FastAPI Application Deployment Using AWS ECS and CDK

Table of Contents

A step-by-step document that outlines the entire process, from containerizing the Fast API application to deploying it on AWS ECS using CDK, along with details on networking, security configurations, and redeployment steps.

1. Containerizing the FastAPI Application:

To containerize the FastAPI application, we used Docker to create a container image that can run on any machine.

Steps:

1.1 Create the FastAPI Application :

- Write a basic FastAPI application (main.py):

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello, World!"}
```

1.2 Create a Dockerfile :

- Create a Dockerfile in the same directory as main.py. This will define the steps to build the image:

```
# Use official Python base image
FROM python:3.9

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container
COPY . /app

# Install FastAPI and Uvicorn
RUN pip install fastapi uvicorn

# Command to run the FastAPI application
```

```
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

1.3 Build the Docker Image :

- Run the following command to build the Docker image: (bash)

```
docker build -t fastapi-app.
```

1.4 Test the Docker Image Locally :

- Test the application locally to make sure everything works:(bash)

```
docker run -p 8000:80 fastapi-app
```

Access the application at http://localhost:8000.

2. Setting Up AWS Infrastructure Using CDK

The next step is to set up the AWS infrastructure using AWS Cloud Development Kit (CDK). The goal is to provision the necessary resources, including an ECS cluster and a load balancer.

Steps:

2.1 Install CDK :

- Install the AWS CDK: (Bash)

```
npm install -g aws-cdk
```

2.2 Create a CDK Project :

- Initialize a CDK project: (Bash)

```
mkdir fastapi-ecs
cd fastapi-ecs
cdk init app --language typescript
```

2.3 Define the ECS Cluster and Services:

- The app.py file to create the ECS cluster, load balancer, and Fargate service:

```
from aws_cdk import (
  Stack,
  App,
  Environment,
)
from aws_cdk import aws_ecs as ecs
from aws_cdk import aws_ec2 as ec2
from aws_cdk import aws_ecr as ecr
from aws_cdk import aws_ecs_patterns as ecs_patterns
from constructs import Construct  # Needed in CDK v2

class FastapiEcsStack(Stack):
  def __init__(self, scope: Construct, id: str, **kwargs) -> None:
    super().__init__(scope, id, **kwargs)

    # Create a VPC (Virtual Private Cloud)
    vpc = ec2.Vpc(self, "FastapiVpc", max_azs=2)  # Max 2 availability zones

    # Create an ECS Cluster within the VPC
    cluster = ecs.Cluster(self, "FastapiCluster", vpc=vpc)

    # Reference the ECR repository
    ecr_repo = ecr.Repository.from_repository_arn(
      self,
      "EcrRepository",
      "arn:aws:ecr:eu-central-1:364119026921:repository/devops"
    )
```

```
        # Create Fargate Service with Load Balancer
        ecs_patterns.ApplicationLoadBalancedFargateService(self, "FastapiService",
            cluster=cluster,
            task_image_options={
                "image": ecs.ContainerImage.from_ecr_repository(ecr_repo),
                "container_port": 8000  # Port exposed by the FastAPI app
            }
        )
# Entry point for AWS CDK
app = App()
FastapiEcsStack(app, "FastapiEcsStack", env=Environment(account="364119026921", region="eu-central-1"))
app.synth()
```

2.4 Deploy the CDK Stack :

- Run the following commands to deploy the stack: (Bash)

  Cdk synth

  Cdk deploy

3. Deploying the Application to ECS.

After the infrastructure is set up using CDK, you will deploy the containerized FastAPI application to ECS.

Steps:

3.1 Push Docker Image to ECR:

- Tag the Docker image with the ECR repository URI:(bash)

  docker tag fastapi-app: latest <ECR-Repo-URI>: latest

- Push the image to the ECR repository:(bash)

  docker push <ECR-Repo-URI>: latest

3.2 Update ECS Service :

- The ECS service created by the CDK will automatically pull the latest image from the ECR repository when you redeploy the stack using:(bash)

```
cdk deploy
```

4. Handling Networking, Security Groups, and AWS-Specific Configurations

4.1 Networking :

- **VPC**: A VPC was automatically created by CDK with public subnets to host the ECS service.
- **Subnets**: The ECS service and ALB were deployed in public subnets, making the FastAPI application accessible from the internet.

4.2 Security Groups :

- **ECS Security Group**: A security group was configured to allow inbound traffic on port 80 for HTTP access to the FastAPI application.
- **Load Balancer Security Group**: The load balancer's security group allowed public HTTP traffic.

4.3 Load Balancer :

- An Application Load Balancer was used to expose the FastAPI application to the public via a DNS name, distributing traffic evenly between Fargate tasks.

5. Redeploying the Application After Modifications

To redeploy the application after making changes (e.g., modifying the response in main.py), follow these steps:

Steps :

5.1 Modify Application Code :

- Edit the main.py file:

```
from fastapi import FastAPI
app = FastAPI()
@app.get ("/")
def read_root ():
  return {"message": "Hello, World, Satya!"}
```

5.2 Rebuild Docker Image :

- Rebuild the Docker image with the latest code:(bash)

```
docker build -t fastapi-app.
```

5.3 Push the Updated Docker Image to ECR :

- Push the updated image to ECR:(bash)

```
docker tag fastapi-app: latest <ECR-Repo-URI>: latest
docker push <ECR-Repo-URI>: latest
```

5.4 Redeploy CDK Stack :

- Run the following command to redeploy the updated application:(bash)

```
cdk deploy
```

6. Challenges Encountered and Solutions

6.1. Port Allocation Issue on Localhost :

- **Challenge**: While testing locally, the FastAPI app would fail due to the port being already allocated.
- **Solution**: I identified that another process was using port 8000 and resolved the issue by either stopping the other process or running the Docker container on a different port.

6.2. Networking Issues on Mobile :

- **Challenge**: When accessing the application from a mobile device, the service was not reachable.
- **Solution**: I ensured the correct security group settings were in place to allow HTTP traffic and confirmed that the load balancer's DNS name was resolving correctly.

7. For reference I have attached some image in the S3

https://task0101.s3.eu-west-2.amazonaws.com/Elastic+Container+Registry.png

https://task0101.s3.eu-west-2.amazonaws.com/Elastic+Container+Service.png

https://task0101.s3.eu-west-2.amazonaws.com/Fastap-Fasta-WsMtt5ag8mmh-1492385024.eu-central-1.elb.amazonaws.com.png

https://task0101.s3.eu-west-2.amazonaws.com/load+balncer.png

https://task0101.s3.eu-west-2.amazonaws.com/Resource+raod+map.png

https://task0101.s3.eu-west-2.amazonaws.com/Screenshot+2024-08-29+130330.png

https://task0101.s3.eu-west-2.amazonaws.com/target+group.png

The fastapi link: http://fastap-fasta-wsmtt5ag8mmh-1492385024.eu-central-1.elb.amazonaws.com/