**Trainer: Prashant Ranjan**

**Email:** [**prashant.ranjan@koenig-solutions.com**](mailto:prashant.ranjan@koenig-solutions.com)

**Az 204 Training**

1. Access LOD
2. Use Microsoft Account ( Personal Account)
   1. Launch Lab : Az204 Lab

Hierarchy of Azure App Services

1. Create an Azure App Service Plan
2. Create web app within App Service Plan.
3. Configure App Settings
4. Deploy your code.

Demo: Create a web app by using the Azure portal.

Steps:

1. Visit <https://portal.azure.com>
2. Create a resource > Web App > Create
3. Provide details
   1. Subscription
   2. Resource Group : Create New
   3. Web App
      1. Instance : unique
      2. Publish: Code
      3. Run Time Stack : .net core 3.1
      4. Plan: Windows
      5. Region: East Us
   4. App Service Plan
      1. Create new
      2. Size: F1
   5. Next -> Deployments ( No Changes)
   6. Next -> Monitoring
      1. Enable Application Insight : Off
   7. No Changes Tag
   8. Review + Create > Create

**Command Execution:**

1. In Portal - > Cloud Shell icon ( Next to Search Bar)

**Or**

[**https://shell.azure.com**](https://shell.azure.com)

**Options: PowerShell /Bash**

1. **On Machine :**  Need to install AZURE CLI

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows>

**Demo: Upload a static HTML file inside Web App Create**

1. Open cloud shell
   1. Select Cloud shell icon next to search icon.
   2. Run below commands
      1. Create a folder
      2. Change to folder and clone git project
      3. Upload inside web app.

**mkdir demoHTML**

**cd demoHTML**

**git clone** [**https://github.com/Azure-Samples/html-docs-hello-world.git**](https://github.com/Azure-Samples/html-docs-hello-world.git)

**cd html-docs-hello-world**

**az webapp up –location eastus –name “WebappName” --html**

1. Refresh the url for web app .
   1. Select resource gorup inside azure portal
   2. Select web app
   3. Get the url and open the same.

**Important Links:**

1. **To schedule Exam + Online preparation:** <https://docs.microsoft.com/en-us/learn/certifications/exams/az-204>
2. **Lab Recordings:** <https://github.com/MicrosoftLearning/Lab-Demo-Recordings>
3. **GitHub Labs:** <https://github.com/MicrosoftLearning/AZ-204-DevelopingSolutionsforMicrosoftAzure>

App Settings:

1. Update the App Settings of app Plan
   1. Select the web app > change app service plan
      1. Choose for the tier.
2. Update the information as similar to Web.config or appsettings.json
   1. Select the Web app > Configuration
      1. Application Settings
      2. Connection Settings
      3. Path mapping
      4. Runtime
      5. Default document.

Ex: If an application built to access storage account. So storage account connection string will be added in Application Settings.

Demo : Apply Auto scale settings with App Service Plan

1. Visit <https://portal.azure.com>
2. Search for “**Monitor”** in search bar
3. Azure Monitor > Auto scale> Select App Service Plan
   1. Click it , to modify rules related to auto scale.
      1. If it not works for you follow below to upgrade app service plan.
         1. Select resource group
         2. Select app service plan
            1. Scale up > choose S1 plan.
            2. Apply to save.
   2. Click on “Custom auto scale”.
      1. Rename the profile if required.
      2. Provide min ,max , default instance.
         1. Add a rule
            1. Rule type : CPU Utilization
            2. Condition : Greater than
            3. Increase count by : 2
         2. Add a Rule
            1. Rule type
            2. Condition: less than
            3. Decrease count : 2
   3. Save Operation.

**Subscription Issue:**

1. Visit <https://portal.azure.com>
2. Search for **Subscriptions.**
3. Select Subscription > Resource Provider
   1. Select “microsoft.insight” and register.
   2. Wait for min 1 minute to get the status update.
4. Done.

**Deleting Resource Group**

1. <https://portal.azure.com>
2. Search for Resource Group
   1. Click to resource group
   2. Select operation of Delete
      1. Provide same name again
      2. Confirm deletion

Demo: Create an HTTP trigger function using Azure Portal.

1. <https://portal.azure.com>
2. Search for Function App : > Create
   1. Fill Details
      1. Subscription
      2. Resource Group
      3. Function App Name: unique
      4. Publish: Code
      5. Runtime Stack: .net Core
      6. Region: East US
   2. Next -> Hosting
      1. Storage Account: No Change
      2. Operating System: No change
      3. Plan: Consumption Plan
   3. Review + Create > Create.
3. Go to resource > Functions
4. Add a new function
   1. Development : In Portal
   2. Trigger : Http
   3. Add
5. Click on Code + Test ( Verify the code and test the code generated by template )
6. Get the function URL , visit the URL in different Tab.
   1. Make sure to open LOG to see details.
7. Pass parameter at the end of URL as below.

**&name=Azure Class**

1. **Check Log for the same.**
2. **Done.**

**Demo: Create Storage Account for block blob.**

1. Visit <https://portal.azure.com>
2. Search for Storage Account
   1. Add a new account & fill details
      1. Subscription: Default
      2. Resource Group: ManagedPlatform
      3. Storage Account Name: unique
      4. Performance : Premium
      5. Account Kind: BlockBlob
      6. Replication: LRS
   2. Review + Create > Create.

Demo: Add Lifecycle Management to Azure Storage Account

1. Identify Default Tier ( Information )
   1. Create storage Account
      1. Advanced Section Tab
         1. Default tier
            1. Hot
            2. Cool
2. Select Account
   1. Search for **Lifecycle Managment**
      1. Add Rule
         1. Define Rule name > Next
         2. If base blobs
            1. Last modified : 180
            2. Then : choose operation
         3. Add many condition if required.

**Day 3:**

**Azure Cosmo DB**

**Consistency Model :** [Consistency levels in Azure Cosmos DB | Microsoft Docs](https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels)

**Demo : Create a COSMO DB Account using Azure portal.**

1. Visit <https://portal.azure.com>
2. Search for Azure Cosmo DB
3. Add an account COSMO DB
   1. Subscription
   2. Resource Group: PolyglotData
   3. Name: polycosmos
   4. Apply free Tier
   5. Account Type: Non Production
   6. API: Core SQL
4. Next Distribution:
   1. Multi Region Write: Make end user to connect with nearest replica ( Enable / Disable)

Module 5:

IASS Solutions

Naming Convention of VM : devwebUE01

Azure Price Calculator : <https://azure.microsoft.com/en-in/pricing/calculator/>

Demo : Create / Provision a VM in Azure Portal

1. <https://portal.azure.com>
2. Search for Virtual Machine
3. Fill details
   1. Subscription
   2. Resource Group
   3. VM Name
   4. Location
   5. Redundancy
   6. Image: Windows 10 pro
   7. Size : D2sV3
   8. User name: azureuser
   9. Password : Pa$$w0rd1234
   10. Allow Inbound ports : RDP
4. Review + Create > Create
5. Go To Resources
   1. Connect
      1. RDP
         1. Download file
         2. Connet, it using RDP.

**Demo: Create ARM Template using AZ-PORTAL.**

**Scenario: Create a storage account using ARM template**

1. <https://porta.azure.com>
2. Search for Storage Account
   1. Subscription
   2. Resource Group
   3. Instance name
   4. Kind
   5. Type
   6. Review + Create
   7. Click “Download template for automation “.
   8. Download the same.
   9. Select **Parameters** tab and make a note of all parameters.

**Deployment Part**

1. Extract the downloaded folder ( template.json)
2. Visit Azure Dashboard > Create a resource
3. Search for Template Deployment
   1. Create > Build your own custom template.
   2. Load Template.json ( downloaded)

**Modification**

1. Delete the below section from **Parameter.**

"storageAccountName": {

"type": "string"

},

1. Update the variable parameter as below

"variables": {

"storageAccountName":"[concat(uniqueString(subscription().subscriptionId),'storage')]"

},

1. Update the resources information as below.

"resources": [

{

"name": "[variables('storageAccountName')]",

1. Save and provide parameters which copied in previous step.
2. Review + Create > Create to deploy the same.

**Demo: Retrieve and Deploy existing Docker Image Locally**

* **Install Docker Desktop**

1. To find the existing images inside Docker.

**docker image list**

1. Pull Image

**docker pull mcr.microsoft.com/dotnet/core/samples:aspnetapp**

1. Verify the image.
2. To run the docker image

**docker run -d -p 8080:80 mcr.microsoft.com/dotnet/core/samples:aspnetapp**

1. Open browser with **localhost:8080**

**Other Commands**

1. **To verify the processes which is running in docker**

docker ps

Or

docker ps –a

1. **To stop container.**

docker container stop < Image Name/ Container ID >

1. **Remove Container from Docker**

docker container rm <Image name / Container ID >

1. **Remove Image**

docker image rm < Image ID / Image Name>

Demo: Create a container image by docker.

* Docker Desktop
* Local Installation Git
  + <https://desktop.github.com>

1. Create a directory and clone the GitHub project inside it

Or

Create a directory get the project from github hub and paste inside the directory.

cd desktop

mkdir hotel-project

cd hotel-project

git clone <https://github.com/MicrosoftDocs/mslearn-hotel-reservation-system.git>

1. Create Dockerfile
   1. cd mslearn-hotel-reservation-system
   2. cd src
   3. echo "">Dockerfile
   4. notepad Dockerfile

Update the Dockerfile with below instruction.

#1

FROM mcr.microsoft.com/dotnet/core/sdk:2.2

WORKDIR /src

COPY ["HotelReservationSystem/HotelReservationSystem.csproj","HotelReservationSystem/"]

COPY ["HotelReservationSystemTypes/HotelReservationSystemTypes.csproj","HotelReservationSystemTypes/"]

RUN dotnet restore "HotelReservationSystem/HotelReservationSystem.csproj"

#2

COPY . .

WORKDIR "/src/HotelReservationSystem"

RUN dotnet build "HotelReservationSystem.csproj" -c Release -o /app

#3

RUN dotnet publish "HotelReservationSystem.csproj" -c Release -o /app

#4

EXPOSE 80

WORKDIR /app

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

1. Build Image

docker build -t reservationsystem .

1. Run the same

docker run -p 8080:80 -d --name reservations reservationsystem

1. Test the same

[localhost:8080/api/reservations/10](http://localhost:8080/api/reservations/10)

Note: if ports are in use, remove previous container by above commands.

**Day 4:**

**Demo:**

1. **Register an app with Microsoft Identity Platform**
2. **Interactive Authentication using MSAL.NET**

**Exercise 1:**

**Steps:**

1. Visit <https://portal.azure.com>
2. Search for “**Azure Active Directory”.**
3. Look it for **App Registration**
   1. **Name: az204-app-reg**
   2. Supported Account Types: Default (Account in this organization)
   3. Redirect URI
      1. Public Client /Native (mobile & Desktop)
      2. <http://localhost>
4. Register

**Exercise 2: Interactive authentication using MSAL.NET**

1. Create a folder on Desktop named “a**z-active-auth**"
2. Open the folder in VS code & run below of commands
   1. Menu -> Terminal -> New Terminal
      1. dotnet new console
      2. dotnet add package Microsoft.Identity.Client
3. Update with **Program.cs**

//1: Import Libraries

using System;

using System.Threading.Tasks;

using Microsoft.Identity.Client;

namespace az\_active\_auth

{

class Program

{

//2: Update Main Method as Async Task

static async Task Main(string[] args)

{

//3: Client ID + Tenant ID

const string \_clientID = "aebe9aac-adb2-480e-aadb-8607763a6c18";

const string \_tenantID = "709035bd-b46f-497a-8cb3-7d54d118aa54";

//4: Build App

var app = PublicClientApplicationBuilder

.Create(\_clientID)

.WithAuthority(AzureCloudInstance.AzurePublic,\_tenantID)

.WithRedirectUri("http://localhost")

.Build();

//5: Provide Permission to read profile

string[] scopes = {"user.read"};

//6: Get the token using Interactive Authentication

AuthenticationResult result = await app.AcquireTokenInteractive(scopes).ExecuteAsync();

Console.WriteLine($"Obtained Token : \t {result.AccessToken}");

}

}

}

1. dotnet build
2. dotnet run

Shorten URL : <https://bit.ly/2QPXgJu>

**Demo: Retrieving Profile Information using the Microsoft Graph**

1. **Create a folder named “az-graph-auth-demo"**
2. Open it using VS Code and run below command.
   1. dotnet new console
   2. dotnet add package Microsoft.Identity.Client
   3. dotnet add package Microsoft.Graph
   4. dotnet add package Microsoft.Graph.Auth --version 1.0.0-preview.2
3. Update with **Program.cs**

//1: Update with library

using System;

using Microsoft.Identity.Client;

using System.Threading.Tasks;

using Microsoft.Graph;

using Microsoft.Graph.Auth;

namespace az\_graph\_auth\_demo

{

class Program

{

//2: Update Main Method as Async Task

static async Task Main(string[] args)

{

//3: Client ID + Tenant ID

const string \_clientID = "aebe9aac-adb2-480e-aadb-8607763a6c18";

const string \_tenantID = "709035bd-b46f-497a-8cb3-7d54d118aa54";

//4: Build App

var app = PublicClientApplicationBuilder

.Create(\_clientID)

.WithAuthority(AzureCloudInstance.AzurePublic,\_tenantID)

.WithRedirectUri("http://localhost")

.Build();

//5:Provide Permissions

string[] scopes = {"user.read"};

//6:Graph Operation

var provider = new InteractiveAuthenticationProvider(app,scopes);

var client = new GraphServiceClient(provider);

//Get the profile details

User me = await client.Me.Request().GetAsync();

Console.WriteLine($"Display Name: {me.DisplayName}");

}

}

}

1. dotnet build
2. dotnet run

Module 7:

Implement secure Cloud Solutions

**Demo: Set & Retrieve Secret from Azure Key vault**

Note: Open cloud Shell. Tap to next icon to Search Bar.

Type: Bash

1. **Creating Resource group**  
   az group create --name az204KeyVaultRG --location eastus
2. **Create Azure Key vault**   
   az keyvault create --name az204myKeyVault --resource-group az204KeyVaultRG --location eastus
3. **Add Secret**
   1. Key: DBPassword
   2. Value: AZ204DBPassword

az keyvault secret set --vault-name az204mykeyvault --name "DBPassword" --value "AZ204DBPassword"

1. **Retrieve secret**  
   az keyvault secret show --name "DBPassword" --vault-name az204mykeyvault

Module 8: Implement API Management

**Demo:**

1. **Create an APIM.**
2. **Implement API by using Azure Portal.**

**Create an APIM**

1. Visit <https://portal.azure.com>
2. Search for “**APIM”**  > Create New
   1. Subscription
   2. Resource Group
   3. APIM Name: [your name]apim //unique
   4. Organization Name: [your name]
   5. Email Address: [your email address]
   6. Plan: Consumption
   7. Review + Create > Create

**Import API inside APIM using Azure Portal**

1. Select APIM Instance.
2. Select API > Open API Specification
   1. Select **full details** to see each detail
      1. Open API Specification:

<https://conferenceapi.azurewebsites.net/?format=json>

* + 1. URL Scheme: https
    2. App URL suffix: conference
  1. Create.

**Test API**

1. Select **Test** tab.
2. Click on “Get Speakers”
   1. Click on send & wait for result.
3. Click on GetSpeaker
   1. Provide an ID : 101
   2. Send.

**Policy Definition in APIM:** [Azure API Management policies | Microsoft Docs](https://docs.microsoft.com/en-us/azure/api-management/api-management-policies)

**Module 9: Azure Logic Apps**

**Demo: When a new email received in one account will forwarded in another account using Logic Apps.**

**Steps**

1. Visit <https://portal.azure.com>
2. Search for Logic Apps
3. Add New > Consumption
   1. Subscription
   2. Resource Group
   3. Name: AutomatedEmailFlow
   4. Review + Create >Create

Workflow Creation

1. Select Blank Template
2. Look for Gmail Connector > Select “When a new email arrives” Trigger.
   1. Connection name: gmailworkflow
   2. Sign In
   3. Frequency: 1minute
3. Add a new Step
   1. Select Gmail Connector
   2. Choose Action: Send an email
      1. Provide Email Address
      2. Add new parameter
         1. Subject : Forwarded mail : Azure Logic apps
         2. Body:

Hi,  
 There is a new mail in your main account.  
 Details:

Sender name: “Dynamic Sender Name”

Information: “Dynamic body”

1. Save
2. Click on Run.

Demo: Publish and subscribe to event grid events

1. Verify the subscription for the feature **“Microsoft.EventGrid”**
   1. Select subscription > Resource Provider > Search for **Microsoft .EventGrid.**
      1. If not registered , register the same.
2. Create a Custom Event Grid Topic.
   1. Azure portal >Azure Event Grid Topics > new
      1. Name:
      2. Next > Advanced
         1. Event Schema: Event Grid Schema
      3. Review + Create > Create
3. **Create a Web app**
   1. Azure portal > Web app
   2. Name: event viewer [your name]
   3. Publish: Docker Container
   4. Operating System: Linux
   5. App Service Plan: Create New
      1. Event Plan
   6. Next >Docker
      1. Image Source : Docker Hub
      2. Access Type : Public
      3. Image & tag : microsoftlearning/azure-event-grid-viewer:latest
   7. Review + Create > Create.
   8. Go to Resource > Browse/ Click on URL
   9. Copy the url and don’t close the browser.

https://eventviewerprashant.azurewebsites.net/

1. Create a new subscription
   1. Select Resource group > Event Grid Topic
   2. Select the Event Grid topic > Event Subscription
      1. Name: basicsub
      2. Event Schema : Event Grid Schema
      3. Endpoint Type: Web hook
      4. Select Endpoint: <https://eventviewerprashant.azurewebsites.net/api/updates>

**Send an Event to Custom Topic**

1. **Retrieve URL & Key for custom topic.**
   1. Select Topic
      1. Topic endpoint: <https://az204-custom-topic-10.eastus-1.eventgrid.azure.net/api/events>
      2. Key: yqokX43siWPCPUknyMUgd94lLF3n++HI6N1ty6iNw6A=
2. Open cloud shell
   1. Create two variables for endpoint & key

**Replace with own endpoint & key**

endpoint=https://az204-custom-topic-10.eastus-1.eventgrid.azure.net/api/events

key=yqokX43siWPCPUknyMUgd94lLF3n++HI6N1ty6iNw6A=

event='[{"id":"'"$RANDOM"'","eventType":"recordInserted","subject":"myapp/vehicles/motorcycles","eventTime":"'`date +%Y-%m-%dT%H:%M:%S%z`'","data":{"make":"Contoso","model":"NorthWind"},"dataVersion":"1.0"}]'

Note : Reload browser for web app

curl -X POST -H "aeg-sas-key: $key" -d "$event" $endpoint

Demo: Create Event HUB

1. Visit Azure Portal
   1. Search for “Event Hub”
   2. Register Namespace
      1. Subscription
      2. Resource Group
      3. Namespace name //unique
      4. Location : east us
      5. Tier : standard
      6. Throughput unit : 1 ( Depends ups Event Hubs Scalability)
      7. Review + Create > Create
2. Register Event Hub
   1. Select Event Hub Namespace ( Registered in previous step )
      1. + Event Hub
         1. Name:
         2. Partition count
         3. Retention
         4. Create

// To access the Event HUB in C#

1. Need Information of Connection String & Key
   1. Select Event Hub
      1. SAS ( Shared Access Signature)
         1. Root Managed Access Key
            1. Connection String
            2. Key

Demo: Using .NET core to send and receive message from Service Bus Queue.

* Visual Studio code
* Part 1: Send Message by an application
* Part 2: Receive message by another application

**Part 1: Creation of Service Bus Namespace**

1. Visit <https://portal.azure.com>
2. Search for Service Bus > Create New
   1. Fill details
      1. Subscription:
      2. Resource Group: Existing One
      3. Namespace name: //unique
      4. Pricing: Standard
      5. Review + Create > Create.
3. **Get the connection string.**
   1. Go to resource
   2. SharedAccessKey> RootManagedAccessKey
      1. Get the primary connection string.

Endpoint=sb://az204svcbus.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=ylkkjF48G6ERc/RCNlkEXu1UXF55elMjLkHWUkOyJBo=

1. **Add the Queue.**
   1. Select Service Bus Namespace
   2. Check for **Queues**
      1. Create New
         1. Name: az204-queue

**Part 2: Create a console application to send message to Queue.**

1. **On Desktop**: Create a folder named “**az204-service-bus-Send**".
2. Open folder using Visual Code and run below commands
   1. Menu > Terminal > New terminal
      1. dotnet new console
      2. dotnet add package Microsoft.Azure.ServiceBus

**Update in Program.cs**

**//1: Importing libraries**

**using System;**

**using System.Text;**

**using System.Threading;**

**using System.Threading.Tasks;**

**using Microsoft.Azure.ServiceBus;**

**namespace az204\_svcbus\_send**

**{**

**class Program**

**{**

**//2: Creating global variables**

**/\***

**1. Service Bus Connection String**

**2. Queue name**

**3. Object of Queue Client**

**\*/**

**const string ServiceBusConnectionString="Endpoint=sb://az204svcbus.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=ylkkjF48G6ERc/RCNlkEXu1UXF55elMjLkHWUkOyJBo=";**

**const string QueueName="az204-queue";**

**static IQueueClient queueClient;**

**//3: Change Main Method as async Task**

**static async Task Main(string[] args)**

**{**

**//4:Intilization**

**const int numberOfMessages =10;**

**queueClient = new QueueClient(ServiceBusConnectionString,QueueName);**

**Console.WriteLine("===============================================");**

**Console.WriteLine("Press Enter to exist after sending all message");**

**Console.WriteLine("===============================================");**

**//5: Sending all message**

**await SendMessageAsync(numberOfMessages);**

**//6: Read for Enter key to get Exit.**

**Console.ReadKey();**

**await queueClient.CloseAsync();**

**}**

**//7: Create Custome Method SendMessageAsync**

**static async Task SendMessageAsync(int numberofMessageToSend){**

**//8: Create a loop up to number of Message to Send**

**try{**

**for(var i = 0; i<numberofMessageToSend; i++){**

**//9: Prepare Message**

**string messageBody = $"Message {i}";**

**var message = new Message(Encoding.UTF8.GetBytes(messageBody));**

**//Send Message**

**Console.WriteLine($"Sending Message ... {messageBody}");**

**await queueClient.SendAsync(message);**

**}**

**}catch(Exception e){**

**Console.WriteLine($"{DateTime.Now} :: Exception {e.Message}");**

**}**

**}**

**}**

**}**

1. dotnet build
2. dotnet run
3. To validate
   1. Visit Azure portal > Service Bus > Queue
   2. Check number of Active Message
      1. It must be 10.

**Demo: Receive Message from Queue**

1. Create a folder named “**az204svcbusRec”**
2. Open in VS Code and run below of commands
   1. Dotnet new console
   2. Dotnet add package Microsoft.Azure.ServiceBus
3. **Update with Program.cs**

**//1: Importing libraries**

**using System;**

**using System.Text;**

**using System.Threading;**

**using System.Threading.Tasks;**

**using Microsoft.Azure.ServiceBus;**

**namespace svcbusrec**

**{**

**class Program**

**{**

**//2: Global Variables**

**/\***

**1. Service Bus Connection String**

**2. Queue Name**

**3. Object of Queue Client**

**\*/**

**const string ServiceBusConnectionString="Endpoint=sb://az204svcbus.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=ylkkjF48G6ERc/RCNlkEXu1UXF55elMjLkHWUkOyJBo=";**

**const string QueueName="az204-queue";**

**static IQueueClient queueClient;**

**//3: Update Main Method as Async Task**

**static async Task Main(string[] args)**

**{**

**//4:Intilization**

**queueClient = new QueueClient(ServiceBusConnectionString,QueueName);**

**Console.WriteLine("===============================================");**

**Console.WriteLine("Press Enter key to Exit After Recieving messages");**

**Console.WriteLine("================================================");**

**//5:Registe the Queu message Handler and Recieve the message in Loop**

**RegisterOnMessageHandlerAndRecieveMessage();**

**//6:Close operation after reading all message**

**Console.ReadKey();**

**await queueClient.CloseAsync();**

**}**

**//7:Define RegisterOnMessageHandlerAndRecieveMessage with handler**

**static void RegisterOnMessageHandlerAndRecieveMessage(){**

**//8: Create Message Handler**

**var messageHandlerOptions = new MessageHandlerOptions(ExceptionRecievedHandler){**

**MaxConcurrentCalls = 1,**

**AutoComplete = false**

**};**

**//9: Processing the message**

**queueClient.RegisterMessageHandler(ProcessMessageAsync,messageHandlerOptions);**

**}**

**//10: Define ProcessMessageAsync to read message**

**static async Task ProcessMessageAsync(Message message,CancellationToken cancellationToken){**

**//11:Read the message**

**var messageSequnceNumber = message.SystemProperties.SequenceNumber;**

**var messageBody = Encoding.UTF8.GetString(message.Body);**

**Console.WriteLine($"Recieved Message: \n Sequence Number {messageSequnceNumber} \n Body: {messageBody}");**

**//12: End the operation after recieving all message**

**await queueClient.CompleteAsync(message.SystemProperties.LockToken);**

**}**

**//13: Define Custom Exception Handler**

**static Task ExceptionRecievedHandler(ExceptionReceivedEventArgs arg){**

**Console.WriteLine($"Message Handler Encounterd Exception");**

**var context = arg.ExceptionReceivedContext;**

**Console.WriteLine("Exception Context for Trouble Shooting");**

**Console.WriteLine($"- Endpoint {context.Endpoint}");**

**Console.WriteLine($"- Entity Path: {context.EntityPath}");**

**return Task.CompletedTask;**

**}**

**}**

**}**

1. dotnet build
2. dotnet run

Application Insights:

1. Visit portal
2. Search for Application Insight
   1. Create
   2. Get the instrumentation key
   3. Add to project
      1. Web Application : Script
      2. Console : Application Insight Nuget Package

Redis cahce:

1. Azure Portal > Azure cache for Redis
2. Provide details
   1. Name: //unique
   2. Plan : Standard/Basic/ Premium
   3. Create