**Cloud Engineer Assignment: CI/CD Pipeline on AWS using GitHub Actions and Docker**

IDEAS to achieve the challenge complete

1. The manual way
2. GitHub actions

**GitHub Actions**

1. Normally if we done it locally, we would need to configure AWS in the local system to push the Image into ECR that’s been discussed in the below manual way
2. But by through GitHub actions, we would have configured AWS in GitHub through secret variables
3. Go to your repository in GitHub, click on repository settings -> secrets & variables
4. Now you would have to add these variables
5. A screenshot of a black screen

   Description automatically generatedWhen you fill it would look something like this
6. Coming to the workflow file where in we would have create a file
7. So as per the requirements when you do it manually right you would need a

* Machine
* Docker installed
* And a volume with source code in it
* AWS configured in a system

1. So, in this case git repository acts as a source code
2. In workflow we mention the implementation of ubuntu OS to complete the GitHub action
3. Also, we would need to configure AWS there would be specific step mentioned in workflow file
4. Also same goes for creating, building a image and pushing it into repositories all at one place

**Workflow file**

A screenshot of a computer program

Description automatically generated



As you see the picture

1. The code effective in the **main branch**
2. It is mentioning that whatever the action that needs to be happen that need to be run on git provided ubuntu machine. Git provides micro machines with multiple image files to do operations in the case we take **ubuntu**
3. We check if there are conflicts over the referred branch if conflicts we break the pipline
4. So normally we need sudo permission to do any operations using docker
5. Now the part where we use secret variables and connect AWS

* Mainly we use the secret variables to protect over resources from potential hacker or any one who can misuse that resources

1. Connect with ECR with docker login command
2. That the command to build the image
3. Docker command to push the repository to ECR

**THE MANUAL WAY**

1. Create the Flask application with the pytests
2. Create a docker file

FROM python:3.12.5-bookworm

WORKDIR /app

COPY . /app/

EXPOSE 80

RUN pip install -r requirements.txt

RUN python3 -m pytest

CMD ["python", "app.py"]

1. Create a compose file [ optional ]

version: '3.8'

services:

  application:

    build:

      context: .

      dockerfile: Dockerfile

    ports:

      - "80:80"

    volumes:

      - .:/app

1. Now create a docker image

**docker compose . -t any name in general**

1. To push the code in to ECR we would have to create a repository in ECR

* Open AWS Console
* Login with credentials
* Search for the service of ECR [ Elastic container Registry]
* Click on create repository
* You would have to create a repo name
* Once you create that you would be taken to dashboard
* Copy the repository link

1. Go to the system and run this command in order to tag them

**docker tag <your image name > < Created repository link>**

1. Now push the tagged image to ECR

**Docker push <created repository link>**

**Implementation of ECS**

**Steps to reproduce**

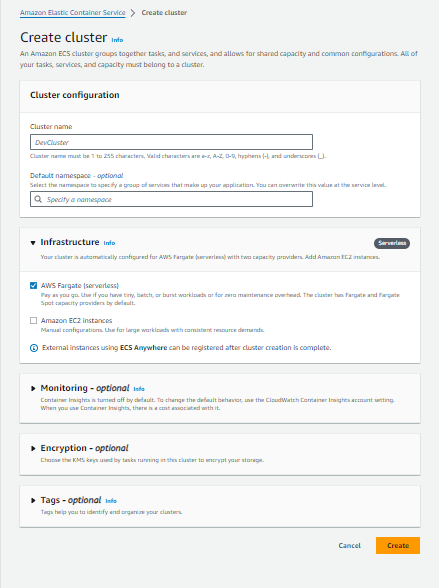
1. Open AWS console
2. Open ECS [Elastic container service ]
3. Click on create cluster
4. Provide cluster name
5. Also provide namespace it’s optional but it you don’t provide it would normally take the name of your cluster name for namespace
6. In Infrastructure choose AWS Fargate (serverless)

* Useful when you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

1. Click on monitoring – optional

* Click on container insights
* Useful because it automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects

1. You can opt for encryption if you want to
2. And tags they are optional as well
3. Click on create [ it might take few seconds to create cluster]



1. Dashboard would look something like this
2. Note when you create cluster automatically CloudFormation would be assigned
3. And we would have to add manually add the cloud watch

A screenshot of a computer

Description automatically generated

1. Now click on Task Definitions in her we would provide the information of what the task is and assign resources that can be used to host the container

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Now open a new tab go to Iam > policies
2. Create a new policy

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "VisualEditor0",

"Effect": "Allow",

"Action": [

"cloudwatch:\*",

"logs:\*",

"cloudformation:\*",

"ec2:\*"

],

"Resource": "\*"

}

]

}

1. After creating a policy attach it to the task definition
2. Next comes up with place of Container allotment

A screenshot of a computer screen

Description automatically generated

1. Fill it accordingly to your image from ECR and the port and stuff for my application details are

ECR link

975050024946.dkr.ecr.ap-northeast-2.amazonaws.com/challenge/flaskapp

1. In the name column you can literally write for the container name to be in my case I use flask-application
2. Arrange the ports
3. You can additionally add health check for the repo it goes something like

curl -f http:localhost/

we didn’t mention any port because 80 is default port used by system

1. You can choose for Monitoring that is optional
2. Click on it and you could see trace collection and metrical collection by Cloud watch

A screenshot of a computer

Description automatically generated

1. Tags are optional
2. Now wait for few minutes while task definition is created
3. By the time go back to dashboard of your cluster
4. Now click on create service it would look something like this

A screenshot of a computer

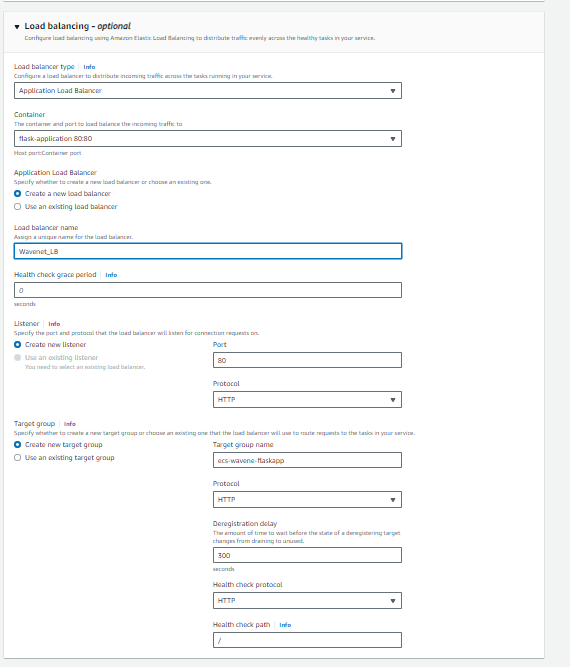
Description automatically generated

A screenshot of a computer

Description automatically generated

Load balancing

1. While configuring this click on load balancing from the bottom
2. Click on drop down choose Application Load Balancer
3. Write any name for load balancing



1. You can opt for Service autoscaling if you want after then click on create and go back to the dashboard of task definition there would be seeing an option for deployment
2. Click on it and create a service

**Dashboard of cluster**

* The dashboard would look something like this

A screenshot of a computer

Description automatically generated

Events after creating service in CloudFormation

A screenshot of a computer

Description automatically generated

It might take a while to create that container mean while go back to dashboard and then you get to see this

A screenshot of a computer

Description automatically generated

Click on your service

A screenshot of a computer

Description automatically generated

**Cloudwatch**

Collected Logs of container

A screenshot of a computer

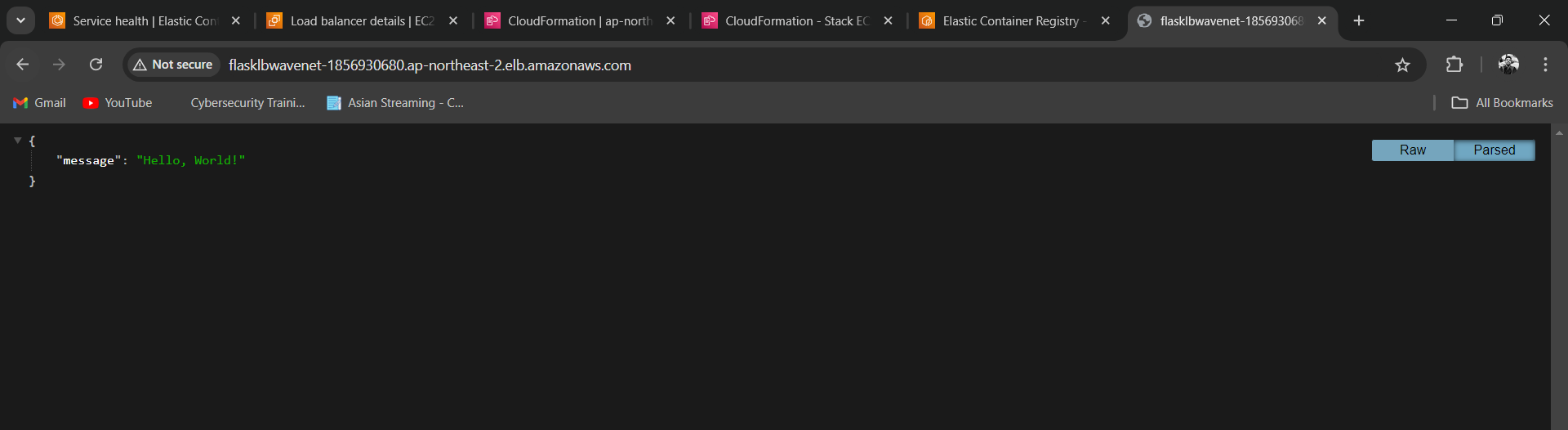
Description automatically generated

**Task Completion**

1. Now in the bottom there you could see load balancer right click on it.
2. It would take you to load balancer
3. Now copy that dns name and paste it in browser you would see the output of your flask application that’s how you know the task/ challenge is completed
4. Expected output is hello world

A screenshot of a computer

Description automatically generated



**Bonus**

BLUE/GREEN Deployment can be configured while creating service

