**CLOUD HANDSON DAY – 82**

1. **Azure resource group creation**

They are multiple steps like creation , implementation , deployment .

1. Home page

**Graphical user interface, application

Description automatically generated**

1. New Resource Group

Graphical user interface, text, application, email

Description automatically generated

1. Name of the resource group DNAzure\_ResGrp

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Creation process

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Deployment

**Graphical user interface, application

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

1. **Navigation to Container registry**

They are multiple steps like creation , implementation , deployment .

1. Home

**Graphical user interface, application

Description automatically generated**

1. Searching

**Graphical user interface, text, application, email

Description automatically generated**

1. Creation

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

1. Implementation

**Graphical user interface, text, application, email

Description automatically generated**

1. Deployment

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

1. **Logic App**

They are multiple steps like creation , implementation , deployment .

1. Home

Graphical user interface, application

Description automatically generated

1. Creation

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. SSR

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Gmail

Graphical user interface, text, application

Description automatically generated

1. Output

Graphical user interface, text, application, email

Description automatically generated

1. **Service Bus Sending Queue**

They are multiple steps like creation , implementation , deployment and we will be using visual studio for console app of dotnet core for sending and receiving the queue of messages.

1. Home

Graphical user interface, application

Description automatically generated

1. Creation

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. Implementation

﻿using Azure.Messaging.ServiceBus;

using System;

using System.Threading.Tasks;

namespace ConsoleApp2

{

class Program

{

static string connectionString = "Endpoint=sb://myservicebus1715.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=2z69nPdMldz8ZiSc1Hbx2wvhI3BOb6Hgm4QaKSiRmbg=";

static string queueName = "Queue";

static ServiceBusClient client;

static ServiceBusSender sender;

private const int numOfMessages = 3;

static async Task Main()

{

client = new ServiceBusClient(connectionString);

sender = client.CreateSender(queueName);

// create a batch

using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

for (int i = 1; i <= 3; i++)

{

// try adding a message to the batch

string message = "Message : " + i + " generated at : " + DateTime.Now.ToString();

if (!messageBatch.TryAddMessage(new ServiceBusMessage(message)))

{

// if it is too large for the batch

throw new Exception($"The message {i} is too large to fit in the batch.");

}

}

try

{

// Use the producer client to send the batch of messages to the Service Bus queue

await sender.SendMessagesAsync(messageBatch);

Console.WriteLine($"A batch of {numOfMessages} messages has been published to the queue.");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await sender.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

Console.ReadKey();

}

}

}

1. Output

Text

Description automatically generated

1. **Service Bus Receiving Queue**

They are multiple steps like creation , implementation , deployment and we will be using visual studio for console app of dotnet core for sending and receiving the queue of messages.

1. Implementation

﻿using Azure.Messaging.ServiceBus;

using System;

using System.Threading.Tasks;

namespace ConsoleApp3

{

class Program

{

static string connectionString = "Endpoint=sb://myservicebus1715.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=2z69nPdMldz8ZiSc1Hbx2wvhI3BOb6Hgm4QaKSiRmbg=";

static string queueName = "Queue";

static ServiceBusClient client;

static ServiceBusProcessor processor;

// handle received messages

static async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body}");

// complete the message. messages is deleted from the queue.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

static Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

static async Task Main()

{

client = new ServiceBusClient(connectionString);

processor = client.CreateProcessor(queueName, new ServiceBusProcessorOptions());

try

{

processor.ProcessMessageAsync += MessageHandler;

processor.ProcessErrorAsync += ErrorHandler;

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

await processor.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

Console.ReadKey();

}

}

}

1. Output

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Conclusion is that we have created an service bus for sending and receiving message.