attribute store with a schemaless design

Basically, we dont want to keep any relationship between table to store duplicate data and using complex joins to retrieve data and all. We just need to store data and retrieve it in a faster manner.

\* A table is a collection of entities

\* The entities don’t abide by any schema

\* Each entity can have a different set of properties

\* An entity is made up of properties

\* Each property is name-value pair

**Entity -- ROW**

Partition key - This is a string value, this identified the partition that the entity belong to

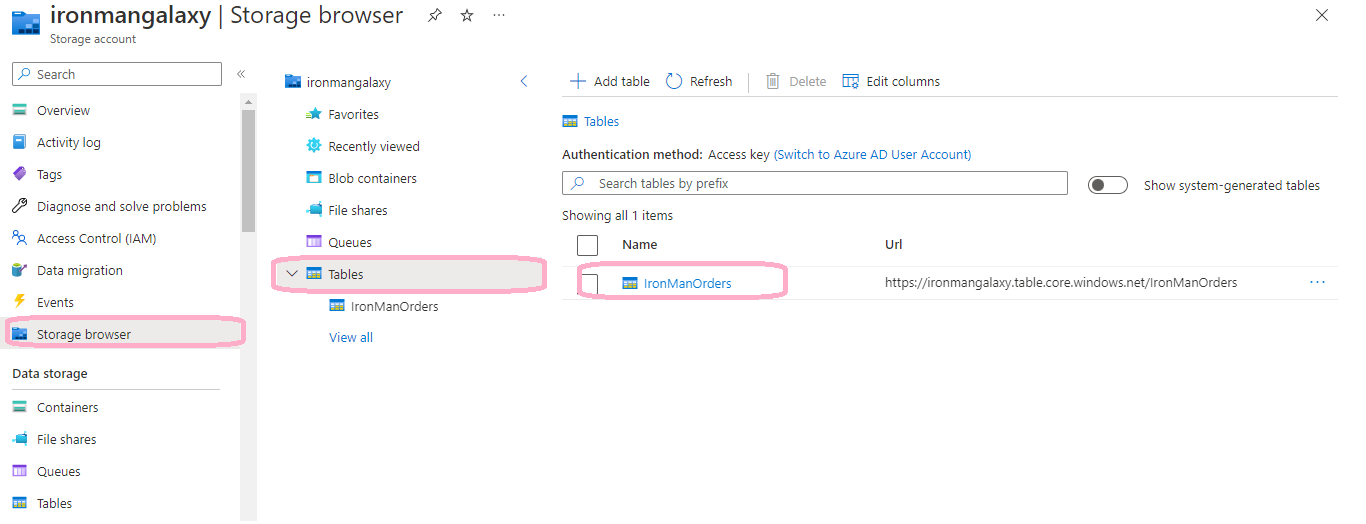
Row key - This is a string value. This uniquely identifies each entity within the partition

The partition key along with the row key helps to uniquely identify the entity within the table.

**Lab - Azure Table storage**

* 1. Storage account -> Tables -> Create table -> Orders

2. Storage browser -> Tables -> Orders -> Add Entity -> We need to enter Partition key and row key. O1 (Rowkey), MObiles(partitionKey), Quantity. These are three property



Graphical user interface, text, application

Description automatically generated

3. As this is schemaless, next entity may have additional property also, example, customerid is added additionally alone with row, partition key, quantity.

**.Net - Azure Table storage - Add Entity.**

1. Install Azure.Data.Tables from nuget.

string constr = "";

string tableName ="Orders";

AddEntity("01","Mobiles",100);

AddEntity("02","Laptop", 200);

AddEntity("03","Mobiles",300);

AddEntity("04","Laptop",300);

void AddEntity(string orderId, string category, int quantity)

{

TableClient tc = new TableClient(constr,tableName)

TableEntity te = new TableEntity(category,orderId)

{

{"quantity",quantity}

};

tc.AddEntity(tableEntity);

Console.WriteLine("Added Entity with order ID {0}", orderId);

}

void QueryEntity()

{

TableClient tableClient = new TableClient(constr,tableName);

Pageable<TableEntity> results = tableClient.Query<TableEntity>(entity => entity.PartitionKey==category);

foreach(TableEntity tableentity in results)

{

Console.WriteLine("Order Id {0}", tableEntity.RowKey);

Console.WriteLine("Quantity is {0}",tableEntity.GetInt32("quantity"));

}

}

DeleteEntity(string category, string orderId)

{

TableClient tc = new TableClient(constr,tableName);

tc.DeleteEntity(category, orderId);

Console.WriteLine("Entity Deleted");

}

**Section 8**

**What is Azure cosmos DB?**

1. Azure Cosmos DB

2. Fully managed NoSQL database

3. you get single digin millisecond response times

4. Scales automatically on demand.

\* SQL API - json based data and sql cinnabds

\* MongoDB -

\* Gremlin - graph based database

\* Cassandra

\* Table

1.Create databased account and choose type of api

2.Database

3.Container

4. Items

Database Term Container term Item Term

SQL API Database Container Item

MongoDB Database Collection Document

Gremlin Database Graph Node/Edge

Cassandra Keyspace Table Row

Table Database Table Item

**Partitions**

1. Logical partitions - Items in a container are divided into subsets called logical partitions

2. Partition key - The partition for the item is decided by the partition key that is associated with the item in the container

3. Item id - Each Item also has an item id. This helps to uniquely identity an item in the partition.

4. Identification - The combination of the partition key and the item id helps to uniquely identify the item within the container

5. Size - Each logical parition can grow up to 20 GB

6. Limit - There is no limit of number of logical partitions within the container

**Partition Key**

\* Choose a property to be a partition key which value wont change

\* The property should have a wide range of possible values

\* Once you decide on the parition key for a container, you CAN'T CHANGE IT

**Costing**

Request Units

\* The cost of database operation is measured in terms of Request units

\* Request Units is a blended measure of CPU, IOPS and memory

\* Request units are used no matter which API you are using for Cosmos DB account

\* The cost of reading a single item (1KB) is 1 request unit

\* Other operations have their own measure of charged request units

\* For each operation, you can see the amount of request units that were used for that operation

**Provisioned Throughput**

1. Here you provision the number of request units. You can always increment or decrement the number of request units provisioned.

2. You are billed on an hourly basis

3. You can provision throughput at the container or database level

**Serverless mode**

1. you dont providion any throughput

2. This is managed by Azure cosmos DB

3. You will be billed based on the Request Units you consume

**Autoscale Mode**

1. the Request units can automatically scale based on demadn

2. The demand is checked both at the container and database level.

3. Great for mission critical workloads.

**Creating new Azure Cosmos DB account**

1. Create cosmos db account then create container

2. Partition key -> /category id (partition key should have / in the beginning) -> OK

3. Go to CosmosDB -> Order (table name) -> Items -> New ITem -> { "orderId":"1", "category":"Laptop","quantity":100} -> Upon saving we will notice some new properties will be added

{

"orderId":"1",

"catgory":"Laptop",

"quantity":100,

"id":"sysgenerated id",

"\_rid":"xys",

"\_self":"",

"\_etag":"",

"\_attachments":"attachments/",

"\_ts":160102002,

}

Partition key should have variety of values, means it should be the criteria column where we can use group by. In our example orderid is not recommended. Because each order id we will be creating a partition. Category is ideal candidate.

Objects Within objects.

Order will have customer information, in ideal sql server scenario we will have customer table and we will join and fetch. In the Cosmos DB we are going to create an object within object. Like below

{

"OrderId":"1",

"category":"Laptop",

"quantity":100,

"customer":{

"customerId":"c1",

"customerName":"pravin"

}

we can query like below

SELECT o.orderId, o.category,o.quantity, o.customer.customerName form Orders o

**.NET - Create Database and container**

1. Create console application and Install Microsoft.Azure.Cosmos package from nuget

2. program.cs

string cosmosEndpoint="URLFromAzureCOsmosDBAccount";

string cosmosDBKey ="Primary/secondarykeFromAzureCosmosDBAccount";

CreateDatabase("appDB");

CreateContainer("appDB","Orders","/category");

aysnc Task CreateDatabase(string databaseName)

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

await cc.CreateDatabaseAsync(databaseName);

Console.WriteLine("Database Created");

}

async Task CreateContainer(string databaseName, string containerName, string parititonKey);

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

await db.CreateContainerAsync(container, parititonKey);

Console.WriteLine("Container created");

}

**Adding an Item**

string cosmosEndpoint="URLFromAzureCOsmosDBAccount";

string cosmosDBKey ="Primary/secondarykeFromAzureCosmosDBAccount";

string databaseName ="appDB";

string containerName="Orders";

AddItem("1","Laptop",100);

AddItem("1","Mobiles",20);

AddItem("1","Desktop",20);

AddItem("1","Laptop",25);

async Task AddItem(string orderId, string category, int quanity)

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

Container c = db.GetContainer(ContainerName);

Order order = new Order()

{

id=Guid.NewGuid().Tostring(),

orderId = orderId,

category = category,

quantity = quantity

}

ItemResponse<order> response = await container.CreateItemAsync<Order>(order, new PartitionKey(category));

Console.WriteLine("Added item with Order Id {0}", orderId);

Console.WriteLine("Request Units {0}", response.RequestCharge);

}

When we execute the above program we will get "id" required property is missing error. When we adding item from Azure automatically Id property is added by Azure. It is not the case with .net program.

**.net - Reading item**

async Task ReadItem()

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

Container c = db.GetContainer(ContainerName);

string query = "SELECT O.orderId,o.category,o.quantity from orders o"

QueryDefinition qd = new QueryDefintion(query);

FeedIterator<Order> feedIterator = container.GetItemQueryIterator<order>(qd);

while(feedIterator.HasMoreResults)

{

FeedResponse<Order> feedResponse = await feedIterator.ReadNextAsync();

foreach(Order order in feedResponse)

{

Console.WriteLine("Order Id {0}", order.orderId);

Console.WriteLine("Category {0}", order.category);

Console.WriteLine("quantity {0}", order.quantity);

}

}

}

**Replace Item**

async Task ReadItem()

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

Container c = db.GetContainer(ContainerName);

string orderId ="01";

string query = "SELECT O.orderId,o.category,o.quantity from orders o where o.orderId='{orderId}'"

string id="";

string category="";

QueryDefinition qd = new QueryDefintion(query);

FeedIterator<Order> feedIterator = container.GetItemQueryIterator<order>(qd);

while(feedIterator.HasMoreResults)

{

FeedResponse<Order> feedResponse = await feedIterator.ReadNextAsync();

foreach(Order order in feedResponse)

{

id=order.id;

category=order.category;

}

}

ItemResponse<Order> response = await container.ReadItemAsync<Order>(id, new PartitionKey )

var item = response.Resource;

item.quantity = 150;

await container.ReplaceItemAsync<Order>(item, id, new PartitionKey(category));

Console.WriteLine("Item is updated");

}

**Delete Item**

async Task DeleteItem()

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

Container c = db.GetContainer(ContainerName);

string orderId ="01";

string query = "SELECT O.orderId,o.category,o.quantity from orders o where o.orderId='{orderId}'"

string id="";

string category="";

QueryDefinition qd = new QueryDefintion(query);

FeedIterator<Order> feedIterator = container.GetItemQueryIterator<order>(qd);

while(feedIterator.HasMoreResults)

{

FeedResponse<Order> feedResponse = await feedIterator.ReadNextAsync();

foreach(Order order in feedResponse)

{

id=order.id;

category=order.category;

}

}

ItemResponse<Order> response = await container.ReadItemAsync<Order>(id, new PartitionKey )

await container.DeleteItemAsync<Order>(item, id, new PartitionKey(category));

Console.WriteLine("Item is updated");

}

**.Net - Array of objects - Adding item**

await AddItem("c1","pravin","New York",

new List<ORder>() {

new Order

{

orderId="01",

category="Laptop",

quantity=100

}

new Order

{

orderId="03",

category="Mobile",

quantity=200

}

}

async Task AddArrayItem(string customerId, string customerName, string customerCity, List<Order> order)

{

CosmosClient cc = new CosmosClient(cosmosEndpoint,cosmosDBKey );

Database db = cc.GetDatabase(databaseName);

Container c = db.GetContainer(ContainerName);

Customer customer = new Customer()

{

customerId = customerId,

customerName = customerName,

customerCity = customerCity,

orders = orders

};

container.CreateItemAsync<Customer> (customer, new PartitionKey(customerCity));

Console.WriteLine("Item is added",customerId );

}

**Stored procedures**

In cosmos DB stored procedures are written in javascript.

Stored procedure id = Display

function()

{

var context = getContext();

var response = context.getResponse();

response.SetBody("Welcome to Stored procedure");

}

string cosmosEndpointUri ="";

string cosmosDBkey = "";

string databaseName="appDB";

string containerName="orders";

await CallStoredProcedure();

{

CosmoClient cosmosClient = new CosmosClient(cosmosEndpointUri, cosmosDBKey);

Container container = cosmosCLient.GetContainer(databaseName, containerName);

var result = container.Scripts.ExecuteStoredProcedureAsync<string>("Display", new PartitionKey(""),null )

Console.WriteLine(result);

}

**Stored Procedure - Add an item**

funcation createItems(items)

{

var context = getContext();

var response = context.getResponse();

if(!items)

{

response.setBody("Error: items are undefined");

return;

}

var numOfItems = items.length;

checkLength(numOfItems);

for(let i=0;i<numofItems;i++)

{

createItem(item[i]);

}

}

function checkLength(itemLength)

{

if(itemLength==0)

{

response.setBody("Error: items are undefined");

return;

}

function createItem(item)

{

var collection = context.getCollection();

var collectionLink = collection.getSelfLink();

collection.createDocument(collectionLink,item);

}

}

string cosmosEndpointUri ="";

string cosmosDBkey = "";

string databaseName="appDB";

string containerName="orders";

CosmoClient cosmosClient = new CosmosClient(cosmosEndpointUri, cosmosDBKey);

Container container = cosmosCLient.GetContainer(databaseName, containerName);

dynamic[] orderItems = new dynamic[]

{

new

{

id=Guid.NewGuid().ToString(),

orderid="01",

category="Laptop",

quantity=30

},

new

{

id=Guid.NewGuid().ToString(),

orderid="02",

category="Mobile",

quantity=40

}

}

var result = container.Scripts.ExecuteStoredProcedureAsync<string>("Display", new PartitionKey("Laptop"), new[] orderedItem )

Console.WriteLine(result);

**Triggers**

Triggers we have pre and post triggers. We can do that pre and post triggers in cosmos DB as well, it will be in javascript function.

Trigger Type -> Pre/Post

Trigger Operation -> Create / Delete/ Replace

Trigger Body ->

function ValidateItem()

{

var context = getContext();

var request = context.getRequest();

var item = request.getBody();

if(!("quantity" in item) )

{

item["quantity"]=0;

}

request.setBody(item)

}

string cosmosEndpointUri ="";

string cosmosDBkey = "";

string databaseName="appDB";

string containerName="orders";

CosmoClient cosmosClient = new CosmosClient(cosmosEndpointUri, cosmosDBKey);

Container container = cosmosCLient.GetContainer(databaseName, containerName);

dynamic[] orderItems = new dynamic[]

{

new

{

id=Guid.NewGuid().ToString(),

orderid="01",

category="Laptop",

quantity=30

};

container.CreateItemAsync(orderItem, null, new ItemRequestOptions { PreTriggers = new List<string> {"validateItem"}})

// we can call multiple triggers, in the above example just calling one trigger.

**Change feed**

Record Changes -> Here changes to the container are recorded in the order that they occur.

Feed - The feed consists of inserts and updates. Deletes are not recorded.

Process change feed - you can process the change feed with the use of Azure Functions or change feed processor.

Records - The records for the change feed are written to another container

Sorting - The change feed is sorted in the order of modification within each logical partition key values.

Throughtput 0 the same provisioned throughput can be used to read from the container containing the changes.

**Lab Change feed - Azure functions**

Create new Azure function -> Cosmos DB account connection (select Cosmos DB connection name)

Database name -> appDB

Collection -> orders

Collection name for leases -> leases

Create lease cllection if it does not exist

// Azure function code.

using System;

using System.Collection.Generic;

using Microsoft.Azure.Documents;

public static void Run(IReadOnlyList<Document> input, Ilogger log)

{

if(input !=null && input.Count > 0)

{

log.LogInformation("Documents modified " + input.Count);

log.LogInformation("First document id " + input[0].Id);

}

}

**Change Feed Processor**

1. Monitored Container - This has the data from which the change feed is generated.

2. Lease container - This stores the state and coordinates the processing of change feed across multiple workers.

3. Computer Instances - This hosts the change feed processor that is used to listen for changes.

4. Delegate - This is the code that is used to process the batch of changes.

Change Feed Processor

\* The change feed processor library automatically check for change

\* If the changes are foud they pushed to the client

\* If you have enough throughput and lot of changes that need to be read, you can have multiple clients to read the change feed.

.net program.cs

----------------------------

using CosmosDB;

using Microsoft.Azure.Cosmos;

string cosmosEndpointUri = "";

string cosmosDBKey ="";

string databasename= "appDB";

string monitoredContainerName ="Orders";

string leaseCOntainerName ="leases";

await StartChangeProcessor();

async Task StartChangePRocessor()

{

CosmosClient cc = new CosmosClient(cosmosEndpointUri,cosmosDBKey );

Container container = cc.GetContainer(databasenmae, leasecontainerName);

ChangeFeedProcessor cfp = cosmosCLient.GetContainer(databaseName, monitoredContainerName )

.GetChangeFeedProcessorBuilder<Order>(processorName: "ManageChanges", onChangesDelegate , onChangesDelegate: ManageChanges )

.WithInstance("appHost")

.WithLeaseContainer(leasecontainer)

.Build()

Console.WriteLine("Starting the change feed processor");

await changeFeedProcessor.StartAsync();

Console.Read();

await changeFeedProcessor.StopAsync()

}

static async Task ManageChanges(

ChangeFeedProcessContext context, IReadOnlyCollection<Order> itemCollection,

CancellationToken cancellationToken)

{

foreach(Order item in itemCollection)

{

Console.WriteLine("Id {0}", item.id);

Console.WriteLine("Order Id {0}", item.orderId);

Console.WriteLine("Creation time {0}", item.creationTime);

}

}

**Composite Indexes**

1. SELECT \* FROM C ORDER BY C.category; this will run. If we order by both category and quantity then we will get the below error. "The order by query does not have a corresponding composite index that it can be served from."

2. Order container -> Settings -> Indexing policy - > add the below property

"compositeIndexes" :[

[

{

"path": "/category",

"order":"ascending"

},

{

"path": "/quantity",

"order":"ascending"

}

]

]

now the query with both category and quantity oreder by will work.

**Time To live**

Delete Items - Here items from the container can be deleted automatically after certain period of time.

Level - This can be set at the item level or container level.

Deletion process - Item will be deleted based on the Time to live that is set in seconds.

Background task - The deletion process is done as a background task that uses the left-over Request Units.

For the item - For the TTL to work at the item level, the setting should also be defined at the container level

Benefit - Here the process of deletion is managed by Azure Cosmos DB.

Orders -> Settings -> Time to live - 10 seconds

**Consistency**

1. Azure Cosmos DB account -> Settings -> Default Consistency

1. Strong

2. Bound Staleness

3. Session

4. Consistent prefix

5. Eventual

Replicate Data globally -> Write Region West US. But if our customers more mostly reside in Centra US then we can replicate the data to Central US so that the latency will be less.

But it will cost us for additional RU/s and storage cost. Remember write will be happening only from West US, Central US is our read regions.

If user writes data from one region then the data should be replicated, before that if some user reads the data then it will not be latest. We have different consistency levels.

Strong - The reads are guananteed to return the most recent committed version. This consistency will give more latency and less throughput. Because till the data is replicated application is not allowed to read data from it.

Bounded staleness - Here the reads might lag behind writes by at most K versions of an item or T time interval.

Maxmimum Lag(time) Days/Hours/Minutes/Seconds or Maximum Lag (Operations)

Session - Here within a single client session, the reads are guaranteed to honor the consistent prefix, monotonic reads and writes, read-your-writes and write-follows-read guarantees.

Consisten prefix - Here client will not see out of order writes.

Eventual - Eventually the data **will** be consistent. But there is no order gurantee for the reads.

**Section 9 - Implementing Azure security**

**Azure Active Directory?**

Azure Active directory is an identity provider and access management service.

Usage: This can be used as an identity provider for Azure, Microsoft 365 and other SaaS products as well.

You can manage identities such as users, groups and even applications

Manage security aspects when it comes to your identities

Pricing

Free - User and group management.

Premium P1 - Dynamic groups, hybrid identities, self-service password reset for on-premise users.

Premium P2 - Azure AD identity protecion and privileged identity Management.

**Role Based Access control**

A user or Application or Azure object first needs to authenticate itself to tell who it is and then role based authentication determine what that can access

Some Predefined roles are below

Contributor

Owner

Reader

**What is Application Object?**

Lets say we want our application to access Storage account. We have two ways to achieve this

1. save access keys of storage account in our application then application will authenticate itself

2. We can create appilcation object and that object will be given some access then it will authenticate and access storage account. Advantage of using application object is that we can give multiple storage account access to our application objects.

Microsoft Identity platform

The platform helps to build applications that users and customers can sign in using their Microsoft identities or social accounts

Azure Active Directory -> App registrations -> New registration - > Enter name "BlobApp" and click Register

Go to Storage account -> Access control -> Add role assignment -> Reader role -> Select member -> select "BlobApp" -> Review and assign

Go to Storage account -> Access control -> Add role assignment -> Storage Blob Data Reader role -> Select member -> select "BlobApp" -> Review and assign

We need couple of properties to connect with bolb now.

Application / Client ID : Overview section of Application object

Tenant ID : Overview section of Application object

Client secret : Certificates and secrets section of Application object

Certificates and secrets -> Create new client secret -> Now you will see some secret please save it in secure place, once you navigate away you cannot see this again.

String clientId ="";

string tenantId="";

string clientSecret="";

string blobUri="";

string filePath="c:/temp/";

ClientSecretCredentials clientSecretCredential = new ClientSecretCredentials(tenantId, clientId, ClientSecret);

BlobClient blobClient = new BlobClient(new Uri(blobUri, ClientSecretCredentials );

await blobClient.DownloadToAsync(filepath);

console.WriteLine("The blob downloaded")

-- should instal Azure.Identity;

**Microsoft Graph**

Is an API https://graph.microsoft.com to provide access to rich, people-centric data and insights in the microsoft lcoud including Microsoft 365, Windows, and Enterprise Mobility + security.

We are going to see how to invoke graph API using application object. In previous example, we were able to create application object and then on then blob storage account we give access control to that application object.

But in order use this API using postman tool, postman is an external tool. For that we need to go application object -> API permissions

By default Microsoft Graph -> User.Read (Delegated Type ) permission

**Types of Permission**

1. Application Permission (Runs on behalf of the application)

2. Delegated permission (Runs on behalf of the user )

For our graph api we delete existing default permission and create application permission and -> User -> User.Read.All (Earlier it was User.Read)

Status -> Not granted for Default -> Click on Grant admin consent for Default Directory

1. Register an application in Azure AD

2. Provide permissions

3. Provide Admin consent

4. Call the Microsoft authentication service

5. Get an access token

6. Accessa resource using the token

how to get access key?

Create a post request in postman.

Go to postman application object -> Overview -> Oauth 2.0 token endpoint (v2)

Enter URL to postman

Notice URL: https://login.microsoftonline.com/70c0xzy/oauth2/v2.0/token -- after domain we see some random string this is directory

x-www-form-urlencoded

Key value

grand\_type client\_credentials

client\_id from postman app object overview

client\_secret Create new secret from postman and copy here

scope https://graph.microsoft.com/.default

Send- >

Token type is Bearer token

To access graph API

Create a get request from postman as below

Get : https:/graph.microsoft.com/v1.0/users

Headers

Authorization Bearer "Token copied from post request"

Note: even there is proper permission isn't provided also we can get permission from microsoft, but that token wont have permission to access your resources.

Update user

1. Add Users.Write.All permission

2. Add -> Click on Grant admin consent for Default Directory

3. Generate token (to get updated permission

4. Make a patch call -> https:/graph.microsoft.com/v1.0/users/userid

5. Add header values with Authorization token

6. Body -> Raw -> JSON {"givenName":"Username"}

**Key Vault**

We have sensitive information like

\* Encryption keys

\* Certificates

\* Secrets

We need a safe way to store this, usually we need special software to store that and server to maintain that software.

We have Azure Key vault

1. Create new key vault

2. Go to Keys -> Create Generate Import Key

3. Create new project

4. Install Azure.Security.KeyVault.Keys

5. Create new app registration

6. Appvault -> Access Policies -> Add Policy -> Key Permission (get, Encrypt, Decrypt)

Select Principal -> Keyapp -> Add -> Save

Now this permission is

tring clientId ="";

string tenantId="";

string clientSecret="";

string KeyVaultURL=""; -- from Keyvault overview section

string keyName="appkey"; --name of your key in azure

string textToEncrypt ="This is a secret text";

ClientSecretCredentials clientSecretCredential = new ClientSecretCredentials(tenantId, clientId, ClientSecret);

KeyClient keyClient = new KeyClient(new Uri(keyVaultURL));

var key = keyClient.GetKey(keyName);

var cryptoClient = new CryptographyClient(key.Value.Id, clientSecretCredential );

byte[] textToBytes = Encoding.UTF8.GetBytes(textToEncrypt);

EncryptResult result = cryptoClient.Encrypt(EncryptionAlgorithm.RsaOaep, textToBytes);

console.WriteLine("The Encrypted string ")

console.WriteLine(Convert.ToBase64String(result.Ciphertext));

byte[] cipherTOBytes = result.Ciphertext;

DecryptResult textDecrypted = cryptoClient.Decrypt(EncryptionAlgorithm.RsaOaep,cipherTOBytes )

console.WriteLine("The Decrypted string ")

console.WriteLine(Encoding.UTF8.GetString(textDecrypted.Plaintext);

**Secrets**

1. Install Azure.Security.KeyVault.Secrets

String clientId ="";

string tenantId="";

string clientSecret="";

ClientSecretCredentials clientSecretCredential = new ClientSecretCredentials(tenantId, clientId, ClientSecret);

string keyVaultUrl="";

string secretName ="dbConnectionString" -- secret name

SecretClient secretClient = new SecreClient(new Uri(keyVaultUrl), clientSecretCredential)

var secret = secretClient.GetSecret(secretName );

string connectionString = secret.value.value;

In Azure, Secret should have Get permission.

**Managed Identities**

A common challenge for developers is the management of secrets, credentials, certificates, and keys used to secure communication between services. Managed identities eliminate the need for developers to manage these credentials.

While developers can securely store the secrets in Azure Key Vault, services need a way to access Azure Key Vault. Managed identities provide an automatically managed identity in Azure Active Directory for applications to use when connecting to resources that support Azure Active Directory (Azure AD) authentication. Applications can use managed identities to obtain Azure AD tokens without having to manage any credentials.

If you host your application on Azure VM, then you can make use of Managed identity, where you dont have to store anything including client secret, client ID etc. This managed identity will be given access to the azure resources. This works on azure services. If your application is hosted on outside Azure environment we can use Application object.

1. Configure managed Identity for your VM , Go to VM -> Identity -> System assigned -> Status -> On -> Save

(you can see this alert appvm will be registered with Azure Active directory. Once it is registered, appvm can be granted permissions to access resources protected by Azure AD. Do you want to enable the system assigned managed identity for appvm?)

2. Storage account -> Access control -> Add role assignment -> add memeber -> select appvm (which is our managed identity)

3. TokenCredential will take access token from various places like Environment, ManagedIdentity, AzureCli, SharedTokenCache etc.

4. Download RDP for VM, right click and edit -> Local Resources -> More -> Drives -> C:\ (so local C drive files are copied to VM )

string blobUri="";

string filePath="c:/temp/";

TokenCredential tokenCredential = new DefaultAzureCredential()

BlobClient blobClient = new BlobClient(new Uri(blobUri, tokenCredential );

await blobClient.DownloadToAsync(filepath);

console.WriteLine("The blob downloaded")

-- should inclue azure.core

**Managed Identitiy - Getting access token**

We are going to use Metadata API to get the access token, this is the way to get access token from ManagedIdentity

169.254.169.254

string token Uri = "http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=http://storage.azure.com"

HttpClient client = new HttpClient ();

client.DefaultRequestHeaders.Add("Metadata","true");

HttpRespoinseMessage response = await client.GetAsync(tokenUri);

string content = await response.Content.ReadAsStringAsync();

Dictionary<string, string> values = JsonConvert.DeserializeObject<Dictionary<string, string>(content);

foreach(KeyValuePari<string, string> pair in values)

{

Console.WriteLine("key "+ pair.Key);

Console.WriteLine("Value "+ pair.Value);

}

string BlobUri ="";

HttpClient BlobClient = new HttpClient();

BlobClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", values["access\_token"]);

bolbClient.DefaultRequestHeaders.Add("x-ms-version","2017-11-09"); // To do a blob call, we need to provide mentioned information, refer microsoft documentation

HttpRespoinseMessage blobresponse = await BlobClient.GetAsync(BlobUri );

string blobContent = await blobResponse.Content.ReadAsStringAsync();

console.WriteLine(content);

**Azure Managed Identity**

Instead of giving clientId, secret etc, we can make use of Managed Identity for Product Service project.

App service -> Identity -> System Assigned -> on -> save

In Azure keyvalue my connection string is saved.

Key Vault -> Access Policies -> Get access -> Select principal -> sqlapp (appservice name) ->save

**Managed Identity Types**

1. System Assigned (this is assigned on the resource itself, like VM, Resource group etc)

2. User Assigned (This is a seperate resource)

Use of User assigned identity is even when the resource is deleted this identity will stay. And we can assign one identity to multiple resources

You have to create managed identity and assign it to VM, then on the Storage account we need to give read and blob read permission to that managed identity.

**User Assinged Identity - Powershell**

-- on terminal Install-Module Az.ManagedIdentity

Connect-AzAccount

$ResourceGroupName="app-grp"

$IdentityName="app-identity"

$VmName ="appvm"

New-AzUserAssignedIdentity -ResourceGroupName $ResourceGroupName -Name $IdentityName -Location "North Europe"

$Identity=Get-AzResource -Name $IdentityName -ResourceGroupName $ResourceGroupName

$Id=$Identity.Id

$vm=Get-AzVm -ResourceGroupName $ResourceGroupName -name $VmName

Update-AzVM -ResourceGroupName $ResourceGroupName -VM $Vm -IdentityType UserAssigned -IdentityId $Id

**Managed Identity - Powershell**

Connect-AzAccount

$ResourceGroupName="app-grp"

$VmName="appvm"

$Vm=Get-AzVM -ResourceGroupName $ResourceGroupName -Name $VmName

Update-AzVm -ResourceGroupName $ResourceGroupName -VM $vm -IdentityType SystemAssigned

**PowerShell - Storage Account**

$StorageAccountName = "appstore5050"

$ResourceGroupName = "app-grp"

$StorageAccountKey= (Get-AzStorageAccount -ResourceGroupName $ResourceGroupName -AccountName $StorageAccountName) | Where-Object {$\_.KeyName -eq "key1"}

$StorageAccountKeyValue= $StorageAccountKey.Value

$KeyVaultName = "appvault20303"

$SecretValue = ConvertTo-SecureString $StorageAccountKeyValue -AsPlainText -Force

Set -AzKeyVaultSecret -VaultName $KeyVaultName -Name $StorageAccountName -SecretValue $SecretValue

**10 Implement Azure Security - Authentication and Authorization**

**Microsoft Identity Platform**

Where Azure Active Directory is one of the Identity provider, we are not restricted to use only AAD. We can also make use of other external identity providers like Facebook, Gmail, LinkedIn etc

Basically we are going to get a token from external identity providers and provide access based on the token.

**Microsoft Authentication Library**

--------------------------------

Enable developers to acquire access tokens from the Identity platforms

This can be used to authenticate users and allow secure access to APIs

It also maintains the token cache and refresh tokens when they are about to expire.

**OAuth2**

An application is accessing some resource on behalf of the user is what OAuth is all about. This process is called Authorization work flow.

oauth.net/2/ will have more information including grant types

\* Authorization code

\* PKCE

\* Client Credentials

\* Device code

\* Refresh token

Click on client credentials you can notice what is required to make client credentials like grant\_type

POST /token HTTP/1.1

Host: authorization-server.com

grant\_type=client\_credentials

&client\_id=xxxxxxxxxx

&client\_secret=xxxxxxxxxx

This is what we followed when we use Azure Active directory

**OAuth2 - Authorization Code Grant**

Resource Owner - This is the user who has access to the protected resource

Client - This is the application requesting access to the resource

Resource Server - In Azure, this can be Web API that will allow access into the Azure resource

Authorization Server - Microsoft Identity platform is the authorization server

How does Authorization code flow works?

1. Application makes a call to authorization server and will provide callback URI to Authorization server. When Authorization server has some response to provide it can use this call back uri

2. The authorization server sends the authorization code to the application. This authorization code is just the initial step in the process. The application then needs to use the authorization code to get the access token

(Note the authorization code is viewable in the browser, but later in the process of getting the access token with the use of the authorization code is done by the application in the backend. This is for security reason to avoid man in middle attack)

**The Authorization Code Flow**

Here’s each query parameter explained:

response\_type=code - This tells the authorization server that the application is initiating the authorization code flow.

client\_id - The public identifier for the application, obtained when the developer first registered the application.

redirect\_uri - Tells the authorization server where to send the user back to after they approve the request.

scope - One or more space-separated strings indicating which permissions the application is requesting. The specific OAuth API you’re using will define the scopes that it supports.

state - The application generates a random string and includes it in the request. It should then check that the same value is returned after the user authorizes the app. This is used to prevent CSRF attacks.

When the user visits this URL, the authorization server will present them with a prompt asking if they would like to authorize this application’s request.

**Use Authentication in Web project**

1. Create web app using .net core and add below configuration in appsettings.json

"AzureAd": {

"Instance" : "https://login.microsoftonline.com",

"TenantId" : "",

"ClientId" :""

}

2. Install Microsoft.Identity.Web

3. Program.cs -> builder.Services.AddMicrosoftIdentityWebAppAuthentication(builder.Configuration) // This will help us our application to authenticate itself automatically.

app.UseAuthentication();

4. Index.cshtml.cs -> Add authorize attribute on class level

5. Go to Application object -> authenticaton -> Add platform -> Configure platform -> Enter the redirect URL

Then Select the tokens you wold like to be issued by the authorization endpoint: -> ID tokens (used for implicit and hybrid flows)

6. Now our web application is asking us to login to Azure, lets provide credentials and login

7. Now browser is asking permissions requested. (consent on behalf of your organization)

Remember, in application object we gave "Grant admin consent for default directory". This is because postman has no ability to get this consent (we are not logged in as azure user account). But this case we did not enabled it, so our application is prompting user permission

Another important point is postman is having application Type permission, means our postman itself should have permission. In this case we have delegated permission, means on user's behalf application will authorize.

After giving our consent we are getting below error. The redirect URI https://localhost:7046/signin-oidc specified in THE REQUEST DOES NOT MATCH THE REDIRECT UriS CONFIGURED FOR THE APPLICATION

"AzureAd": {

"Instance" : "https://login.microsoftonline.com",

"TenantId" : "",

"ClientId" :""

"CallbackPath": "/signin-oidc"

"SignedOutCallbackPath": "/signout-oidc"

}

the same has to be configured Authentication section of Application object

Redirect URL

https://localhost:7046/signin-oidc

Front Channel logout url

https://localhost:7046/signout-oidc

oidc - OpenIDConnect We are using OpenIdConnect to authenticate then Oauth to authorize.

**Adding Sign-out**

1. For sign-in and sign-out we need to use the following library Microsoft.Identity.Web.UI

2. builder.Services.AddControllersWithViews()

3. Place AddRazorPages below with opions

builder.Services.AddRazorPages().AddMvcOptions(options =>

{

var policy = new AuthorizationPolicyBuilder()

.RequireAutheticatedUser()

.Build();

options.Filters.Add(new AuthorizeFilter(policy));

)}.AddMicrosoftIdentityUI();

// add required library using if error is popping up

4. Remove Authorize attribute, because we are now going to use authenticated users only

5. app.UseEndpoints(endpoings =>

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Nome}/{action=Index}/{id?}"))

6. Add new Scaffolded Item -> Go to Identity (it will add default nuget packages that are necessary for this) -> Choose Account\Login and Account\Logout -> Click on Add new Data context class -> Add SQLLite instead on SQL server.

7. Areas -> Identity -> Data (AuthAppContext)

-> Pages -> Account (Login and logout.cshtml)

8. Program.cs also sqllite configuration and identity configuration will be added.

9. To signout asp-area="MicrosoftIdentity" asp-controller="Account " asp.action="SignOut"

Getting user claims?

Claims is nothing but information about users.

In authentication we enabled ID tokens through which we will get information about users.

@using System.Security.Claims

@foreach(var \_claim in ((ClaimsIdentity) User.Identity).Claims)

{

<tr>

<td>@\_claim.Type</td>

<td>@\_claim.Value</td>

</tr>

}

**Getting Group claims**

We can create groups, add members. and give access to that group, so that whoever assigned to that group get same level of permission

If application need certain role to access the application then the application can check what group user is assigned to and get their role based on that.

To do that go to AuthApp -> TokenConfiguration -> Add Group claim -> Select group types to include in Access, ID and SAML tokens -> select "Security groups"

Customize token properties by type -> ID -> Group ID (is default )

Access -> Group ID -> Add

Go to Manifest section (it is a json file having information about our cliam) go and check groupMembershipClaims : "SecurityGroup"

No in our claim program we should have got this group ID . We can make note of group ID which we are looking for and take action based on it.

Apart from that we can also Add optional claim -> ID -> email, givenname

Turn on the Microsoft graph email, profile permission(required for claims to appear in token)

**Getting An Access token**

1. Lets say we want out applicationto access storage account to perform some action

But now, we are authenticating user and giving users permission to access storage account, but application need certain permission to access storage account.

1. To do that API permissions -> Add permission -> Azure storage -> Delegated permission (only this is possible) -> Permission -> user\_impersonation (application will act like user and get permission to access storage account)

2. Also Authentication -> Access tokens (used for implicit flows)

3. program.cs

string[] scope = new string[] {"https://storage.azure.com/user\_impersonation"}

builder.Services.AddMicrosoftIdentityWebAppAuthentication(builder.Configuration, "AzureAd")

.EnableTokenAcquisitionToCallDownstreamApi(scope);

.AddInMemoryTokenCaches();

Now we are getting token from application like below

private readonly ITokenAcquisition \_tokenAcquistion;

public string accessToken;

public IndexModel(ITokenAcquisition tokenacqu)

{

\_tokenAcquistion = tokenacqu

}

public Task OnGet()

{

string[] scope = new string[] {"https://storage.azure.com/user\_impersonation"}

accessToken = await \_tokenAcquisition.GetAccessTokenForUserAsync(scope)

}

Not its showing error "MsalClientException: One client credential type required either: ClientSecret, Certificate OR clientAssertion must be defined when creating a confidential client. ONly specify one.

Go to appsetting.json

"ClientSecret" : "secretfromAzure"

**Using Access Token**

public Task OnGet()

{

//string[] scope = new string[] {"https://storage.azure.com/user\_impersonation"}

//accessToken = await \_tokenAcquisition.GetAccessTokenForUserAsync(scope)

TokenAcquisitionTokenCredential tokenAcquisitionTokenCredential = new TokenAcquisitionTokenCredential(\_tokenAcquisition);

Uri blobUri = new Uri("");

BlobClient blob = new BlobClient(blobUri, tokenAcquisitionTokenCredential );

MemoryStream ms = new MemoryStream(ms);

blobContent = sr.ReadToEnd();

}

--install Azure.Storage.Blobs from nuget package.

Now we can get blobcontent to our web application..

In order to publish to Azure web app, we need to update sign in and signout properties in Application object with new web app url.

Publish as Zip option.

**Accessing Blob storage from Postman**

1. Select Storage account and give access to postman application object with reader and blob read access

2. Goto Postman and in the token post request change the scope alone with below

https://storage.azure.com/default

3. Send will get us the access token

4. Go to storage account we will get bolburi

5. When we give access token in the header section we will get contents from azure blob.

**Creating web api & Publish is normal process. Then add authentication**

1. Go to manifest -> accessTokenAcceptedVersion -> change null to 2 . 2 means supporting oauth2.0 , null means oauth.

2. In project install Microsoft.Identity.Web from nuget

3. Program.cs -> builder.Services.AddMicrosoftIdentityWebApiAuthentication(builder.Configuration,"")

3. In appsettings.json -> add AzureAd section with clientid, TenantId, Instance details

4. App.userAuthentication() and App.UseAuthorization();

5. Now accessing URL will give 401 error

(as our postman tool cannot login as user, we need to create an application object for API project ProductAPI)

6. Go to "Expose an API " -> Application ID URI -> SET -> will populate one api(scopes defined by this API . Define custom scopes to restrict access to data and functionlaity protected by the API. An applicationthat requires access to parts of this API can request that a user or admin consent to one or more of these.

Adding a scope here creates only delegated permissoins. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. Go to app roles.

7. Copy the api and paste it in scope when getting access token

8. GO to post man -> API permission -> select our Produt API -> Application permission (we have this ) -> Permissions -> ProductAccess -> Add permission -> Add Admin consent

Go to Product API application object and Add roles ->

**Invoking API from console application**

IcConfidentialClientApplication cca ;

string clientid="";

string clientSecret="";

string tenantId = "";

cca = ConfidentialClientApplication.Create(clientId)

.WithTnenatId(tenantId)

.WithClientSecret(clientSecret)

.Build();

string[] scopes = new string[] {"api/xy/.default"}

AuthenticationResult result = await cca.AcquireTokenForClient(scope).ExecuteAsync();

string accessTOken = result.AccessToken;

Console.WriteLine(accessToken);

string apiURL="WebAPI url";

HttpCLient client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer" , accessToken );

HttpResponseMessage responseMessage = await client.SendAsync(apiUrl);

string content = await resonpseMessage.Content.ReadAsStringAsync();

**Section 11**

**Azure Monitor**

Search -> Monitor -> Metrics -> select any resource and apply (we can see many metrics types,

\* CPU credits Consumed

\* CPU credits Remaining

\* Available Memory Bytes

\* Data Disk Bandwidth Consumed percentage

\* Data Disk IPOs Consumed Percentage

\* Data Disk Max Burst Bandwidht etc

We can save it to Dashboard

Activity Log -> This will have administrative activities are recorded here.

**Configure alerts**

1. Select scope (This is choosing your resource - VM for example)

2. -> Condition (selecting your condition, ex CPU % reach 70 #

Signal type -> (All, Metrics, Log, Activity Log) -> Metrics -> Network ->

alert logic -> Static -> Threshold value 100 B

Evaluated based on Aggregation granularity 5 mins, Frequency of evaluation 5 mins

3. Actions -> Create new action group ->

Notification type -> Email/SMS message/Push / Voice -> Give your email id

4. Details -> Severity (Informational)

Actions ->

5. After 5 mins -? there will be new alert logged in alerts section.

**Dynamic thresholds**

Azure Monitor uses machine learning to check the historical behavior or metrics.

Based on the historical data, it can then identify patterns and anomalies that could indicate possible issues.

Sensitivity

High - Here the alert rule will be triggered even for the smallest deviation.

Medium - you have more balanced threshold and fewer alerts will be generated

Low- alerts will only triggered on large deviation.

Select resource-> Condition -> CPU % -> Threshold -> Dynamic -> Threshold sensitivity -> Medium

**Log analytics**

Central solution for all of your logs

Create resource -> Log Analytics -> Create ->

How to connect your VM into log analytics

VM -> connect -> Extentions + applications -> microsoftmonitoringagent is being installed -> this agent will be sending logs to microsoft based vm

for linux it will be OmsAgentForLinux.

Agent Configuration ->

\* Windows event logs

\* Windows performance counters

\* Linux performance counter

\* Syslog

\* IIS logs

you need to wait for 20 mins to 30 mins to see the logs reflecting

Analytics -> Logs -> LogManagement -> Heartbeat (is already created)

based on the log you configured, that table will be created.

1. Add windows event log -> Logname -> Application -> Error, warning, information

2. syslog -> Add facility -> syslog -> Emergency, alert, Critical , Error, Warning, Notice, Info, Debug etc.

Azure Web app - Diagnostic

Web app-> Diagnostic settings -> We can collect (AppServiceHTTPLogs, AppServiceCOnsoleLogs, AppServiceAppLogs, AppServiceAuditLogs etc)

Destination -> Send to Log Analytics workspace

{

"$schema":"https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate"

"contentVersion":"1.0.0.0",

"parameters":{},

"functions":{},

"varibales":{},

"resources":[

{

"type": "Microsoft.Insights/actionGroups",

"apiVersion": "2021-09-01",

"name":"GroupB",

"location":"Global"

"properties":{

"description": "Alert for when the VM CPU % goes beyond 70%",

"severity":2,

"enabled":true,

"scopes": ["[resourceId('Microsoft.comute/virtualMachines','appvm')]"] // this will get resourceId of appvm

"evaluationFrequency":"PT5M",

"windowSize":"PT5M",

"criteria":{

azure-monitor/essentials/metrics-supported#microsoftcomputevirtualmachines this url will give list of metrics we can apply for virutual machine

"odata.type" :"Microsoft.Azure.Monitor.MultipleResourceMultipleMetricCriteria"

"allOf":{

"name":"CPU Criteria",

"metricName":"Percentage CPU" // taken from microsoft docuemnetation

"dimensions":[],

"operator":"GreaterThanOrEqual",

"threshold":70,

"timeAggregation":"Average"

}

]

}

}

"enabled":true,

"groupShortName":"GroupB",

"emailReceivers":[

{

"name":"AdminEmail",

"emailAddress":"techsup4000gmail.com",

"useCommonAlertSchema":false

}]

}

Create resource -> Template deployment -> Build my own template in the editor -> copy pastethe above file content

ARM Templates -> Azure Monitor alerts

{

"$schema":"https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate"

"contentVersion":"1.0.0.0",

"parameters":{},

"functions":{},

"varibales":{},

"resources":[

{

"type": "Microsoft.Insights/metricAlerts",

"apiVersion": "2018-03-01",

"name":"CPUAlert",

"location":"Global"

"properties":{

"enabled":true,

"groupShortName":"GroupB",

"emailReceivers":[

{

"name":"AdminEmail",

"emailAddress":"techsup4000gmail.com",

"useCommonAlertSchema":false

}]

}

**What is Application Insights?**

\* This provides the feature of application performance managment and monitoring of live web applications - Monitoring

\* You can see aspects such as detecting performance issues or any other Issues - Aspects

\* Support for .Net, node.js, java and python.

\* This works for application hosted in Azure, on-premises environments or other cloud platforms

\* It has Integration with the Visual studio IDE

How it works ?

You can install a small instrumentation package (SKD) for your application or use the application insights agent.

You can instrument web applications background components and javascript in web pages .

The telemetry data sent by application Insights has very little impact on the perfromance of your application.

In Nuget packages we can see that Appinsight is deprecated, we need to use latest stable version to get rid of this issue.

**Application Insights on Azure**

Go to Web app -> Turn on Application Insights -> Enable -> Create New rosource (this will create new application insight ) -> We need to select Log analytics workapace also -> Go to .net core and select different aspects -> Recommended -> Profiler (On) , Snapshot debugger (on) , SQL commands (ON)

Click on connected services -> remove SDK dependency and save. But we are going to keep telemetry code in program.cs because we need this line to use azure application insights. but SDK is not required on cloud.

Once published to cloud you can see new app insight resource is created -> Live metrics -> We can see incoming and outgoing requests.

**Performance data - Application Insights**

App insights -> Performance tab -> We can see time taken for that request and we can also see dependent request etc from this performance tab.

To view the sql script executed we can use the following code

builder.Services.ConfigureTelemetryModule<DependencyTrackingTelemetry>((module,o) => {module.EnableSqlCommandTextInstrumentation = true;});

Right click on the project -> Configure application insigts -> Application Insights SDK (local) or Azure Application insights -> Next -> It will install nuget packages and code dependency will be added

This step will install Microsoft.ApplicationInsights.dotnetcore and program.cs will have builder.Services.AddApplicationInsightsTelemetry();

Run your application -> View -> Other Windows -> Application Insights search -> Search debug session telemetry -> Search we can see different types of events.

**Application Insights -> Users, Sessions and Events**

Users - Here you can see how many people have used your applicationand its features.

Session - You can see sessions of user activity. This includes certain pages and features of the application.

Events - This gives a view of how often certain pages and featres have been used in the application.

Funnels - Here you can have multiple stages like a pipeline and then you can see how users are progressing through your application as an entire process

Cohorts - This is set of users, sessions, events or operations that have something in common. It helps to analyze a particular set of users or events.

Impact - Here you can see how load times and other aspects of your application impact the conversion rate for your application.

Retention - You can analyze how many users return back to your application.

User flows- This can help in answering useful questions such as

1. What do users click on a page within the applicaiton

2. Where are the places within the application that users chrun the most from the site.

3. Are there places in the application where the users repeat the same action over and over again.

**Application Insights - Availablity Tests**

Tests - You can define tests that can monitor the availability and responsiveness of an application

Web Requests - This feature can send web requests to your web application from different points across the world.

End point - You can setup tests against an HTTP or HTTPS endpoint that's accessible over the public internet

API - You can also test your REST API's as well

Go to application Insights -> Availability -> Add Standad (preview) test -> Test URL (app service URL ) (Parse dependent requests -- this wil test for dependent resources like image etc).

Test Frequency -> 5 mins , Test location . Lets test after 20 mins.

This is automated test for your we application availability.

**Application Insights - Tracking Users**

Now we can see that under app insights Sessions -> Users -> Authenticared user -> we can see undefined in the authenticated users.

To log authenticated user -> Create a class "TelemetryServices.cs" -> inherit from TelemetryInitializerBase class

public TelemetryService(IHttpContextAccessor httpContextAccessor) : base(httpContextAccessor){

}

protected override void OnInitializeTelemetry(HttpContext platformContext, RequestTelemetry)

{

telemetry.Context.User.AuthenticatedUserId = platformContext.User?.Identity.Name ?? string.Empty

}

**Optimizing Content Delivery**

Frequently accessed data can be saved on Redis cache for better performance.

Azure Redis Cache

Data store: This is an in-memory data store that is based on the Redis software

Server memory : Here you can store frequently accessed data in server memory.

Advantage : the advantage of server memoery is tht you can write and read data quickly.

For the application: It helps to provide low latency to your data and high throughput

**Data cache**

------------------------------------------------------------

1. We are taking very frequently accessed data from source (Azure SQL or any possible datasource) and then store it in Azure Cache for Redis then

our application directly accessing from Redis cache. Keep in mind that Redis cache and Azure SQL database wont know each other. They wont in sync, unless it is ensured by the web application. Its web apps responsibility.

**Content cache**

---------------------------------------------------------------

We store header footer and static content in the cache.

**Session Store**

-----------------------------------------------------------------

Cart item can be stored in cache.

Go to Azure Cache -> Console

Set top:course:rating 4.9

get top:course:rating

exists top:course:rating -- Integer(1) means yes, integer(0) means no.

del top:3:course

lpush top:3:courses "az-104"

lpush top:3:courses "az-204"

lpush top:3:courses "az-304"

leNFW ROP:3:COURSES 0-1

**Developing Solutions for Microsoft Azure**

Install StackExchange.Redis package from Nuget package.

string connection ="connectionstring from azure for redis cache"

ConnectionMultiplexer redis = ConnectingMultiplexer.Connect(connectionString);

Void SetCacheData()

{

IDatabase database = redis.GetDatabase();

database.StringSet("top:3:courses", "az104,az-204,az-304");

console.writeline("cache data has been set");

}

Void getCacheData()

{

IDatabase db = redis.GetDatabase();

if(db.KeyExists("top:3:courses")){

console.WriteLine(db.StringGet("top:3:courses"));

else

Console.WriteLine("Key does not exist");

}

}

**Key Evection**

We need to ensure that the cache does not fill up.

Ensure that the cache does not have stale data. There is a default policy in place to evict keys that is LRU (least recently used)

We can refer which key eviction policy is suitable for our situation

noeviction: New values aren’t saved when memory limit is reached. When a database uses replication, this applies to the primary database

allkeys-lru: Keeps most recently used keys; removes least recently used (LRU) keys

allkeys-lfu: Keeps frequently used keys; removes least frequently used (LFU) keys

volatile-lru: Removes least recently used keys with the expire field set to true.

volatile-lfu: Removes least frequently used keys with the expire field set to true.

allkeys-random: Randomly removes keys to make space for the new data added.

volatile-random: Randomly removes keys with expire field set to true.

volatile-ttl: Removes keys with expire field set to true and the shortest remaining time-to-live (TTL) value.

SetKeyExpiry(string key, TimeSpan expiry)

{

IDatabase database = redis.GetDatabase();

database.KeyExpire(key, expiry);

Console.WriteLine("Set the key expire to 30 seconds");

**Use Redis cache in Web application**

1. Install StackExchange.Redis nuget package

2. Connecting string in project

3. use multiplexer class to connect with Redis cache.

4. Add Multiplexer and Produce service dependencies

5.

public async Task<List<Product>> GetProducts()

{

List<Product> \_product\_list = new List<Product>();

IDatabase database = \_redis.GetDatabase();

string key ="productList";

if(await database.KeyExistsAsync(key))

{

long listLength = database.ListLength(key);

for(int 1=0; i<listLenght;i++)

{

string value = database.ListGetByIdent(key, i);

for (int i=0;i<listLenght;i++){

string value = database.ListGetByIndex(key, i);

Product product = JsonConvert.DeserializeOject<pRODUCT>(VALUE);

\_product\_list.Add(product);

}

return \_product\_list;

else {

string sql ="select \* from Products";

SqlConnection conn = new SqlConnection();

con.Open();

SqlCOmmand cmd = new SqlCommand(sql,con);

using (SqlDataReader reader = cmd.ExecuteReader())

{

while(reader.Read())

{

Product \_product = new Product()

{

ProductId = reader.GetInt32(0),

ProductName = reader.GetString(1),

Quantity = \_reader.GetInt32(2)

};

database.ListRightPush(keh, JsonConvert.SerializeObject(\_product));

\_prodcut\_lst.Add(\_product);

}

con.close();

return \_product\_lst;

}

}

**What is Azure Content Delivery?**

CDN profile is created at global level..

And end point will be pointing to web application

1. The user in the East US location makes a request to the CDN endpoint

2. The CDN checks whether the point of presence location closest to the user has the requested file

3. If not a request is made to the source to get the required file

4. A server in the point of present location will then cache the required file

5. The server will also send the file to the user

**CDN**

All resources -> Create resource -> Front Door and CDN profiles -> Create -> Explore other offerings -> Azure CDN standard from Verizon -> This resource is Global level -> Create End point -> name will be appended with .azureedge.net -> OrginType (Web app) -> orgin host name (Web app\_

4 - 5 mins to take effect ->

After deployment of resource we can see Endpoint in CDN profile -> Click on the endpoint we can see

Origin hostname : webapp url

Endpoint Hostname : CDN url, this one we need to share it to client . This will load the website with CDN support

**CDN caching**

We need to refresh the CDN if the data is changing

CDN profile -> Endpoint -> caching rules -> Default cache expiration duration is 7 days by default

Caching behavior -> Override -> Set expiration to 1 mins for demo purpose

\* Bypass cache

\* Override

\* Set if missing

\* Not set

After doing that we need to first purge the cache. Then wait for 1 mins we can see the data is in sync.

**Azure Front door**

Azure Front door is another content deliver tool and it has more features.

1. Content delivery

2. Web application firewall is there to protect your application.

3. We can route traffic based on certain criteria

We are creating two web apps and put them behind Azure front door.

Go to WebApp -> App service editor-> create basic HTML file. (we can directly do some changes here)

Create new resource -> Front Door and CDN profile -> Tier (standard) Endpoint -> specify name , Orgin type (app service) > Orgin host name (select web app) -> WAF (Web application Firewall policy)

Now Azure Front door is entry point front facing for user, based on condition we can route user to different server / application

Azure Front door -> Front door manager -> Routes -> Edit Origin group -> Add another orgin -> name , origin type, Hostname (another wep app) .. Now both webapps are having same priority and same weightage. we need to wait for 10 mins

If we access web endpoint of front door, we are landing in UK south, even though we have same priority to both web apps. Reason is UK south has low latency from my current location. we can check this on azure speed.com. Azure front door will identify which one is best then redirect there.

Caching - You can enable caching your endpoint routes

When delivering large files, Azure Front Door request the file from the origin in chunks of 8 MB

When a chunk of data is received, the data chunk is cached and immediately sent to the user.

you have different query string behaviors when it comes to caching.

Ignore query strings - Here the query strings from the requestor is passed to the origin for the first request and then the asset is cached. All further requests ignore the query strings until the expiry of the cached asset.

Cache every Unique URL - Here each URL is treated as unique and cached separately.

Specify cache key query string - you can decide to include/ exclude specified parameters to determine what gets cached.

Compressing is possible.

**Section 12**

1. Consider a scenario we have an application to upload videos, once the upload is over another application need to process it.

Consider we have many instances to process the video, then we should ensure each application should take unique video should not take same video by multiple applicaiton. Also If suppose application took one video to process then failed in between then there should be a way to convey other / same instances to take that video and reprocess it.

To achieve these kinds of operation requires the messaging services. Where we are decoupling the application which process the application and list of videos to be processed in a seperate services. This is where queue storage comes into picture. We can handle retry on failure and many more advanced feature available on messaging services like service bus.

**Azure Storage Queue**

1. Need a general purpose V2 account. -> Add Queue -> Add messages -> Message Text, Expires in 7 days by default ->

Click on the messages we can see the below properties. Id, Message body, Insertion time, Expiration time, Dequeue count

2. Dequeue message -> will remove FIFO order. Clear queue will remove all messages.

.**Net Sending messages**

Program.cs

using.Azure.Storage.Queues;

string connectionString="Constr from Queue -> accesskey";

string queueName="appqueue";

SendMessage("Test Message 1");

SendMessage("Test Message 2");

void SendMessage(string message)

{

QueueClient qc = new QueueClient(connectionString, queueName);

if(qc.Exists())

{

qc.SendMessage(message);

Console.WriteLine("The message has been sent");

}

}

// Peek message will not remove message from queue, but we will be able to read messages

void PeekMessage()

{

QueueClient qc = new QueueClient(connectionString, queueName);

int maxMessages = 10;

PeekedMessage[] peekMessages = qc.PeekMessages(maxMessages);

foreach(var peekMessage in peekMessages)

{

console.WriteLine(peekMessage.Body);

}

}

**Receive Message**

void ReceiveMessage()

{

QueueClient qc = new QueueClient(connectionString, queueName);

int maxMessages = 10;

QueueMessage[] queueMessages = qc.ReceiveMessages(maxMessages);

foreach(var message in peekMessages)

{

console.WriteLine(message.Body);

qc.DeleteMessage(message.MessageId, message.PopReceipt)

}

}

Please note ReceiveMessage is two step process, it will read and remove the messages from queue and if the message is not successfully processed then it should delete the message or else the message will be placed back to queue. If you see the dequeue count is 1 now. So lets add delete message in our program

**Get QueueLength**

int GetQueueLength()

{

QueueClient qc = new QueueClient(connectionString, queueName)

if(queueClient.Exists())

{

QueueProperties prop = qc.GetProperties();

return properties.ApproximateMessageCount;

}

return 0;

}

**Sending and Receiving objects**

void SendMessage(string orderid, int quantity)

{

QueueClient qc = new QueueClient(connectionString, queueName)

if(qc.Exists())

{

Order order = new Order {OrderId = orderid, Quantity=quantity};

qc.SendMessage(JsonConvert.SerializeObject(order));

Console.WriteLine("The Order information has been sent");

}

}

**Azure storage Queue - Azure functions**

1. Create new Azure Function -> Queue trigger -> Storage emulator ( Connection string setting name => connectionString, Queue Name => appqueue

public class Queue

{

[FunctionName("GetMessages")]

public void Run([QueueTrigger("appqueue",Connection="connectionString")]string myQueueItem, Ilogger log )

{

log.LogInformation($"C# Queue trigger function processed:{myQueueItem}");

}

}

At this point of time, we would receive message "The input is not a valid Base-64 string as it contains a non-base 64 character, more than two padding characters "

If the message is not delivered after defined max recount then it will be moved to originalqueueName-poison

void SendMessage(string orderid, int quantity)

{

QueueClient qc = new QueueClient(connectionString, queueName)

if(qc.Exists())

{

Order order = new Order {OrderId = orderid, Quantity=quantity};

var jsonObject = JsonConvert.SerializeObject(order);

var bytes = System.Text.Encoding.UTF8.GetBytes(jsonObject);

var message = System.Convert.ToBase64String(bytes);

qc.SendMessage(message);

Console.WriteLine("The Order information has been sent");

}

}

**Output to Table Storage**

1. Create a table called Orders

2. Create a class to represent table Order

public class TableOrder

{

public string PartitionKey {get;set;}

public string RowKey {get; set;}

}

public class Queue

{

[FunctionName("GetMessages")]

[return: Table("Orders", Connection="connectionString")]

public TableOrder Run([QueueTrigger("appqueue",Connection="connectionString")]string myQueueItem, Ilogger log )

{

TableOrder to = new TableOrder()

{

PartitionKey = order.OrderId

RowKey = order.Quantity.ToString()

}

log.LogInformation("Order Information has been written to the table");

return tableOrder;

}

}

Another way of writing the code to output to table storage

You can actually also write the below code to write to Azure Table storage

Here in the definition of the Run method, you can add a reference to the table. And then also add the output type as ICollector. Then add the table objects to the ICollector Interface reference.

using System;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Host;

using Microsoft.Extensions.Logging;

namespace QueueFunction

{

public class Queues

{

[FunctionName("GetMessages")]

public void Run([QueueTrigger("appqueue", Connection = "connectionString")]Order order, ILogger log, [Table("Orders", Connection = "connectionString")]ICollector<TableOrder> tableorder)

{

TableOrder tableOrder = new TableOrder()

{

PartitionKey = order.OrderID,

RowKey = order.Quantity.ToString()

};

tableorder.Add(tableOrder);

log.LogInformation("Order written to table");

}

}

}

**Azure Queue Message Trigger - Important Note**

Concurrecy

Concurrency

When there are multiple queue messages waiting, the queue trigger retrieves a batch of messages and invokes function instances concurrently to process them. By default, the batch size is 16. When the number being processed gets down to 8, the runtime gets another batch and starts processing those messages. So the maximum number of concurrent messages being processed per function on one virtual machine (VM) is 24. This limit applies separately to each queue-triggered function on each VM. If your function app scales out to multiple VMs, each VM will wait for triggers and attempt to run functions. For example, if a function app scales out to 3 VMs, the default maximum number of concurrent instances of one queue-triggered function is 72.

The batch size and the threshold for getting a new batch are configurable in the host.json file. If you want to minimize parallel execution for queue-triggered functions in a function app, you can set the batch size to 1. This setting eliminates concurrency only so long as your function app runs on a single virtual machine (VM).

The queue trigger automatically prevents a function from processing a queue message multiple times simultaneously.

**Azure Service Bus**

Azure Service Bus is fully managed Enterprise message broker.

Data can include structured encoded data with formats. JSON , XML, Apache avro

\* The messages in the queue are ordered

\* The messages are held in triple-redundant storage (means we wont lose our messages even in failure. redundate 3 times for backup)

\* The data is available across availablility zones if enabled

\* The messages can then be retrieved via the pull mode.

Topics

-------------------

The subscriber to the topic will receive a copy of the message sent to the topic

You can define rules which contain filters on each subscription. The filter will decide which mesages are received by the subscription.

**Creating an Azure Service bus Queue**

1. Create Service Bus -> Namespace name -> Pricing Tier (Basic -only supports Queue, Standard - supports queue and Topics)

2. Queue ->

\* Max queue size -

\* Max delivery count

\* Message time to Live

\* Lock Duration

\* Enable auto-delete on idle

\* Enable duplicate detection

\* Enable dead lettering on message expiration

\* Enable partitioning

\* Enable sessions

\* Forward messages to queue/topic

In Service Bus namespace -> Service bus Explorer -> We can send / Receive / Peek on the azure portal

**Azure Service Bus queues - .Net sending messages**

1. Create .net project and install Azure.Messaging.ServiceBus ->

2. Azure Service Bus queue -> Shared access policies -> Add -> Manage / Send /Listen -> Save -> now we will get primary and secondary key.

3. Program.cs ->

using Azure.Messaging.ServiceBus;

using ServiceBusQueue;

List<Order> orders = new List<Order)()

{

new Order() {orderId="1",Quantity=100,unitPrice=9.9f},

new Order() {orderId="2",Quantity=200,unitPrice=9.9f},

new Order() {orderId="3",Quantity=300,unitPrice=9.9f}

}

async Task SendMessage(List<Order> orders)

{

ServiceBusClient sbc = new ServiceBusClient(connectionString);

ServiceBusSender sbs = serviceBusClient.CreateSender(queueName);

ServiceBusMessageBatch sbmb = await serviceBusSender.CreateMessageBatchAsync()

foreach(Order order in orders)

{

if(!sbmb.TryAddMssage(

new ServiceBusMessage(JsonConvert.SerializeObject(order))))

{

throw new Exception("Error occured");

}

}

Console.WriteLine("Sending messages");

await sbs.SendMessagesAsync(sbmb);

}

**Peek Message**

async Task PeekMessages()

{

ServiceBusClient serviceBusClient = new ServiceBusClient(connectionString);

ServiceBusReceiver serviceBusReceiver = serviceBusClient.CreateReceiver(queueName,

new ServiceBusReceiverOptions() { ReceiveMode = ServiceBusReceiveMode.PeekLock });

IAsyncEnumerable<ServiceBusReceivedMessage> messages = serviceBusReceiver.ReceiveMessagesAsync();

await foreach(ServiceBusReceivedMessage message in messages)

{

Order order = JsonConvert.DeserializeObject<Order>(message.Body.ToString());

Console.WriteLine("Order Id {0}",order.OrderId);

Console.WriteLine("Quantity {0}",order.Quantity);

Console.WriteLine("Unit price {0}",order.UnitPrice);

})

}

async Task SendMessage(List<Order> orders)

{

ServiceBusClient sbc = new ServiceBusClient(connectionString);

ServiceBusSender sbs = serviceBusClient.CreateSender(queueName);

ServiceBusMessageBatch sbmb = await serviceBusSender.CreateMessageBatchAsync()

foreach(Order order in orders)

{

ServiceBusMessage serviceBusMessage = new ServiceBusMessage(JsonConvert.SerializeObject(order));

serviceBusMessage.ContentType = "application/json";

serviceBusMessage.ApplicationProperties.Add("Importance","High")

if(!sbmb.TryAddMssage(

serviceBusMessage))

{

throw new Exception("Error occured");

}

}

Console.WriteLine("Sending messages");

await sbs.SendMessagesAsync(sbmb);

}

application level properties can be read by ApplicationProperties["PropertyName"] ;

**Message Processor**

1. Create message processor method

ServiceBusClient sbc = new ServiceBusClient(connectionString);

ServiceBusProcessor serviceBusProcessor = serviceBusClient.CreateProcessor(queueName, new ServiceBusProcessorOptions());

serviceBusProcessor.ProcessMessageAsync += ProcessMessage;

serviceBusProcessor.ProcessErrorAsync += ErrorHandler;

await serviceBusProcessor.StartProcessingAsync();

Console.WriteLine("Waitint for messages");

Console.ReadKey();

await serviceBusProcessor.StopProcessingAsync();

await serviceBusProcessor.DisposeAsync();

await serviceBusClient.DisposeAsync();

static async Task ProcessMessage(ProcessMessageEventArgs message)

{

Order order = JsonConvert.DeserializeObject<Order>(messageEvent.Message.Body.ToString());

Console.WriteLine("Order Id {0}",order.orderId);

Console.WriteLine("Quantity {0}",order.Quantity);

Console.WriteLine("UnitPrice {0}",order.UnitPrice);

}

static Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

**Message Lock Duration**

async Task PeekMessages()

{

//Peeklock will pick the message from the queue and lock it for the lock duration mentioned in the azure portal, after that lock duration another application can access that message. If that message is not processed by previous pick.

ServiceBusClient serviceBusClient = new ServiceBusClient(connectionString);

ServiceBusReceiver serviceBusReceiver = serviceBusClient.CreateReceiver(queueName,

new ServiceBusReceiverOptions() { ReceiveMode = ServiceBusReceiveMode.PeekLock });

ServiceBusReceivedMessage messages =await serviceBusReceiver.ReceiveMessageAsync();

Order order = JsonConvert.DeserializeObject<Order>(message.Body.ToString());

Console.WriteLine("Order Id {0}",order.OrderId);

Console.WriteLine("Quantity {0}",order.Quantity);

Console.WriteLine("Unit price {0}",order.UnitPrice);

await serviceBusReceiver.CompleteMessageAsync(message); // this will mark our message operation as complete and dlete the message from the queue.

}

**Time To Live**

How long our messages should be active. After this Time to Live period messages should be saved permemnantly to dead letter queue. Provided dead letter queue is enabled and its not manually deleted.

We can do peek-lock delivery.

We can read this messages from dead letter queue as well. for that below is the setting

string queueName="appqueue/$DeadLetterQueue"

on connection string in the end - Entitypath=appqueue/$DeadLetterQueue\

**Duplicate Message Detection**

You can't enable or disable duplicate detection after the queue or topic is created.

1. Enable duplicate detection during queue creation -> enable time duration -> We need to keep some unique id, with the same id, it wont allow messages for 10 (whatever time duration we have mentioned ) mins duration time.

2. Create new SAS and update in .net program and update the new queue name. (Please note we have RootManageSharedAccessKey where we can access all the queue )

**Azure Service Bus - Topics**

1. Its a publisher subcriber model, if application subscribe to a topic then it will receive message when a message is published in that topic.

2. Create a topic then under topic -> create 2 subscriptions - >

3. Go to service bus explorer -> send message -> now both subscripers will get this message.

4. When peek or receive, we need to tell on behalf of which subscription we are doing that.

**Sending Messages from .**Net

There is no changes in sending messages same as storage queue.

1. Create .net project and install Azure.Messaging.ServiceBus

2. Create Shared access key for send -> and upadte this key in connection string.

3. Define topicName

**Sending Messages from .**Net

There is no changes in sending messages same as storage queue.

1. Create .net project and install Azure.Messaging.ServiceBus

2. Create .net project and install Azure.Messaging.ServiceBus

2. Create Shared access key for send -> and upadte this key in connection string.

**Sending Messages from .**Net

1.Create .net project and install Azure.Messaging.ServiceBus

2. Install ServiceBusTopicReceiver from nuget.

3. Create access policy to listen to messages and copy access key to this program connection string, topic name and subscriptionName in the ServiceBusReceiver code

**Topic Filters**

1. Subscripers can decide which messages they want to receive, with the help of fileters.

2. This can be defined with the use of rules

3. Each rule has a filter condition.

4. The message will contain a property called RuleName that shows the matching rule.

Types of files

\* Boolean filters - this is either the TrueFilter or FalseFilter. This can cause all the messages to be send to the subscription or for the subscription to receive no messages.

\* SQL filters - Here SQL like language can be used to evaluate the messages user-defined or system properties

\* Correlation filters - Here also conditions can be matched against the messages user or system defined properties.

\* Correlation filters are more efficient than SQL filters.

**Boolean Filters**

1. Go to subscription -> Overview -> Filters -> Add filter -> put condition like 1=0 -> This is always false and no messages will be recieved to the subscription under that particular topic.

**SQL Filters**

There are two types of properties.

1. User defined properties, this will go like user.Importance="High", so this topic will pick message only if the user defined Importance property is equal to High

2. System properties, this will work like sys.MessageId='1' only then pick that message.

3. We can use like opertion etc in sql filter.

**Correlation filters**

Correlation filters are same as sql filter, but we will have list of properties on a dropdown we can choose system and Custom properties and give values for our condition. It is efficient than sql filter.

**Azure Function to pick service bus**

1. Create a .net project -> Azure function -> Service Bus QueueTrigger -> Connection string setting name =ConnectionString and Queue Name =orderqueue

2. Settings.Json copy the Shared access key without Entitypath.

**Azure Event Hubs**

Azure Event hubs are basically take data from multiple sources and ingest the data in real time and then analyze or store the data for further analysis

This is a big data streaming platform.

This service can receive and process millions of events per second

You can stream log data, telemetry data, any sort of events to Azure Event Hubs.

**Different components**

1. Event producers - This is an entity that sends data to an event hub. The events can be published using the protocols - HTTPS, AMQP, Apache Kafka

2. Partitions - The data is split across partitions. This allows for better throughput of your data onto Azure Event Hubs.

3. Consumer groups - This is a view (state, position or offset) of an entire event hub

4. Throughput - This controls the throughput capacity of Event Hubs.

5. Event Receivers - This is an entity that reads event data.

**Lab - Event Hubs**

All resources -> Create resource -> Event Hubs -> Create Namespace

Then you can create Event Hub -> Partition count -> With more event hubs more events can be processed paralelly hence better throughput.

In Azure Service bus we had service bus explorer to send messages. but we dont have anything to send data to event hub, we need to create .net program to send data.

Sending data from .net program

--------------------------------

Create new console app -> Install Azure.Messing.EventHubs ->

program.cs ->

string connectionString =""; // connection string is taken from new Shared access key with send permission .

string eventHubName = "apphub"

sendData();

public class Device

{

public string deviceId {get;set;}

public float temperature {get;set;}

}

List<Device> devices = new List<Device>()

{

new Device() { deviceId ="D1", temperature=40.0f}.

new Device() { deviceId ="D2", temperature=39.0f},

new Device() { deviceId ="D3", temperature=42.0f},

new Device() { deviceId ="D4", temperature=32.0f},

new Device() { deviceId ="D5", temperature=23.0f},

}

void SendData()

{

EventHubProducerClient eventHubProducerClient = new EventHubProducerClient(connectionString, eventHubName);

EventDataBatch eventDataBatch = await eventHubProducerClient.CreateBatchAsync();

foreach(Device device in deviceList)

{

EventData eventData = new EventData(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(device));

if(!eventDataBatch.TryAdd(eventData))

Console.WriteLine("Error has occured adding data to the batch");

}

await eventHubProducerClient.SendAsync(eventDataBatch);

Console.WriteLine("Events sent");

await eventHubProducerClient.DisposeAsync();

}

After running this program we can see the request in the azure portal event hub resource. But there is no message

**Reading data to Azure Event Hub**

1. Create new console app -> Install Azure.Messing.EventHubs ->

2. Create Shared access policies -> with Listen permission

3. EventHub -> Consumer groups -> Default consumer group name is "$Default"

4. Program.cs

string connectionString="ListenpermissionSharedAccessPolicy";

string consumerGroup="$Default";

aysnc Task GetPartitionIds()

{

EventHubConsumerClient eventHubConsumerClient = new EventHubConsumerClient (consumerGroup, connectionString);

string[] partitionIds = await eventHubConsumerClient.GetPartitionIdsAsync();

foreach(string partitionid in partitionIds)

{

console.WriteLine("Partition Id {0}", partitionId)

}

}

async Task ReadEvents()

{

EventHubConsumerClient eventHubConsumerClient = new EventHubConsumerClient (consumerGroup, connectionString);

var cancellationSource = new CancellationTokenSource();

cancellationSource.CancelAfter(TimeSpan.FromSeconds(300));

await foreach(PartitionEvent partitionEvent in eventHubConsumerClient.ReadEventsAsync(cancellationSource.Token))

{

Console.WriteLine($"Partition Id {paritionEvent.Partition.PartitionId}"); // How data is split across partition is Azure's decision

Console.WriteLine($"Data Offset {partitionEvent.Data.Offset}"); // Offset is size of the data

Console.WriteLine($"Sequence Number {partitionEvent.Data.SequenceNumber}");

Console.WriteLine($"Partition Key {partitionEvent.Data.EventBody}");

Console.WriteLine(Encoding.UTF8.GetString(paritionEvent.Data.EventBody));

}

}

**Important points**

1. The consumer application needs to keep on running to process events in real time from the Event Hub.

2. After consuming the events do the events get deleted? - No, because Azure Event Hubs serves a different purpose. May be another type of consumer needs to read the events for different purpose.

3. Does that mean Azure Event Hubs will keep the message indefinitely? - No There is message retention period, after the period it will get deleted.

4. After running the consumer program again, it is again reading all of the events again from the begiinning, so our program needs to keep track of events being read.

**Reading from particular partition**

async Task ReadEvents()

{

EventHubConsumerClient eventHubConsumerClient = new EventHubConsumerClient (consumerGroup, connectionString);

string partitionId = await (eventHubConsumerC lient.GetPartitionIdsAsync()).First();

var cancellationSource = new CancellationTokenSource();

cancellationSource.CancelAfter(TimeSpan.FromSeconds(300));

await foreach(PartitionEvent partitionEvent in eventHubConsumerClient.ReadEventsFromPartitionAsync(partitionId, EventPosition.Latest))

{

Console.WriteLine($"Partition Id {paritionEvent.Partition.PartitionId}"); // How data is split across partition is Azure's decision

Console.WriteLine($"Data Offset {partitionEvent.Data.Offset}"); // Offset is size of the data

Console.WriteLine($"Sequence Number {partitionEvent.Data.SequenceNumber}");

Console.WriteLine($"Partition Key {partitionEvent.Data.EventBody}");

Console.WriteLine(Encoding.UTF8.GetString(paritionEvent.Data.EventBody));

}

**Using Event Processor Class**

Lets say we have our event hub data is read from particular position. But in case of any failure it should know to resume from its failure point.

We have Event processor class to achieve the same. It is doing by storing data of different checkpoints in azure storage accounts.

1. Create console based app -> Azure.Messinging.EventHubs.Processor, Azure.Storage.Blob

2. Program.cs

string connectionString ="ConnectionStringofSharedAccessPolicywith lisen permission";

string consumerGroup ="$Default"

string containerName ="storageAccountContainerName"

string blobConnectionString ="";

BlobContainerClient blobContainerClient = new BlobContainerClient(blobConnectionString ,containerName );

EventProcessClient eventProcessClient = new EventProcessClient(blobContainerClient ,consumerGroup, connectionString );

eventProcessClient.ProcessEventAsync += ProcessEvents;

eventProcessClient.ProcessErrorAsync += ErrorHandler;

await eventProcessClient.StartProcessingAsync();

Console.ReadKey();

eventProcessClient.StopProcessingAsync();

async Task ProcessEvents(ProcessEventArgs processEvent)

{

Console.WriteLine(processEvent.Data.EventBody.ToString());

}

static Task ErrorHandler(ProcessErrorEventArgs errorEvent)

{

Console.WriteLine(errorEvent.Exception.Message);

return Task.CompletedTask;

}

**Event Hubs Other concpets**

1. Event Hub namespace -> Azure Event Hub (Produce will send the event) -> Partitions -> Consumer group ($Default will be default) -> Receiver.

2. Throughput capacity -> Ingress -> up to 1MB per second or 1000 events per second

-> Egress -> up to 2MB per second or 4096 events per second

3. When you receive more than defined throughput units then You might start getting ServerBusyExceptions when the ingress traffic goes beyond the limit.

4. You cannot change the number of partitions after the hub is created, except for the dedicated cluster and premium tier offering.

5. Recommended throughput of 1 MB/s per partition

6. you can mention which property in your data can be the partition key. Based on the paritition key Azure will hash the value and map the event to the relevant partition.

7. The recommendation is to have one receiver per partition, you can have up to 5 concurrent readers per partition per consumer group, but you have to be careful not to duplicate the process of reading the same messages.

**Partition key Lab**

1. We need to send data to only particular deviceId

2. Send data to sent batch of records

void SendData()

{

EventHubProducerClient eventHubProducerClient = new EventHubProducerClient(connectionString, eventHubName);

List<EventData> dataList = new List<EventData>();

foreach(Device device in deviceList)

{

EventData eventData = new EventData(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(device));

dataList.Add(eventData);

if(!eventDataBatch.TryAdd(eventData))

Console.WriteLine("Error has occured adding data to the batch");

}

await eventHubProducerClient.SendAsync(dataList, new SendEventOptions() {PartitionKey="D1 "});

Console.WriteLine("Events sent");

await eventHubProducerClient.DisposeAsync();

}

**Enabling the capture feature**

1. This is available for standard tier of event hub namespace

2. Namespace -> capture -> Azure Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to an Azure Blob storage or Azure Data Lake Store account of your choice, with the added flexibility of specifying a time or Event Hubs throughput units. Event Hubs Capture is the easiest way toload streaming data into Azure and enables you to focus on data processing rather than on data capture.

3. Capture On -> Either any Minute Time Window or Size of any MB

4. Files will be saved on .Avro format, which is best for big files sizes. We have read this file using specific code.

**Streaming diagnostics data from an Azure SQL database**

1. This section we are going to send events from different source, we have dianostic data for various resources, we will see Aqure SQL

2. Change the throughput units of Event hub namespace -> from 1 to 5. (as we have more sources to publish event)

3. Create new Event Hub to receive events -> Sqlhub -> partition count as 4 ->

4. Azure SQL -> Diagnostics setting -> Metrics -> Basic -> Destination details -> Stream to an event hub -> Event hub policy name is used from the namespace shared access key (RootManagedSharedAccessKey) -> save -> Wait for 40 mins max. Now Azure SQL will send its diagnostics setting to event hub. \

5. Modify the existing program to point connection string of sqlhub

6. sqlhub -> shared access policy -> create a key to listen permission.

**Comparision with Azure Service Bus and Event Hub**

1. Azure Service Bus is message broker. means different applications will connected thourgh message broker, one application publish a message and another application consume it.

Azure Event Hub is injection service, means from different services send data to Event hub and consumer can consume it and also possible to store it in the database.

**Event** **Grid**

1. Event grid allows you to easily build applications with event-based architectures.

Trigger Event grid when there was an event in storage account

1. Create functional app (Plan type ->App service plan in ) -> Select Azure Event Grid Trigger -> BlobEventGridTrigger

2. Create Event Subscription on Storage account -> Topic Type (Storeage account)

-> System Topic Name -> blobtopic

-> Blob created , Blob Deleted, Directory created, Directory Deleted, Blob Renamed, Directory Renamed, Blob Tier Changed, Blob Inventroy Completed, Async Operation Initiated, Lifecycle policy completed.

3. Select Endpoint -> Azure storage account and function app in same azure region (North Europe)

Notice we are using topic concept, for example only Blob created and Blob deleted topic is listened by the Azure function

4. We can use Azure Blob storage trigger -> We can use new or updated blob is detected. But if you want to use more feature go for the above

Different between Azure Blob trigger and Event grid

1. If you Azure Function is based on the Consumption Plan, there can be a latency in processing new blobs, then consider two options

\* Change to the App Service Plan and put the Always On enabled option

\* Use the Event Grid trigger

Use the Event Grid trigger in high-scale events like processing more than 100,000 blobs or 100 blob updates per second.

Another option for faster and reliable processing of blobs

1. Consider creating a queue message when creating the blob

2. Use the queue trigger and then process the blob.

**Azure Functions - Processing Blobs**

1. Azure function -> Blob trigger -> RenameFunction -> blob -> FunctionName "ProcessBlob" ->

blob.cs

public class Blob

{

[FunctionName("ProcessBlob")]

public void Run([BlobTrigger("data/{name}", Connection ="connectionString")]Stream myBlob, string name, Ilogger log)

{

StreamReader reader = new StreamReader(myBlob);

string blob = reader.ReadToEnd();

log.LogInformation(blob);

}

}

AppSettings.json -> give connectionstring of storage account ->

**Azure Functions - Copying blobs**

1

public class Blob

{

[FunctionName("ProcessBlob")]

public aync Task Run([BlobTrigger("data/{name}", Connection ="connectionString")]Stream myBlob, [Blob("newdata/{name}", FileAccess.ReadWrite)] BlobClient newBlob , Ilogger log)

{

newBlob.UploadAsync(myBlob);

}

}

2. Install Azure.Storeage.Blobs

3. Install Microsoft.Azure.WebJobs.Extensions.Storage latest version

4. We will sometime get an error like Azure.Storage.Blobs: Service request failed, this is because azure functions require storage account for functioning, but in local it is using storage emulator but if you are getting this error then you can copy paste azure storage account connection string to get rid of this error.

**Event Grid Schema**

1. Event Grid event schema

2. Cloud event schema

**Debug Azure Event Locally**

(How we can point to our localhost when some blob is created, for that we have to use Web Hook as endpoint type and then we have to run ngrok on our local machine(this tool is suggested in microsoft documentation) this tool will make our local url available on the internet

1. Download Exe and then go to cmd prompt then navigate to the URL where our exe file is

2. ngrok config add-authtoken 1bocrHaa1gfmXLmPCWLTWKzumk\_326HAVXvLEeYhFRnjmaAi --copied from ngrok documentation)

3. ngrok http 7070 // port that is used by our localhost application

1. Create new Azure function -> Event grid trigger -> https://1925-86-98-187-66.ngrok.io/runtime/webhooks/EventGrid?functionName=ProcessEvent

Session Status Online

Account your account mail

Version 3.0.3

Region United States

Latency 198.22 ms

Web Interface http://127.0.0.1:4040

Forwarding https://1925-86-98-187-66.ngrok.io -> http://localhost:7071

Connections ttl opn rt1 rt5 p50 p90

**Getting Events at the resource group level**

Resource Group -> Events -> Create Event Subscription -> Event Type -> We can select all events of Resource group -> Endpoint type -> Azure Function -> EndPoint (Select Azure Function) ->

Go To All resources -> topic -> resourceGroupTopic of type Event Grid System Topic

**Azure Queue storage as consumer for your Event**

1. Storage account -> Add Queue -> Event subscription

2. Event subscription -> End point -> Storage Queue -> Select End point -> Select queue

**Filters - Subject Begin with**

Subject Filters -> Subject Begins with / Subject Ends With (Case sensitive subject matching)

**Connecting to HTTP endpoint**

1. Create a Azure function based on HTTP trigger (here we need to manually do the Handshake)

public static class EventGrid

{

[FunctionName("ProcessEvents")]

public static async Task<IActionResult> Run(

[HttpTrigger(AuthorizationLevel.Function, "get", "post", Route=null)] HttpRequest req, ILogger log)

{

string request = new StreamReader(req.Body).ReadToEnd();

log.LogINformation($"Received request body {request}");

EventGridSubscriber eventGridSubscriber = new EventGridSubscriber();

EventGridEvent[] eventGridEvents = eventGridSubscriber .DeserializeEventGridEvents(reqiuest);

foreach(EventGridEvent eventGridEvent in eventGridEvents)

{

if(eventGridEvent.Data is SubscriptionValidationEventData)

{

}

SubscriptionValidationEventData validationEventData = (SubscriptionValidationEventData) eve

log.LogInformation($"Validation code {validationEventData.ValidationCode}");

log.LogInformation($"Validation URL {ValidationEventData.ValidationUrl}");

SubscriptionValidationResponse response = new SubscriptionValidationResponse();

{

ValidationResonse = validationEventData.ValidateCode

};

return new OkObjectResult(response);

}

else

log.LogInformation(eventGridEvent.Data.ToString());

}

return new OkObjectResult(string.Empty);

**Create Custom Topic**

1. Create Event Grid Topic -> Create Event Subscription -> Event type -> app.neworder -> Endpont ->

2. Create new Function app project -> Install Azure.Messageing.EventGrid -> Create order class -> Program.cs ->

Uri topicEndpoint = new Uri("TopicEndpontofCustomEventTopic");

AzureKeyCredential azureKeyCredential = new AzureKeyCredential("Key from same CustomTopic");

List<order> orders = new List<order>()

{

new order() { OrderId =1, UnitPrice=9, Quantity=100 }

}

EventGridPublisherClient eventGridPublisherClient = new EventGridPublisherClient(topicEndpoint, azureKeyCredential);

string subject="adding new order";

string eventType="app.neworder"

string dataVersion="1.0";

List<EventGridEvent> events = new List<EventGridEvent>();

foreach(Order order in orders)

{

EventGridEvent eventGridEvent = new EventGridEvent(subject, eventType, dataVersion, JsonConvert.SerializeObject(order));

events.Add(eventGridEvent);

}

await eventGridPublisherClient.SendEventsAsync(events);

Console.WriteLine("Events Sent");

**API management Service**

You can place all of your API behind API management service, so that you can

\* Better Security

\* Define policies

\* Built-in cache

1. Create API Management -> Pricing tier -> Developer (no SLA) -> Creation of this resource will take 45 mins.

2. Sample API Echo API is created by default , Create New API -> Display name , Name, -> Add Operation (Add, Update, Delete etc) -> Test you can call your API just same as postman.

3. Invoke URL via postman -> By default when we use API management authorization is implemented. Subscription key and value is there under HTTP request of API management.

**Allow Access only via API management**

Azure web app (of API project) -> Networking -> Add rule -> AllowAPI -> Add API management public API -> and add rule to block all other access by IP Address block 0.0.0.0/0

**Using Swagger definition**

Instead of creating all operation manually, we can make use of swagger definition to populate all operations.

1. Open Swagger index page and download .json file which you can find just under the projectname

2. Create from definition -> OpenAPI -> Open API specification -> Select Json file -> Create

3. Notice json file have only operation not base URL. so we have to go to ProdcutAPI -> Settings -> Web Service URL

**API Management policy - IP restriction**

this URL gives all the possible policies https://learn.microsoft.com/en-us/azure/api-management/api-management-access-restriction-policies

1. API management -> Design -> Design -> In bound -> Policies -> We can see inbound, outbound, backend and onerror etc.

<policies>

<inbound>

<base/>

<ip-filter action="forbid">

<address>13.66.201.169</address>

<address-range from="13.66.140.128" to="13.66.140.143" />

</ip-filter>

</policies>

</inbound>

**Expressions**

https://learn.microsoft.com/en-us/azure/api-management/api-management-policy-expressions

**Rewrite** **URL**

Transformation policies -> Rewrite URL

1. http://domain/api/Products/{id} - this is the format for our api. to get particular product, but if we give id as query string it will not understand. We can use rewrite rule for this scenarios.

2. <set-variable name="" value="@(context.Request.Url.Query.GetValueOrDefault("id"))" />

<rewrite-uri template="@{return "/api/products/" + context.Variables.GetValueOrDefault<string>("id");}" />

3.

Inbound -> Incoming request to API management service (rule we write here will format the incoming request so that it can be understood by the API management)

Backend -> Modify the request before it reaches the backend or API service

Outbound -> Modify the response before it send to the user

Onerror -> if timeout occurs / authorization error occurs

4. Policy can be applied on the whole API level or in the Operation (get/post/put) levels also.

5. <base /> represents whatever policies applied on the base level is applicable to this operation.

**API mangement - return policy**

1. <choose>

<when condition="@(context.Response.StatusCode ==200)">

<return-response>

<set-status code="200" reason="OK">

<set-header name="Response-reason" exists-action="override">

<value>"Returned product list"</value>

</set-header>

<set-body> @{

string text= context.Response.Body.As<string>(preserveContent:true);

return tex;

}

</set-body>

After this policy we can see the header Response-reason is added and also the returned response is not properly formated as json, but the string is as is.

**Cache**

1. Go to any API operation -> getcall for example -> Send -> then you have message and track in the httpresponse -> cache-lookup -> "using cache 'internal'"

1. Cache lookup results in a miss / hit.

2. In bound rule

<cache-lookup vary-by-developer="false" vary-by-developer-groups="false" downstream-caching-tyupe="none"

must-revalidate="true" caching-type="internal" >

in the outbound

<cache-store duration="60">

https://learn.microsoft.com/en-us/azure/api-management/api-management-caching-policies

https://learn.microsoft.com/en-us/azure/api-management/api-management-sample-cache-by-key

**API management policy - oauth setup**

1. Azure AD -> Register an application "productapi"-> go to Manifes of "productapi and update accessTokenAcceptedVersion as 2 -> save

2. Expose API on "productapi" -> Add scope -> save andcontinue -> Scope name -> we will get Application ID URI

3. API management instance -> OAuth 2.0 + OpenID connect -> Add -> Client registration page URL ="http://localhost" -> Authorization code, and Client credentials (if url is going to be accessed from postman tool)

4. Authorization endpoint URL -> productapi -> overview -> oauth 2.0 authorization endpoint will come here. Token endpoint url also same location

5. Default scope -> application id uri of Expose API -> Application ID URI

6. Client secret -> Create a new secret from productapi

7. Client ID - > product API overview

8. Redirect URI -> Authorization code grant flow -> copy url and then go to prodcut api -> Authentication -> redirect url past here.

9. Select Access token and ID tokens.

10. API -> Product API -> Settings -> Uncheck "subscription required" and select Security -> OAuth 2.0 and OAuth 2.0 server as Product API

in these steps we are trying to register productapi is integrated with oauth of API management.

Create application object in app service

**API Management - Virtual network**

If our app service is hosted on virtual machine then it is not straightforward like our app service public ip address, where we are directly accessing.

As virtual machines are isolated virtual network we need to use load balancer to router our API management request to virtual machine. There are two approaches to do that.

1. External - the gateway is accessible from the public internet via an expernal load balancer.

2. Internal - the gateway is only accessible from the virtual network via an internal load balancer.