**RUNBOOK AZURE DEVOPS TO AWS CLOUD (END – END PIPELINE)**

**CONTENTS**

1. **Introduction ……………………………………………………………...3**
2. **Overview………………………………………………………………….3**
3. **Prerequisites………………………………………………………………3**
   1. **Adding tools to Azure devops**
   2. **Create service connection**
4. **.NET Application………………………………………………………….6**
   1. **CI Pipeline**
      1. **Create Pipeline**
      2. **Build the Docker image**
      3. **Add Post to slack task**
      4. **Add AWS ECR Push task**
      5. **Add AWS CLI task**
      6. **Add Post to slack task**
      7. **Add Kiuwan task**
5. **Zapier for automate work items creation in Azure boards……13**
   1. **Creating A ZAP**
6. **Creating ECR and ECS in AWS………………………………………18**
   1. **ECR**
   2. **Deploy Container to ECS**
      1. **Create a Load Balancer**
      2. **Creating a cluster**
      3. **Create a new task definition**
      4. **Create Container**
      5. **Create Service**

**7. VSTS Bot Integration in Slack to trigger build from Slack……….22**

**1. Introduction:**

The document contains the process of building a CI/CD pipeline with Azure Devops to AWS Cloud that includes .NET application build with Docker Image Build kiuwan code analysis and code security, Slack Integration, VSTS Bot integration and Zapier integration. The Deployment platform is AWS Cloud.

**2. Overview:**

The overview of the pipeline is as:

Docker Build

Kiuwan Code Analysis

Build

Slack Notification

.NET Build and Test

AWS ECS for deployment

Slack and email notification

AWS ECR Push

Create Azure dashboard

Zapier for create work items in boards if build fails

**-**

**3. Prerequisites:**

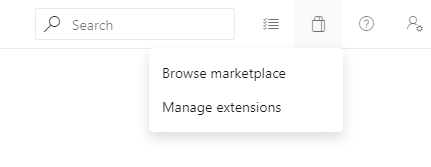
* Azure Devops agent
  + Ubuntu-latest
* Repos with .net application
* AWS account ,access key and access key value for service connection
* Slack channel to post notifications and slack api token
* Zapier account

**3.1 Adding tools to Azure devops**

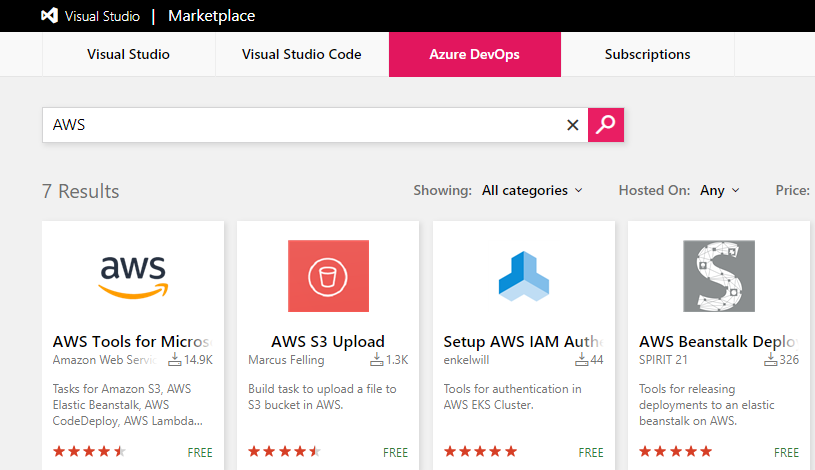
* Add AWS, Kiuwan and post to slack tools to Azure devops organization.
* Go to azure project home page and click on bag icon



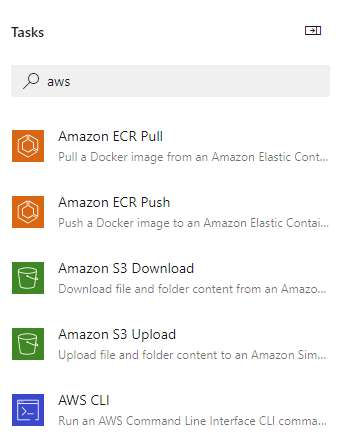
* Click on manage extension and

****

* Click on Browse marketplace to search required tools to integrate with Azure devops

****

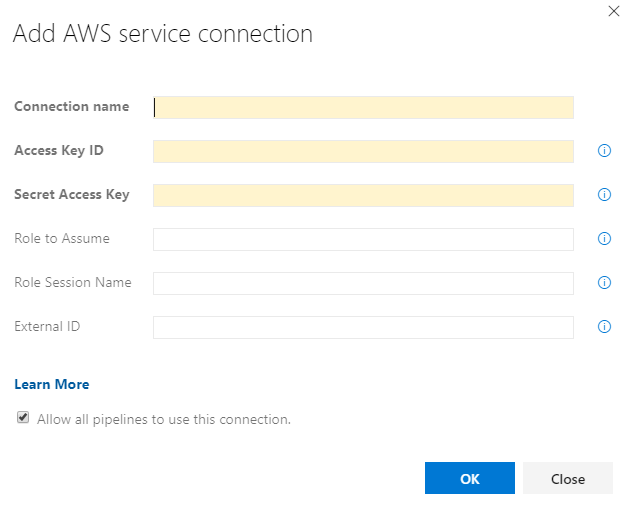
* Click on AWS and add that to your azure project. Once you add AWS to project you can see AWS tasks in Azure pipelines.



* Same process follow for post to slack and kiuwan tools.

**3.2 Create service connection**

* Click on project settings-> click on **service connections**->click on **new service connection->**select **AWS** from dropdown list then below wizard will appear
* Enter the connection name, Access Key ID and Secret access key and click on Ok.
* For AWS Access Key id and secret access key, we need to create IAM user in AWS

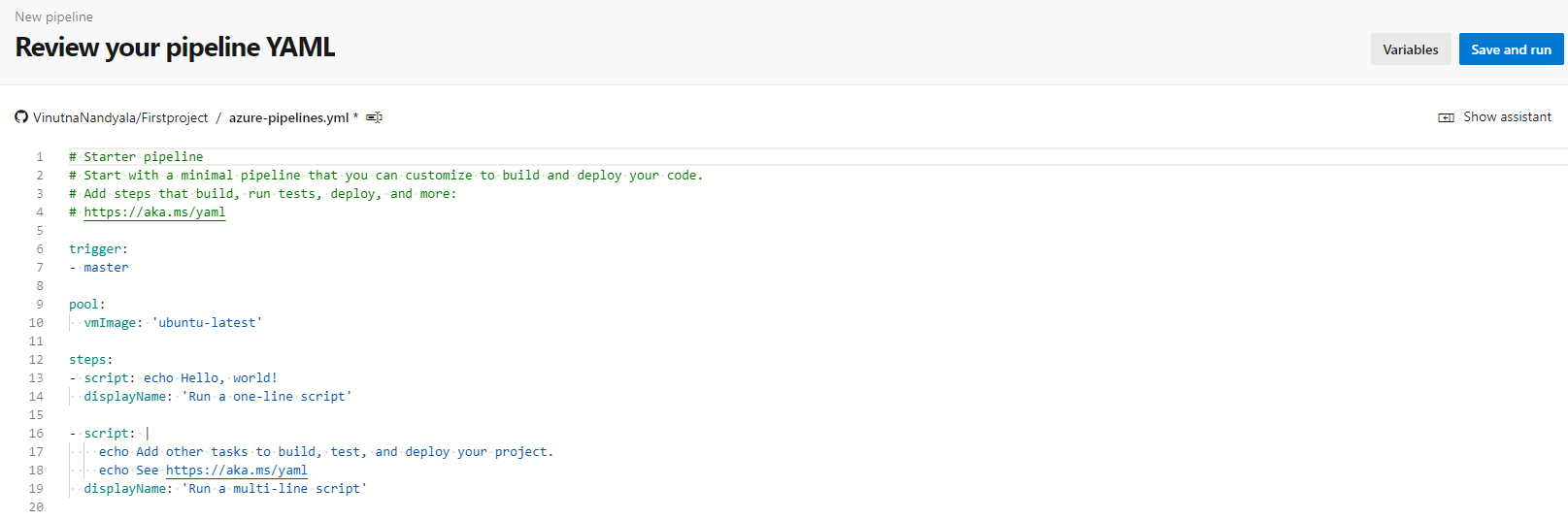
****

**4. NET APPLICATION**

**4.1 CI Pipeline:**

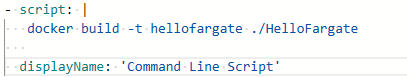
**4.1.1 Create Pipeline**

* Click on Pipeline>Build.
* Select New Build pipeline.
  + Choose Repo name with Repo name.
* Click on starter Pipeline



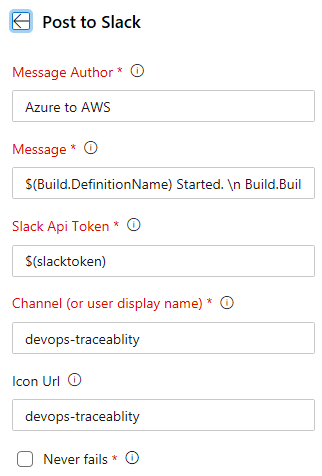
**4.1.2 Build the Docker image**

* In yml code under the script task enter the below commands to build the Docker image



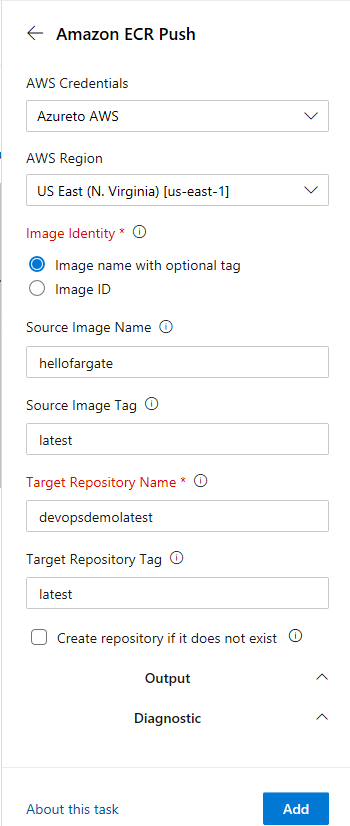
* Click on **show assistant**

**4.1.3 Add Post to slack task**

* Search post to slack task and add it. Enter the details 

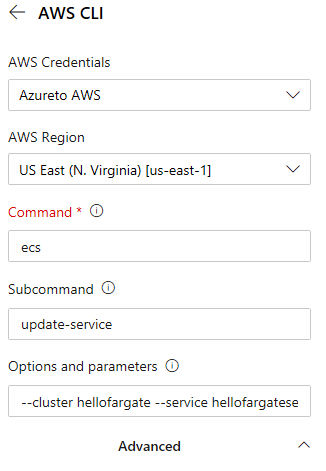
**4.1.4 Add AWS ECR Push task**

* Add AWS ECR push task and enter the details
* AWS credentials: Service connection name which we created earlier
* Source Image name: The Image name which we build
* Target Repository Name: Amazon Docker registry name(ECR)



**4.1.5 Add AWS CLI task**

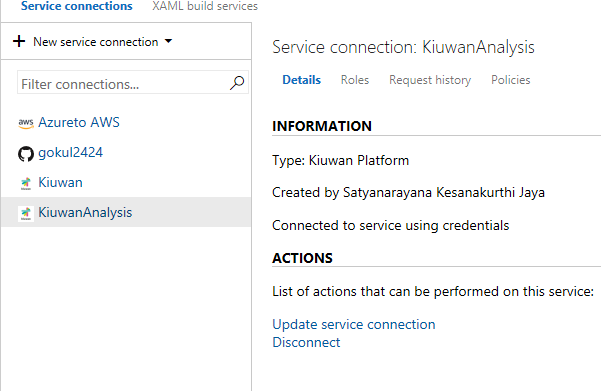
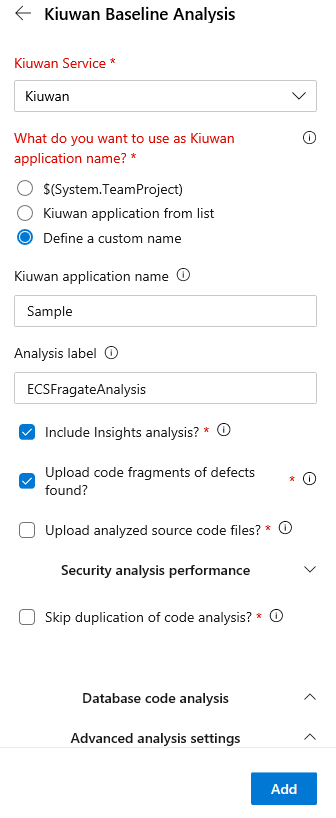
* Add AWS CLI task to deploy Docker image to AWS ECS and enter the details
* AWS credentials: Service connection name which we created earlier
* Command: ecs
* Subcommand: update-service
* Options and parameters: provide ecs cluster name and service name

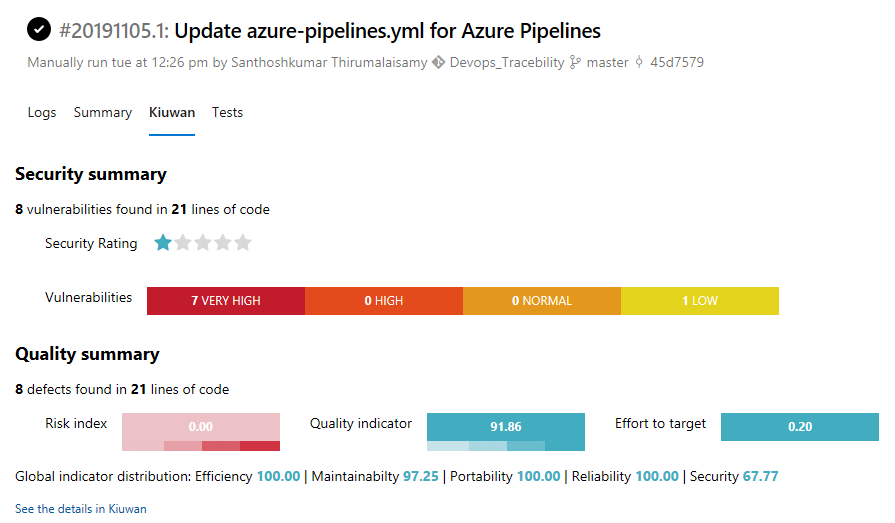


**4.1.5 Add Post to slack task**

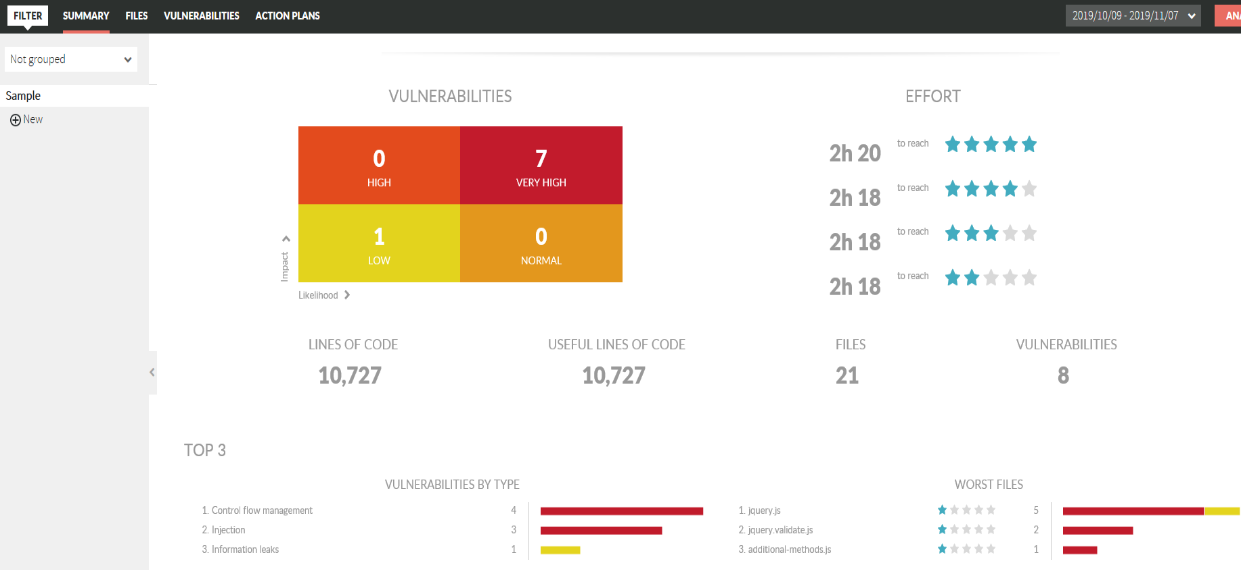
* Add post to slack task again to notify team in slack after build completed

**4.1.6 Add Kiuwan task**

* Add the kiuwan service connection
* Now add Kiuwan baseline analysis task to pipeline and configure like below
* Kiuwan service: select the service connection name
* Select define custom name radio button
* Give any name as application name
* Give analysis label name
* After running the pipeline we can see the kiuwan code coverage report like below screen shot



* For detailed report, click on See details in Kiuwan hyperlink available under the above report page.
* By clicking on the link you can navigate to kiuwan site and provide username and password then we can see the full report like below



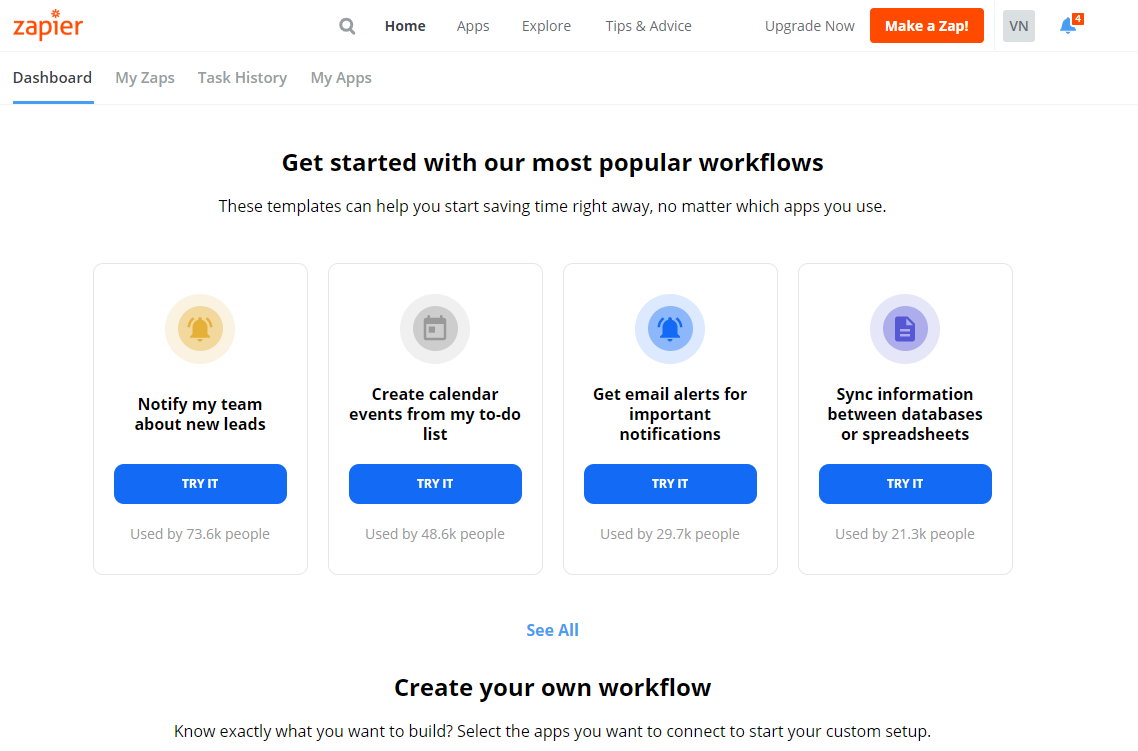
* By clicking on every tab on top of the page, we can see reports related files, vulnerabilities reports and action plans

**5. Zapier for automate work items creation in Azure boards**

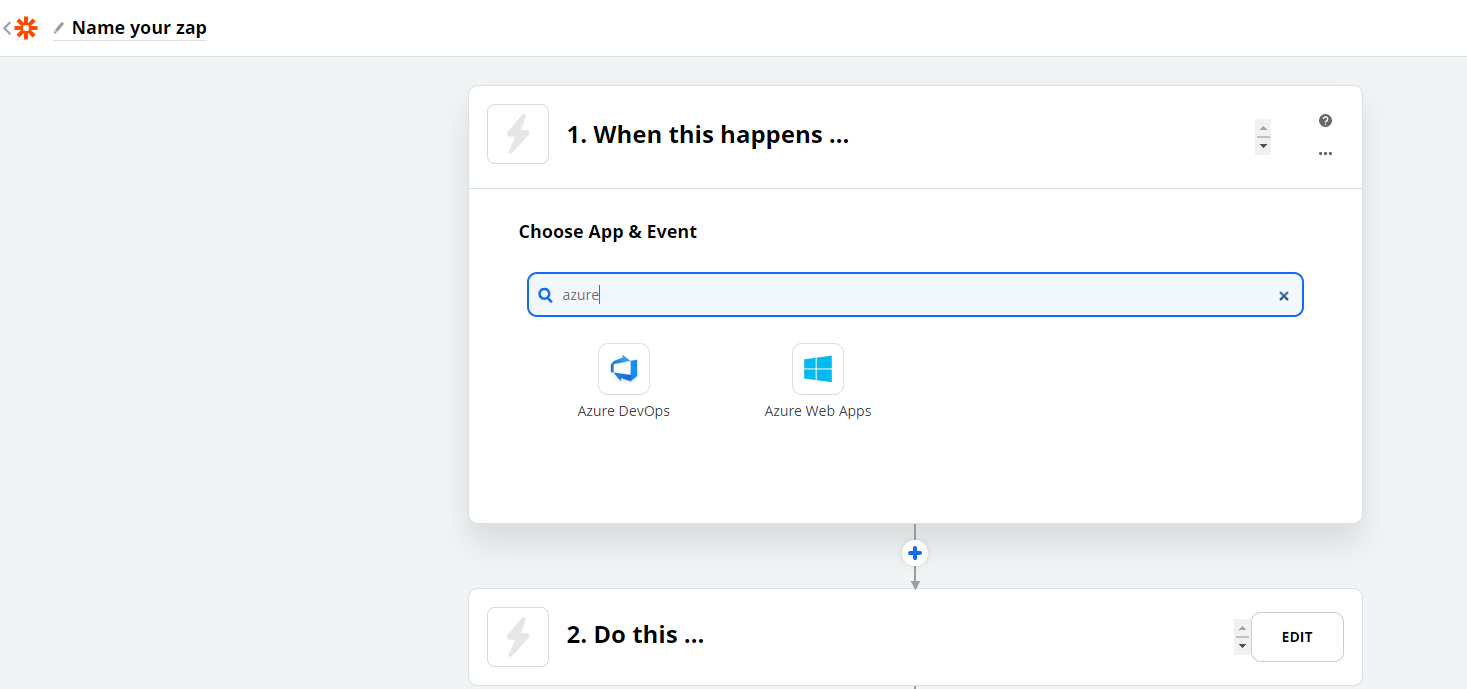
* Login to zapier site and create ZAP to automate work item creation in boards whenever build fails

**5.1 Creating A ZAP**

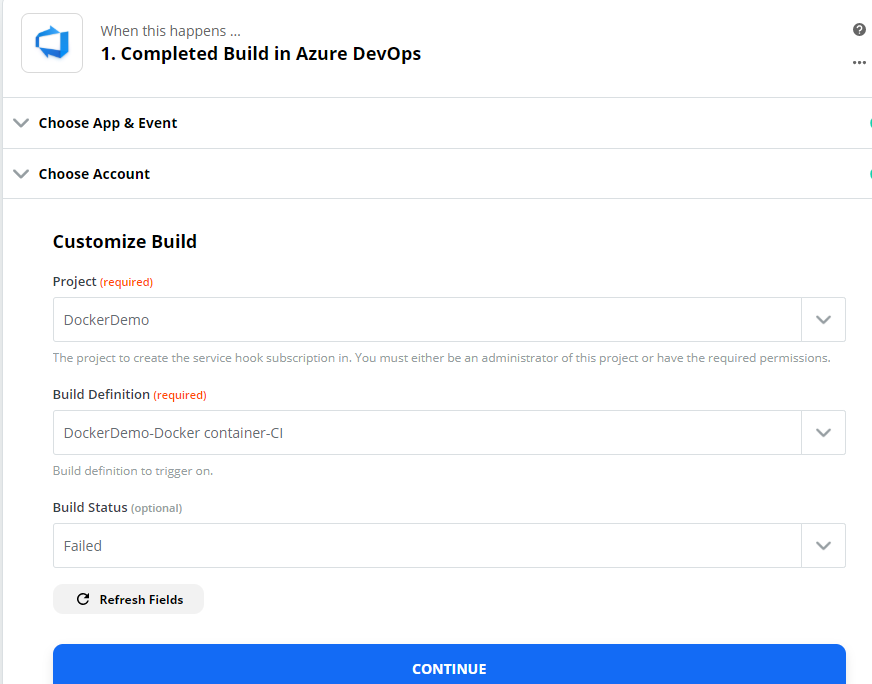
* Zapier Home page will look like as below screen shot

****

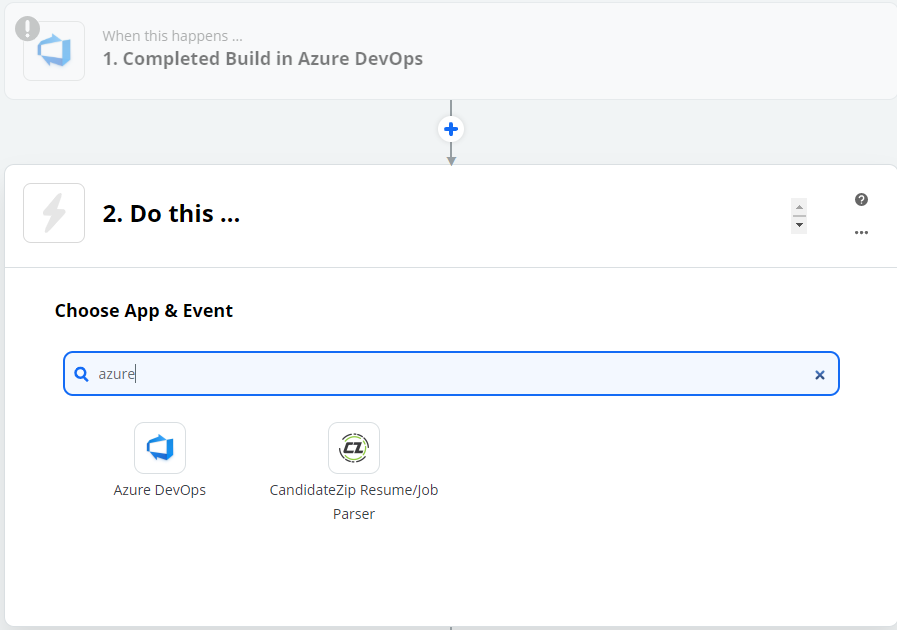
* To create Zap, click on **Make a Zap** tab at top right corner of the page.
* Give the name to Zap
* Search for Azure devops and select



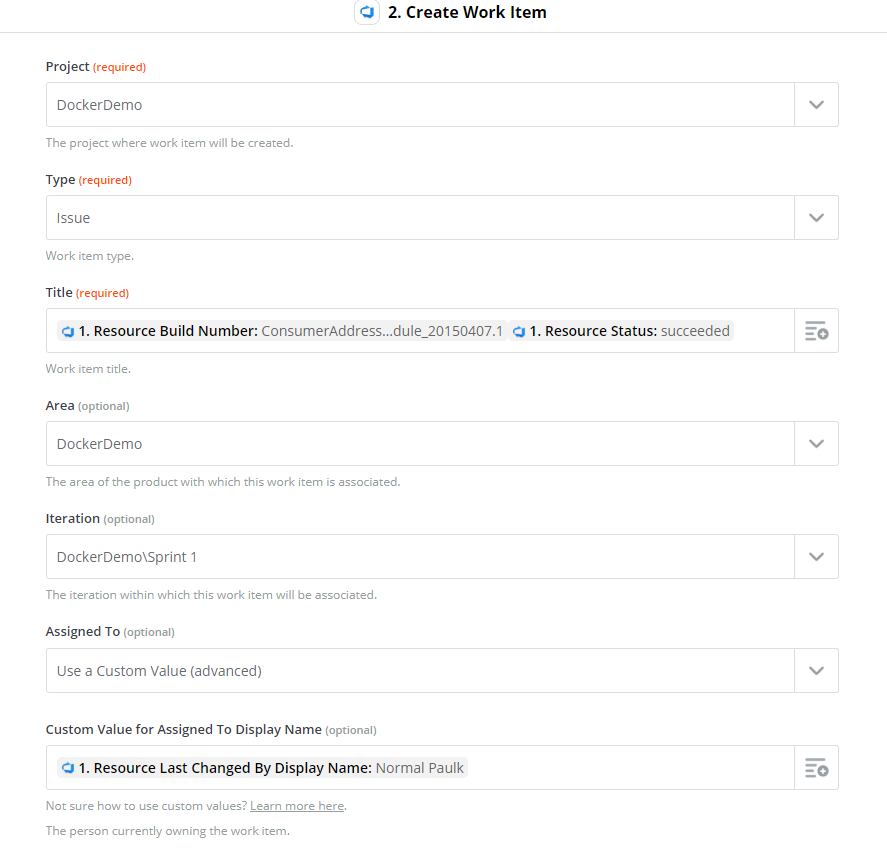
* Select **Completed build** for Choose trigger event option and click on Continue
* Provide your Azure account details and click on Continue
* Provide Project, Build definition and build status details for **Customize Build** step and click on Continue.



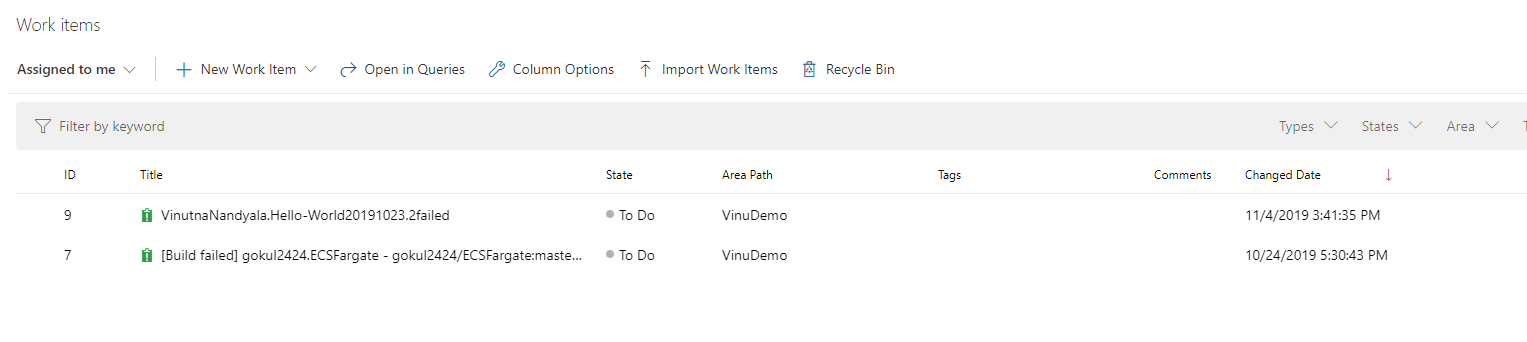
* Click on next step **Do This** and search for azure devops and select



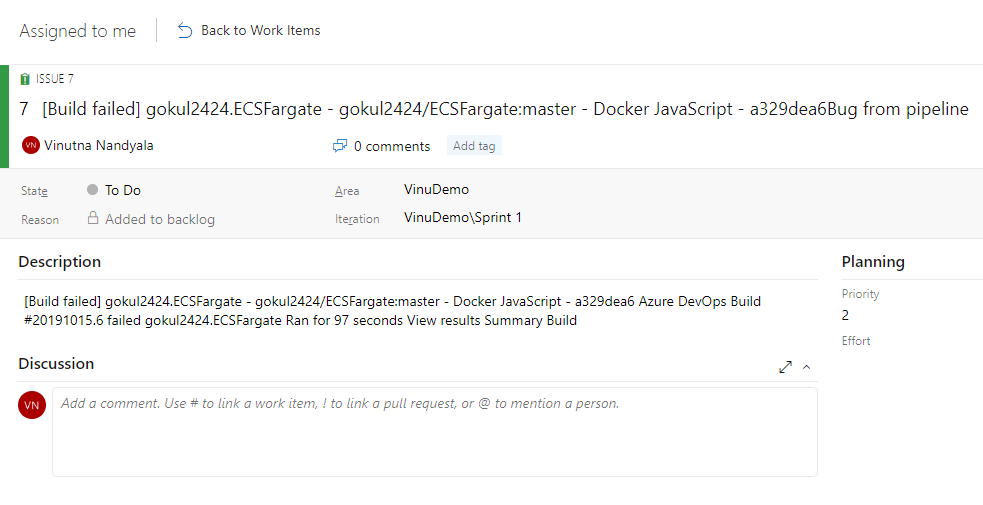
* Select create work item for **Choose Action event** task and click on Continue
* Provide your Azure devops account details and click on Continue
* Provide Project name , Area and work item type details and click Continue



* If any build failed, the work item type **issue** logged in Azure boards like below



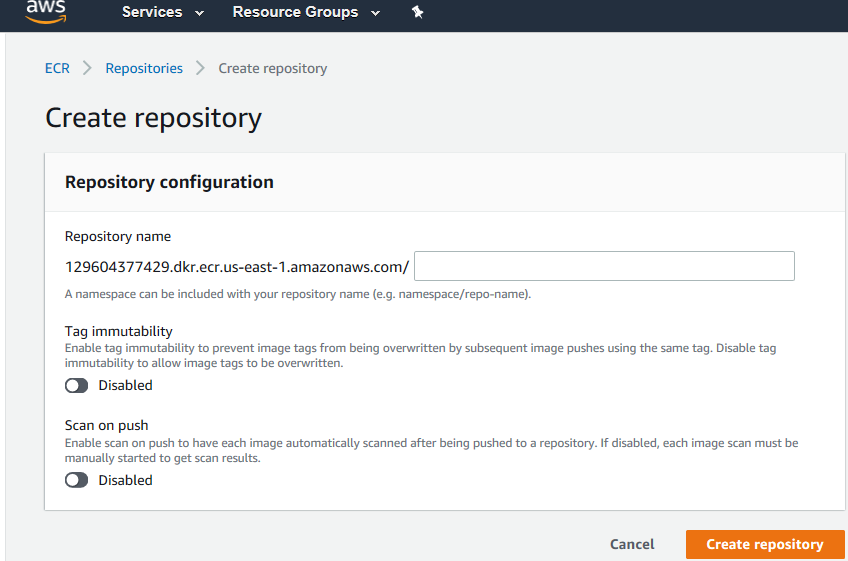
* If you click on work item you can see details of the issue



**6. Creating ECR and ECS in AWS**

**6.1 ECR**

* Login to AWS and click on services and select ECR.
* Click on create repository and give the name for the repository and click create repository



**6.2 Deploy Container to ECS**

**6.2.1 Create a Load Balancer:**

**Step1:**

* [Select a **Load Balancer Type**](https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-getting-started.html#select-load-balancer-type)
* Choose **Create Load Balancer**.
* For **Classic Load Balancer**, choose **Create**.

**Step2:**

[Define Your Load Balancer](https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-getting-started.html#define-load-balancer)

* Type a name for your load balancer.
* For Available subnets, select at least one available public subnet using it’s add icon.

**Step3:**

[Assign Security Groups to Your Load Balancer in a VPC](https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-getting-started.html#select-vpc-security-group)

* Type a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your load balancer to use.

**Step4:**

[Configure Health Checks for Your EC2 Instances](https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-getting-started.html#configure-health-check)

* On the Configure Health Check page, leave Ping Protocol set to HTTP and Ping Port set to 80.

**Step5:**

Register EC2 instance with your load balancer

* Select the instances to register with your load balancer.

**Step6:**

Create and verify load balancer

**6.2.2 Creating a cluster**

**Step1**:

Select Cluster template

* In the navigation pane, choose **Clusters**.
* Choose Create Cluster and Select cluster compatibility Networking only

**Step2**:

Configure Cluster

* Choose a **Cluster name**.

**6.2.3 Create a new task definition**

**Step1**:

Select launch type compatibility

* Choose Fargate

**Step 2**:

Configure task and container definition.

* Type a name for your task definition
* Choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.
* For Task execution **IAM role**, either select your task execution role
* Choose a value for **Task memory (GB)** and **Task CPU (vCPU)**

**6.2.4 Create Container**

**Step1**:

Choose Add container.

**Step2**:

Configure container

* Provide container name, image path to be used and the port mappings.

**6.2.5 Create Service**

**Step1**:

Configure Service

* Review the task definition, and choose **Actions, Create Service**.
* Fill out the following parameters accordingly:

**Cluster Service name**

**Service type**

**Number of tasks Minimum healthy percent**

**Maximum percent**

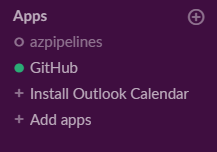
**Step2:**

Configure Network

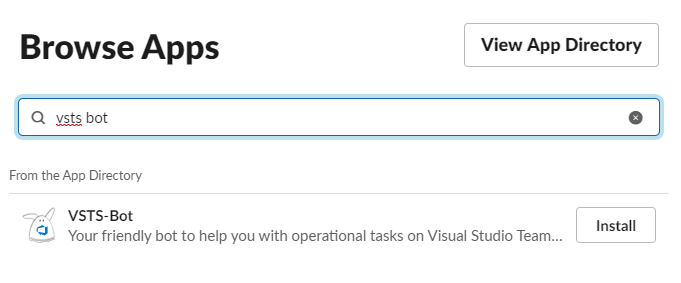
* If you selected the Fargate launch type, select the VPC that the Fargate tasks should use.
* Choose the available subnets for your service task placement.
* Choose the load balancer type to use with your service: Application Load Balancer.
* Choose the container and port combination from your task definition that your load balancer should distribute traffic.

**7. VSTS Bot Integration in Slack to trigger build from Slack**

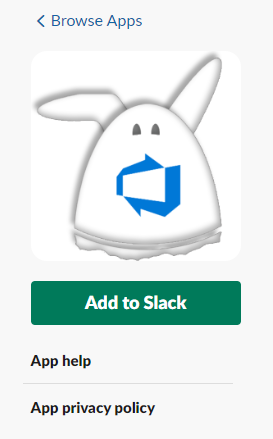
* Login to slack workspace and click on Apps and search for VSTS Bot



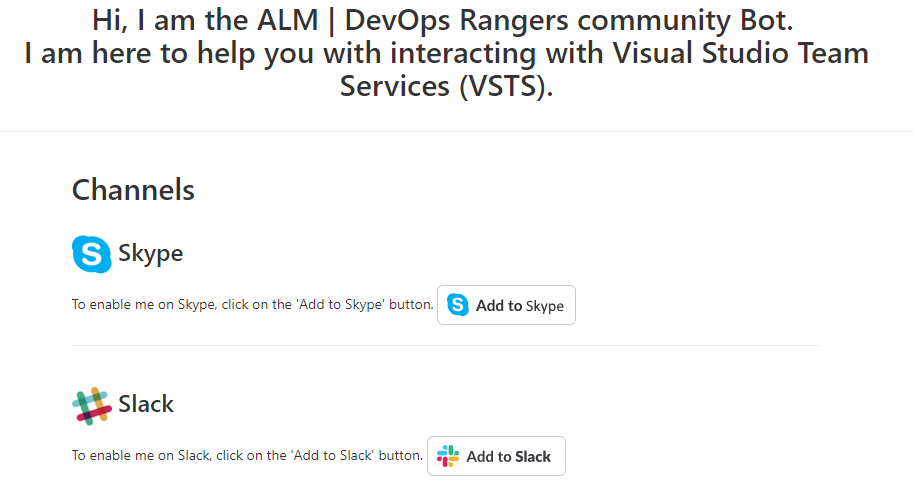
* Click on install



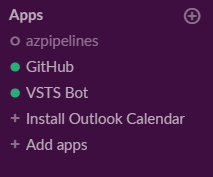
* Click on Add to slack



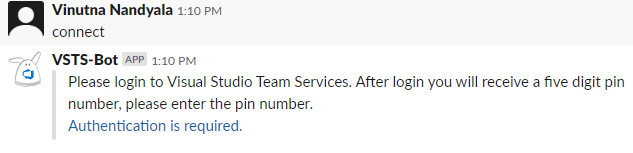
* Now again click on add to slack and in next window click on Allow



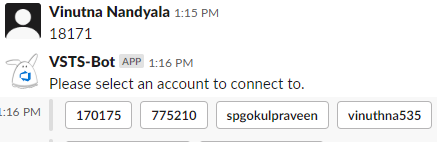
* Now go to Slack workspace. Under the Apps you can see VSTS Bot



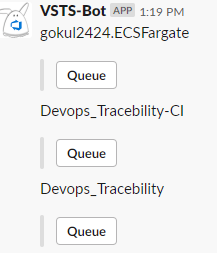
* Click on VSTS Bot and enter **Connect** command. You can see the text like below. Now you have to login to your Azure account to get PIN number. Click on Authentication is required and enter pin number in slack



* Now select which account you want to connect and select the project



* Enter **builds** command in slack, you can see list of pipelines available under that project



* Click on **Queue** which pipeline you want run. Go to Azure devops account, you can see pipeline is running.

**8. Troubleshooting steps**

**Issue1**

While integrating slack with Azure pipeline we got issue with slack token as we are giving slack token value directly in yml file. We got mail from slack as **Notification of disabled Slack token**

**Resolution**

We have created one variable for slack token in Azure and passed that variable in yml file

**Issue2**

While building the Docker image, we got error like **not able to tag Docker image**

**Resolution**

After changing the image name issue got resolved