**TASK-6: IMPLEMENTING CONTINUOUS DEPLOYMENT PIPELINE**

**Goal**:

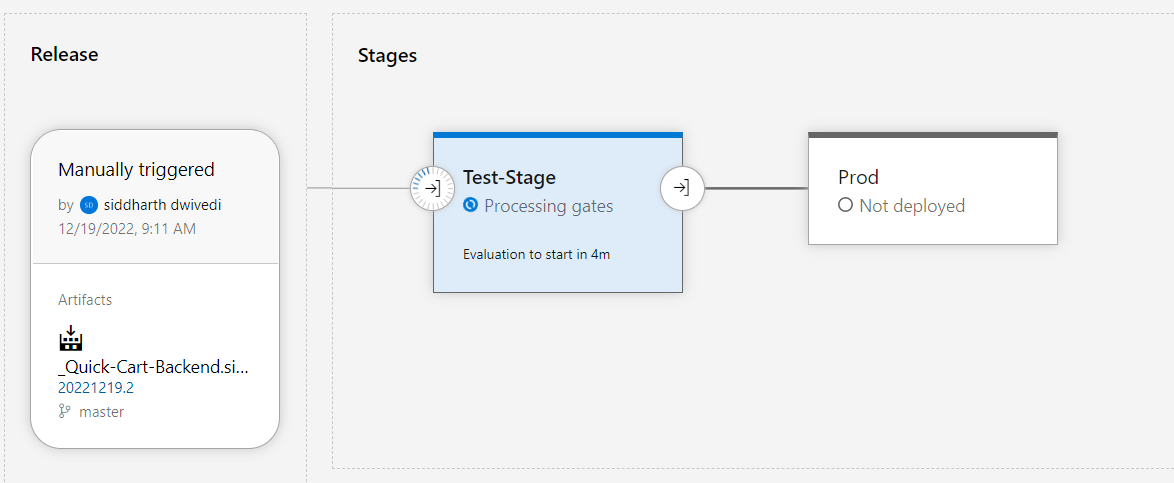
* Learn to build CD mechanism using Classic Interface.
* Understand Approval mechanism + Gates.

**1.** Make sure the Source code is present in the Azure Repos.

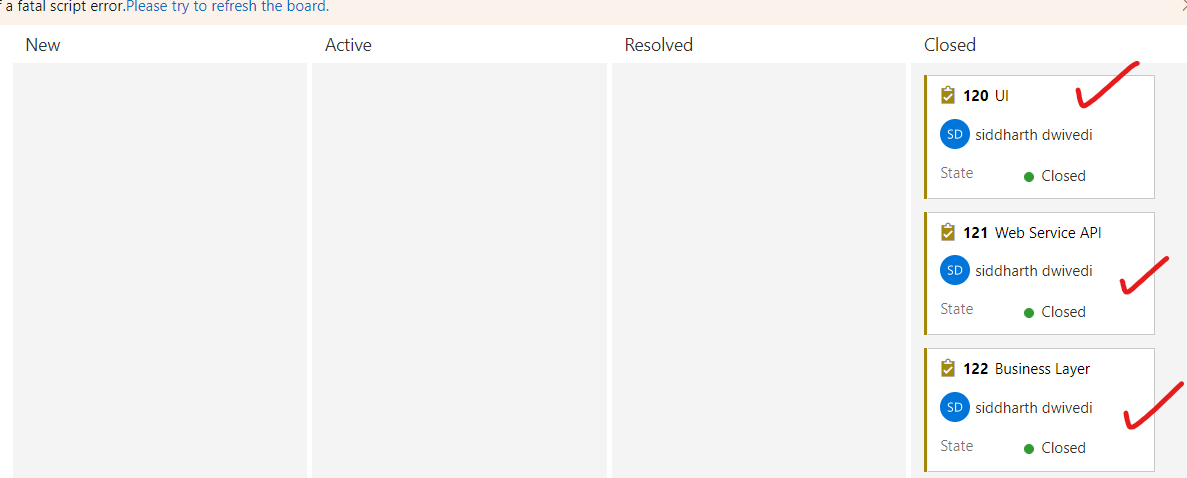
**2.** Make sure the CI Pipeline is building the solution and keeping the Binaries in Artifact.

**3**. Create a new Release Pipeline, which will deploy the changes to Azure App Service.

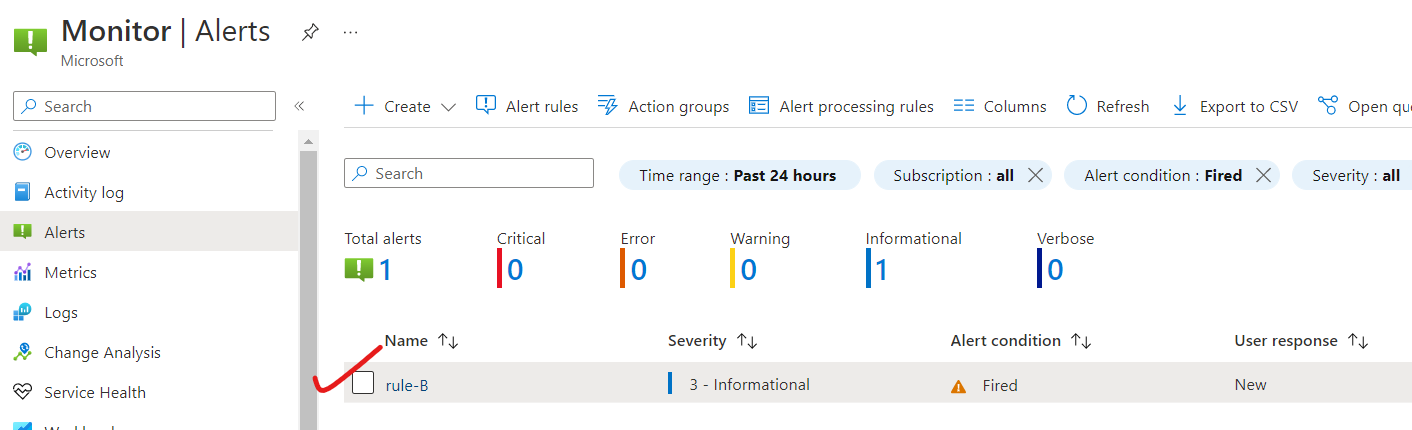
**4**. There should be 2 stages, Test and Prod.



**5**. Test stage should have a Pre-deployment Condition that All the Tasks should be in closed state from Azure Boards.



**6**. Test Stage should also have a Post-deployment Condition (GATE) that if there is any Azure Monitor Alerts raised on the Test-App Service (Average Working Memory Set > 1 Byte) then the further deployment should stop.



**7**. Prod Stage should have an Approval (Pre-deployment condition) in place which would always ask one person for the manual approval before the Prod deployment.

Keep the timeout to be 5 mins.

**8**. Now reference a different DB to this App Service for the Production Stage.

Data Source=quickcart-server.database.windows.net;Initial Catalog=test-dB;user id=demouser; password=Siddharth@1234

Make sure that the DB Value is coming from the Keyvault.

**TASK-7: IMPLEMENTING MESSAGES USING SERVICE BUS**

**Goal**:

* Create Service Bus Queue.
* Flow the messages from 1 Microservice to Another.

1. Clone the Frontend Project from <https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-FrontEnd>

2. Execute Orders Table

create table orders

(

orderid int identity,

custEmail varchar(50) references Customers(emailID),

prodID int,

prodCost int,

orderdate dateTime default getDate()

)

and Execute your own Procedure in the Database with this Format

create proc usp\_AddOrder\_<Your\_Name>

as

begin

begin try

insert into orders values ('Your\_email',1,2000,default)

return 1

end try

begin catch

return 0

end catch

end

Make sure in the customer’s table your emailID is present.

3. Create Payment Microservice Using Azure Function which would Process the Payment and then send the Payment Details to the Service Bus Queue.

Write the code yourself!

4. Create Order Microservice using Azure Function which would get triggered whenever there is a new message in the Queue. This service would insert a new row in the Order table.

**Reference Code:**

[FunctionName("order")]

public static async Task Run([ServiceBusTrigger("ffff", Connection = "con")]string myQueueItem, ILogger log)

{

dynamic data = JsonConvert.DeserializeObject(myQueueItem);

string prodCost = data.prodCost;

string prodID = data.prodId;

int result = 0;

//SQL connection

SqlConnection conObj = new SqlConnection("");

//command

SqlCommand cmdObj = new SqlCommand("usp\_AddOrder", conObj);

cmdObj.CommandType = CommandType.StoredProcedure;

try

{

SqlParameter prmReturnValue = new SqlParameter();

prmReturnValue.Direction = ParameterDirection.ReturnValue;

cmdObj.Parameters.Add(prmReturnValue);

conObj.Open();

cmdObj.ExecuteNonQuery();

int res = Convert.ToInt32(prmReturnValue.Value);

if (res == 1)

result = 1;//it means added

else

result = 0;//error

}

catch (Exception e)

{

result = -1;

}

finally

{

conObj.Close();

}

log.LogInformation(""+result);

}