2.0 Analysis of Monolithic Application & Microservices Conversion

2.1 Original Monolithic Application Architecture

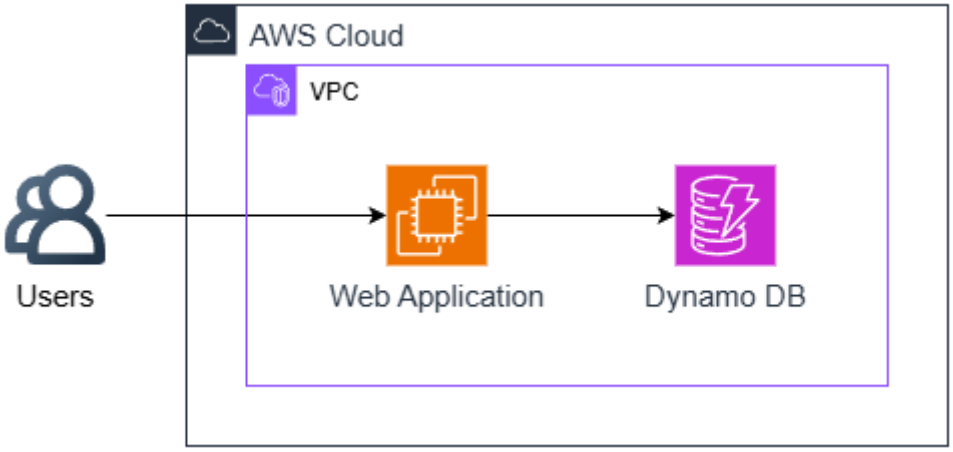


Figure 1: Original Monolithic Application Architecture

Originally, our web application was running on a single EC2 instance. Users direct connect to EC2 to access the web application, the EC2 instance perform CRUD to the Dynamo DB.

When a developer needs to update the web application, developer needs to connect to EC2 and manually replace the code, which wastes time and has services downtime.

2.2 Microservices Application Architecture

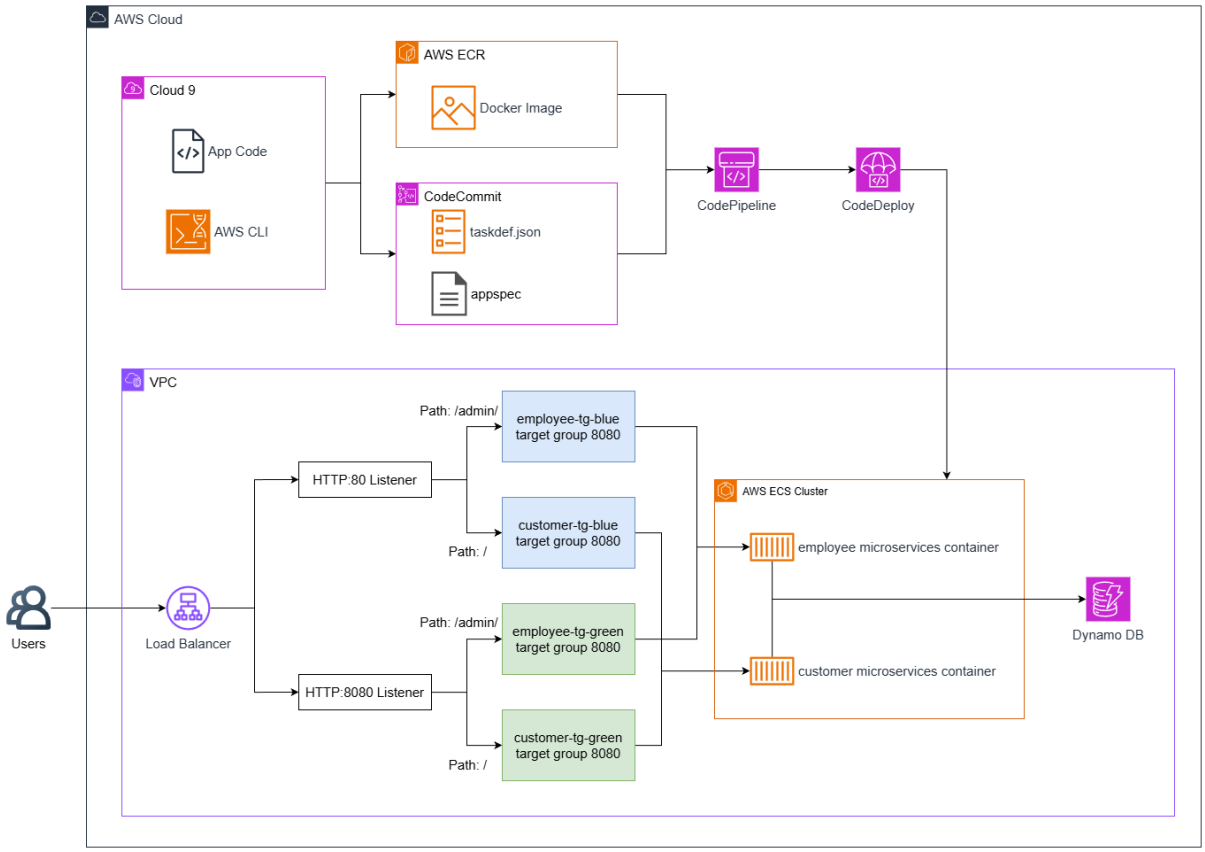


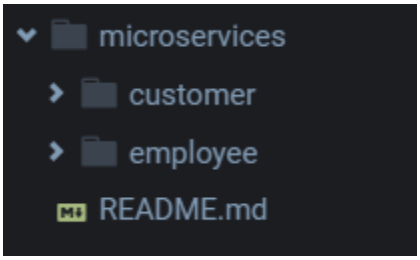
Figure 2: Microservices Application Architecture

After converting the Monolithic Application into Microservices Application. Users access the web application by accessing the Load Balancer. The Load Balancer will route the requests based on the URL path to the appropriate container (employee or customer).

This architecture also includes CI/CD pipeline. When a developer makes changes and pushes a new Docker Image to AWS ECR, it will trigger CodePipeline. The pipeline tells CodeDeploy to launch a new test version of the services. After testing via port 8080, CodeDeploy swaps traffic, promoting test to production with zero downtime.

How the Monolithic Application Converted into Microservices Application:

Before we uploaded the source code files to Cloud9, our original HELP EMS website source code was modified and converted from a monolithic application into two separate microservices. We created a directory named "microservices" and added two subfolders within it, representing our two microservices: "customer" and "employee". Each subfolder contains the respective code for its microservice.



The functionality of these 2 microservices is shown in below table:

Primary User	Microservice Functionality	Access Level (CRUD)
Customer	• Register and login as **attendee only** • View public upcoming events only • View ticket types and seat availability • Create payments and seat bookings • Create and delete waitlist entries • View payment history and booking details	**Events/Tickets:** Read only **Seats:** Read + Update **Payments:** Create + Read **Seat Bookings:** Create + Read **Waitlist:** Create + Read + Delete **Users:** Create (self-registration)
Employee	• Register and login as **admin or event organizer** • For admin: they can create event organizers with email notifications • For event organizer: Reset passwords for first-time organizers • For event organizer: Create events with image uploads • For event organizer: View all events (published + unpublished) • For event organizer: Create and delete ticket categories • For event organizer: Create and assign seat sections	**Events:** Create + Read + Update **Tickets:** Create + Read + Update + Delete **Seats:** Create + Read + Update + Delete **Users:** Create + Update (password reset)

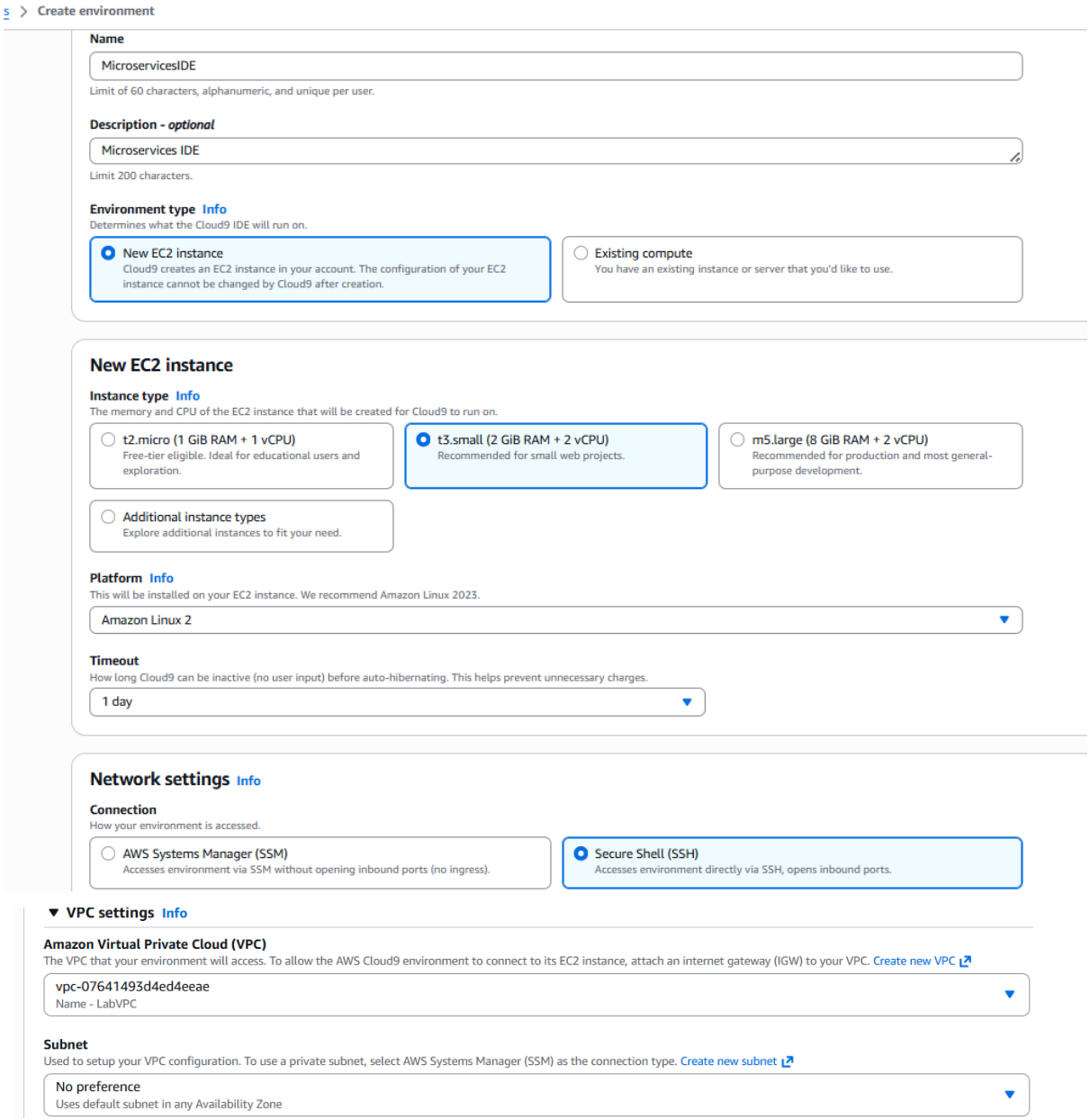
	• For event organizer: Publish events to public	

3.0 AWS Implementation

Phase 1: Create Cloud 9 Environment and Git Repository

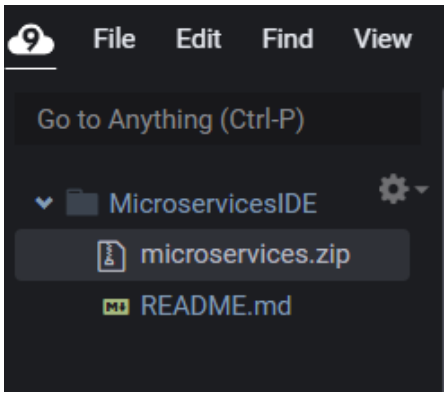
Task 1: Create an AWS Cloud 9 IDE

1. Navigate to AWS Cloud 9 → Create an AWS Cloud 9 instance with the below info as shown in picture:

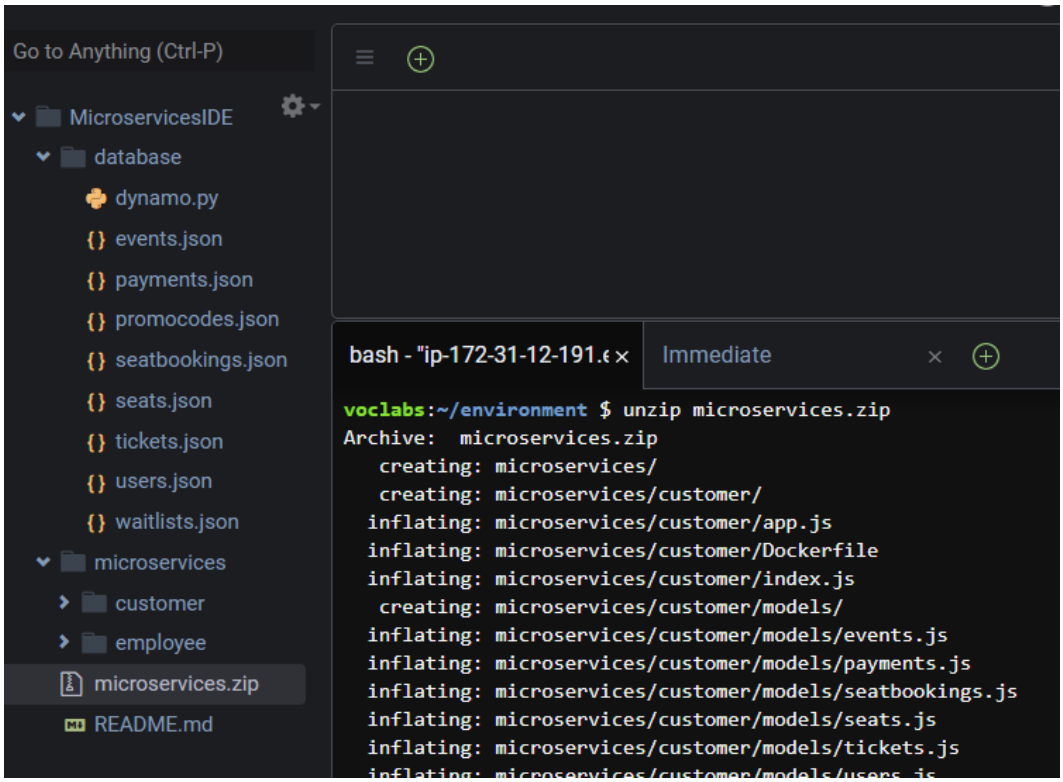


Task 2: Upload Source Code & Create GIT Repository

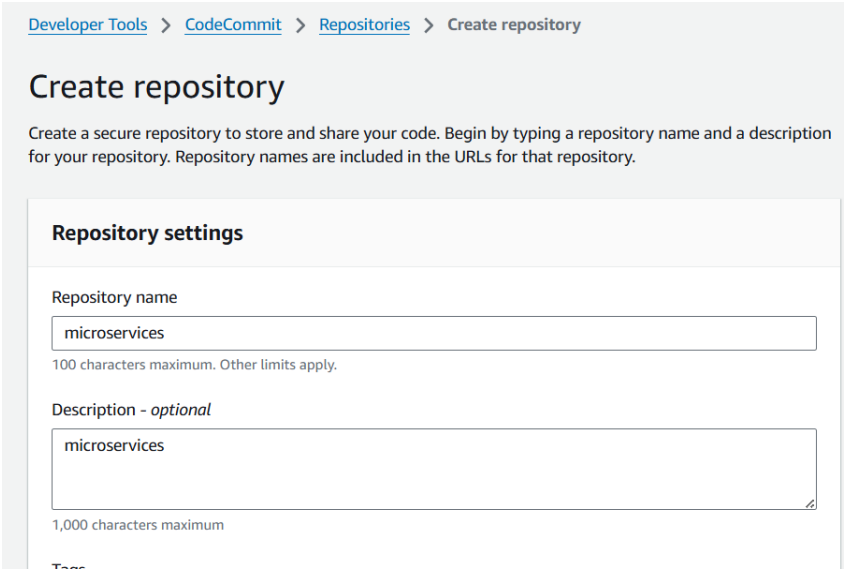
1. Drag and drop the edited source code zip file from local device (for our HELP EMS website) to Cloud IDE just created.



2. Unzip the uploaded file by running this code in the terminal: **unzip microservices.zip**



3. Create a CodeCommit repository named "**microservices**":



4. Initialize the Git repository and push the code:

```
voclabs:~/environment/microservices $ cd ~/environment/microservices
voclabs:~/environment/microservices $ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ec2-user/environment/microservices/.git/
voclabs:~/environment/microservices (master) $ git branch -m dev
voclabs:~/environment/microservices (dev) $ git add .
voclabs:~/environment/microservices (dev) $ git commit -m 'Add existing source code for customer and employee microservices'
[dev (root-commit) 51f1d42] Add existing source code for customer and employee microservices
 Committer: EC2 Default User <ec2-user@ip-172-31-12-191.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 73 files changed, 1895 insertions(+)
 create mode 100644 customer/Dockerfile
 create mode 100644 customer/app.js
 create mode 100644 customer/index.js
 create mode 100644 customer/models/events.js
 create mode 100644 customer/models/payments.js
 create mode 100644 customer/models/seatbookings.js
 create mode 100644 customer/models/seats.js
 create mode 100644 customer/models/tickets.js
 create mode 100644 customer/models/users.js
voclabs:~/environment/microservices (dev) $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
voclabs:~/environment/microservices (dev) $ git push -u origin dev
```

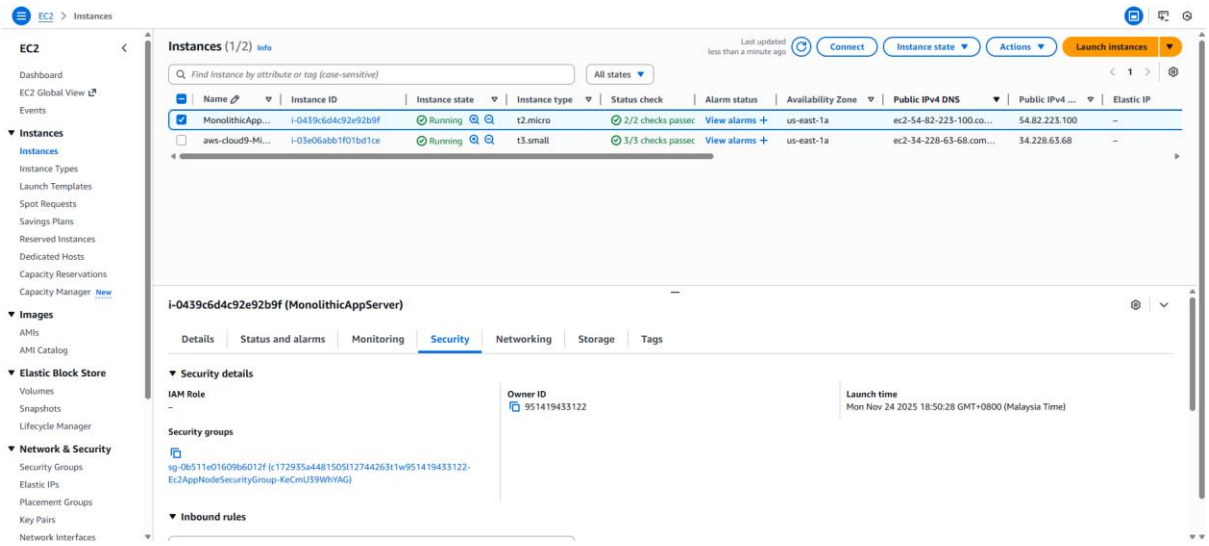
```
Enumerating objects: 51, done.
Counting objects: 100% (51/51), done.
Delta compression using up to 2 threads
Compressing objects: 100% (49/49), done.
Writing objects: 100% (51/51), 3.38 MiB | 10.88 MiB/s, done.
Total 51 (delta 11), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 * [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
```

```
voclabs:~/environment/microservices (dev) $ git config --global user.name "zhengye"
voclabs:~/environment/microservices (dev) $ git config --global user.email "zhengye@gmail.com
```

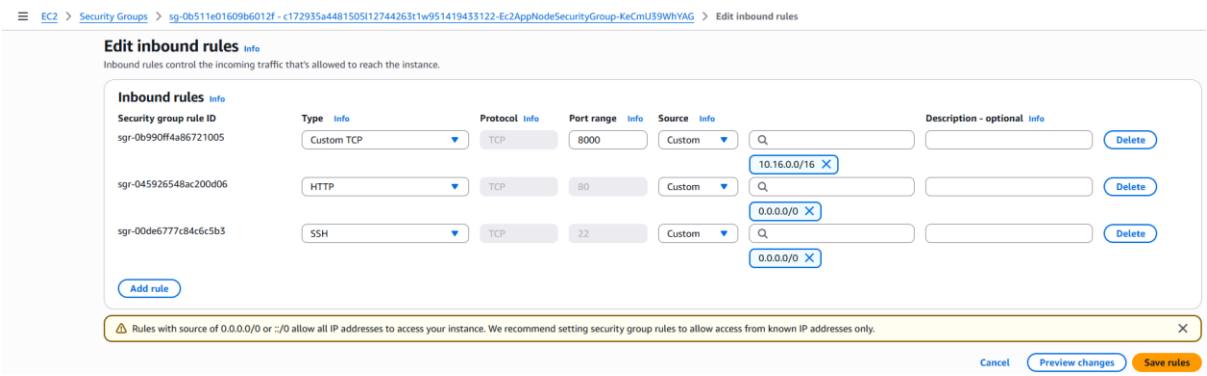
- These commands set up the microservices folder as a Git repository and prepare it for use with AWS CodeCommit.
- The folder is initialized as a Git repo, the default branch is renamed to dev, and all files are staged and committed.
- A remote connection to the CodeCommit repository named microservices is then created.
- Finally, the committed code in the dev branch is pushed to the remote repository.

Task 3: Create Dynamo DB

1. Navigate to EC2 Console, select instance "**MonolithicAppServer**", then select security tab



2. Select the security groups and click “Edit inbound rules”. Add new rule with port range 8000, IP range 10.16.0.0/16 and save rules.



3. From the AWS Details panel on this lab instructions page, download the labsuser.pem file to your local computer.

4. Upload the .pem file to AWS Cloud9 IDE, and use “chmod 400 labsuser.pem” command to set the proper permissions on the file.

5. Type in terminal “scp -r -i ./labsuser.pem ./ setup-dynamo.sh ubuntu@ <app-server-ip>:/home/ubuntu/”, replace app-server-ip to actual ip. It copy setup-dynamo.sh to app server.

6. Type in terminal “ssh -i ./labsuser.pem ubuntu@<app-server-ip> "chmod +x ./setup-dynamo.sh && ./setup-dynamo.sh"”, replace app-server-ip to actual ip. It execute the setup.sh.

```
voclabs:~/environment $ chmod 400 labsuser.pem
voclabs:~/environment $ scp -r -i ./labsuser.pem ./setup-dynamo.sh ubuntu@10.16.10.66:/home/ubuntu/
setup-dynamo.sh                                                                100%  494   492.1K
voclabs:~/environment $ ssh -i ./labsuser.pem ubuntu@10.16.10.66 "chmod +x ./setup.sh && ./setup-dynamo.sh"
```

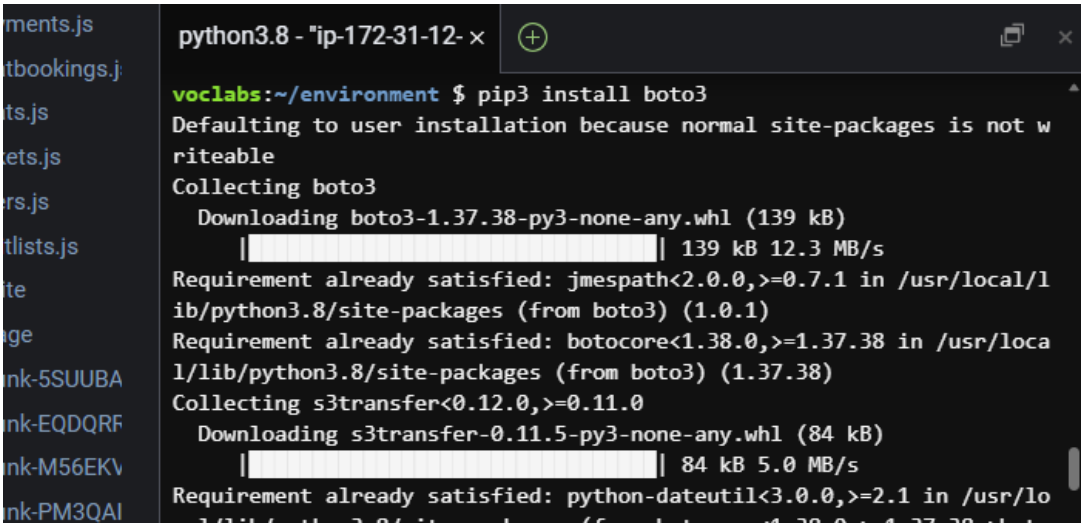
7. The setup-dynamo.sh script will install java and DynamoDB local, after that run the DynamoDB local.

setup-dynamo.sh

```
#!/bin/bash
PID=$(pgrep -f "sudo node index.js")
sudo kill $PID

sudo apt install openjdk-21-jre-headless
sleep 2

wget https://d1ni2b6xgvw0s0.cloudfront.net/v2.x/dynamodb_local_latest.zip
unzip dynamodb_local_latest.zip
sleep 2

java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb &

sudo sh -c "cat <<EOF > /etc/rc.local
#!/bin/bash
cd /home/ubuntu/
sudo java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
EOF
"
sudo chmod +x /etc/rc.local
```

8. Install the **boto3 Python package** by running this code in the terminal:

   pip3 install boto3



9. In terminal, direct to the database directory with **cd ~/environment/database**, then run the Python script using **python3 dynamo.py**. This will set up the DynamoDB tables and add sample items to each table.

```
voclabs:~/environment $ cd ~/environment/database
voclabs:~/environment/database $ python3 dynamo.py

Created table: users
Added 8 items to users
Created table: events
Added 6 items to events
Created table: tickets
Added 25 items to tickets
Created table: seats
Added 1541 items to seats
Created table: seatbookings
Added 5 items to seatbookings
Created table: payments
Added 17 items to payments
Created table: waitlists
Added 1 items to waitlists
voclabs:~/environment/database $
```

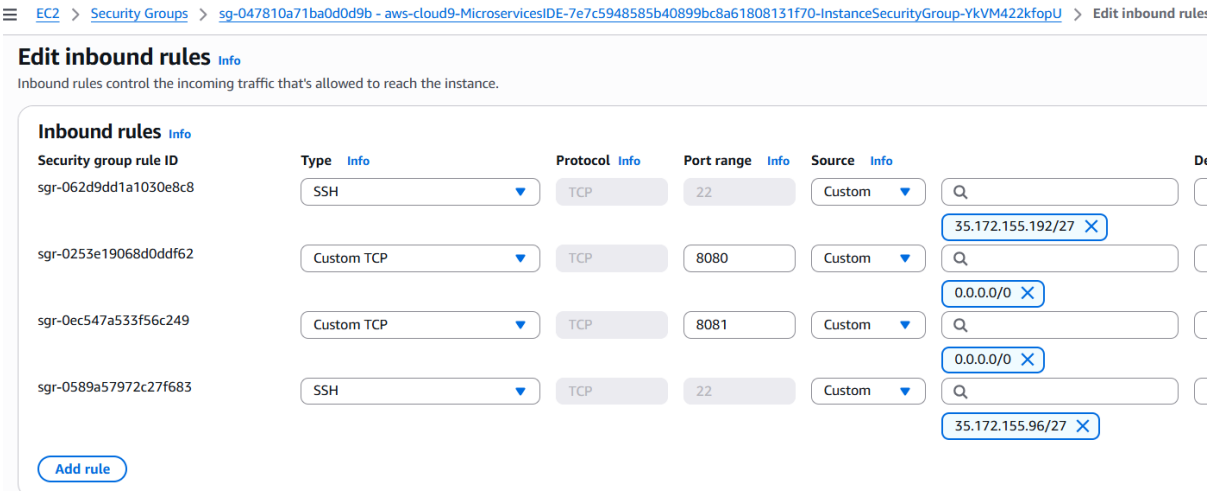
10. Verify that all tables were created by using AWS CLI

```
voclabs:~/environment/database $ aws dynamodb list-tables --endpoint-url http://10.16.10.66:8000
{
    "TableNames": [
        "events",
        "payments",
        "seatbookings",
        "seats",
        "tickets",
        "users",
        "waitlists"
    ]
}
```

Phase 2: Creating Docker Images and ECR Repositories

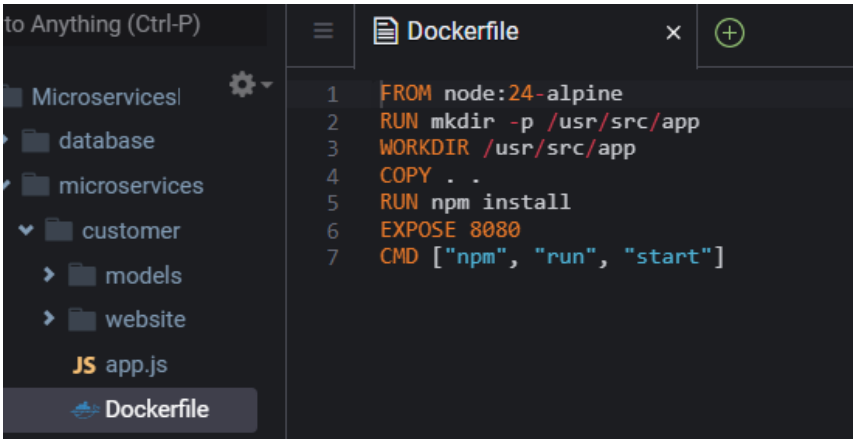
Task 1: Adjust AWS Cloud 9 Instance Security Group Settings

1. Edit inbound network rules by adding two TCP ports 8080 and 8081, then set them can be accessed in Anywhere-IPv4.



Task 2: Create Docker Image for Customer Microservice

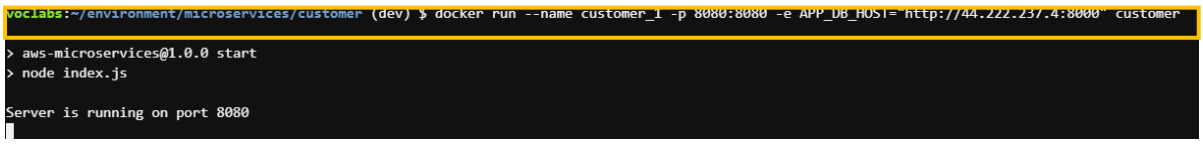
1. In our source code, the Dockerfile was created for customer directory:



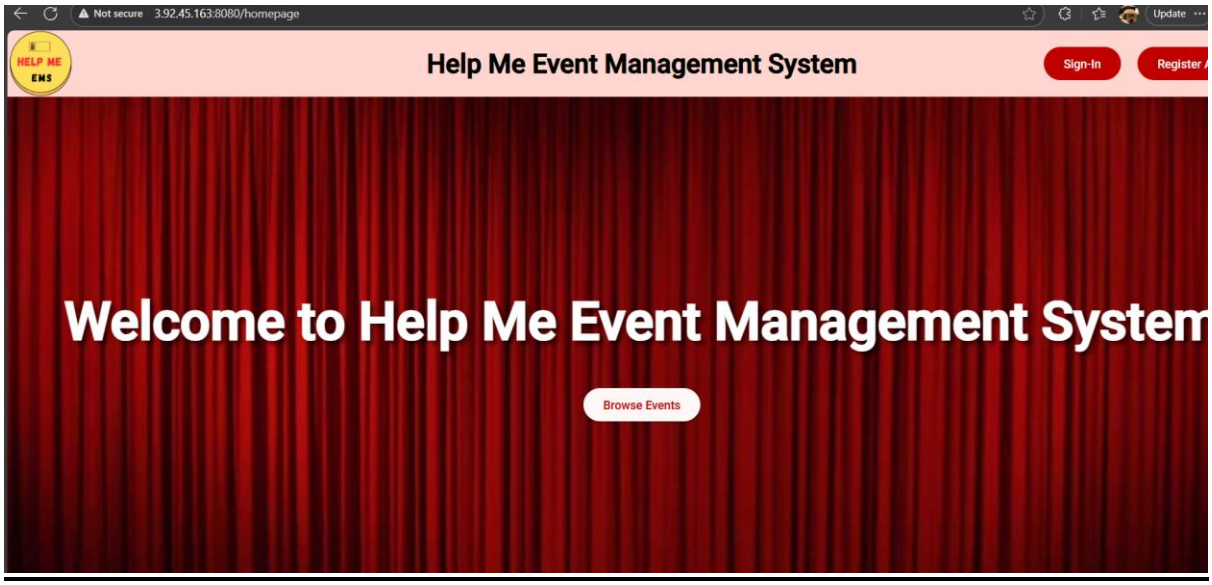
2. Build an image from the Dockerfile within the customer directory we created in the source code:

3. Runs the customer Docker image as a container named **customer_1**. It maps port 8080 from the container to the host and sets environment variables for DynamoDB (APP_DB_HOST) so the microservice can connect to our **local DynamoDB** instance at port 8000 from MonolithicAppServer.

Command: docker run --name customer_1 -p 8080:8080 -e APP_DB_HOST="http://< private IPv4 address of the MonolithicAppServer>:8000" customer

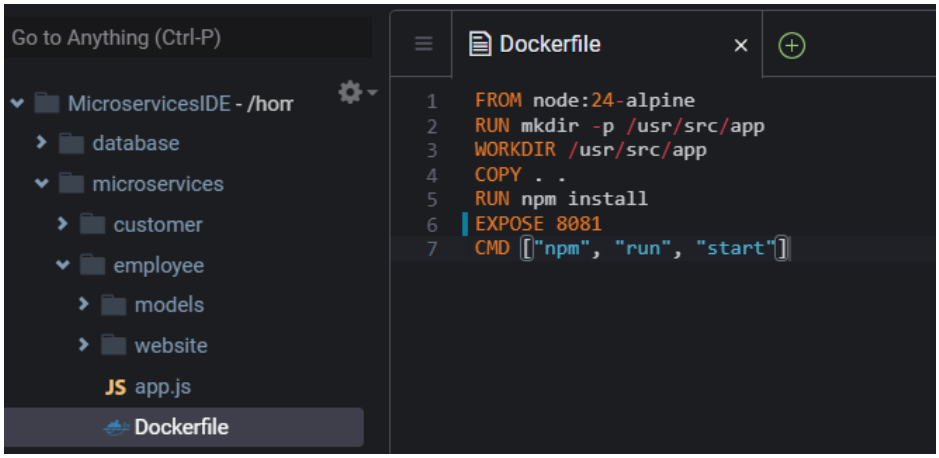


4. Access through the http://<cloud-9-public-IPv4-address>:8080



Task 3: Create Docker Image for Employee Microservice

1. In our source code, the Dockerfile was created for employee directory:



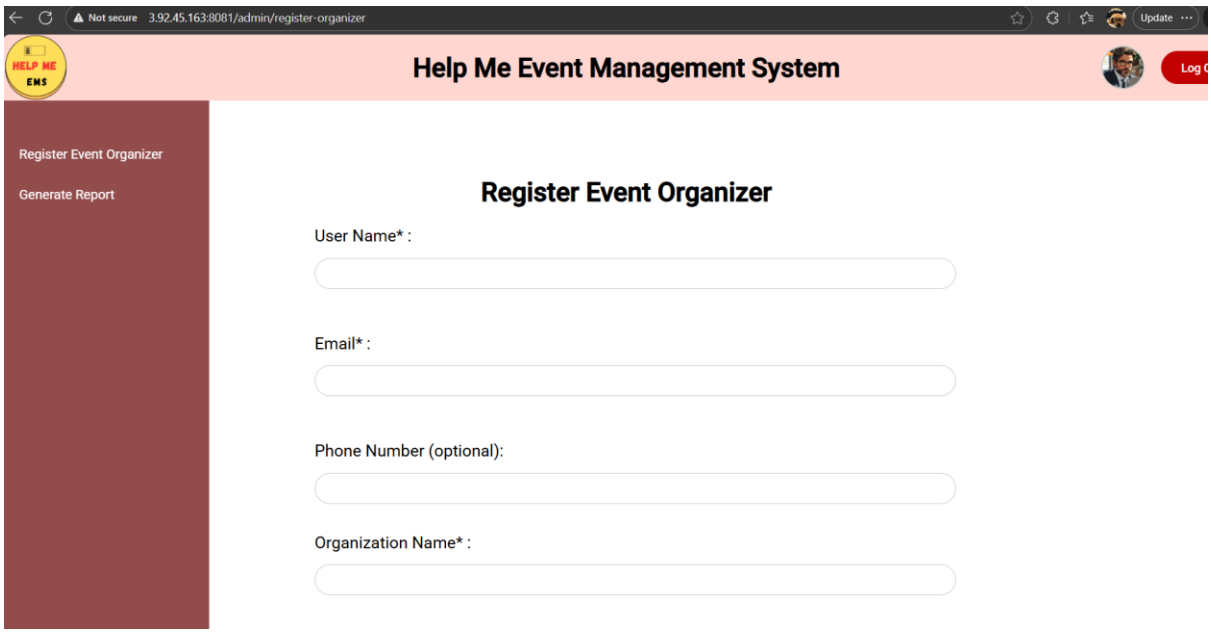
2. Build an image from the Dockerfile within the employee directory we created in the source code:

```
voclabs:~/environment/microservices/customer (dev) $ cd ~/environment/microservices/employee
voclabs:~/environment/microservices/employee (dev) $ docker build --tag employee .
[+] Building 15.1s (10/10) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                           0.0s
 => => transferring dockerfile: 176B                                           0.0s
```

3. Runs the employee:latest Docker image in the background as a container named employee_1.

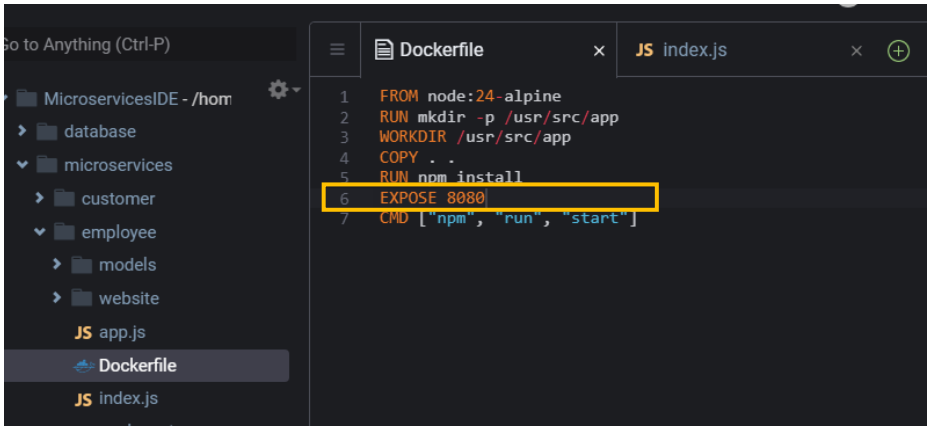
```
voclabs:~/environment/microservices/employee (dev) $ docker run --name employee_1 -p 8081:8081 -e APP_DB_HOST="http://10.16.10.66:8000" employee

> aws-microservices@1.0.0 start
> node index.js

Server is running on port 8081
```

4. Access through the http://<cloud-9-public-IPv4-address>:8081

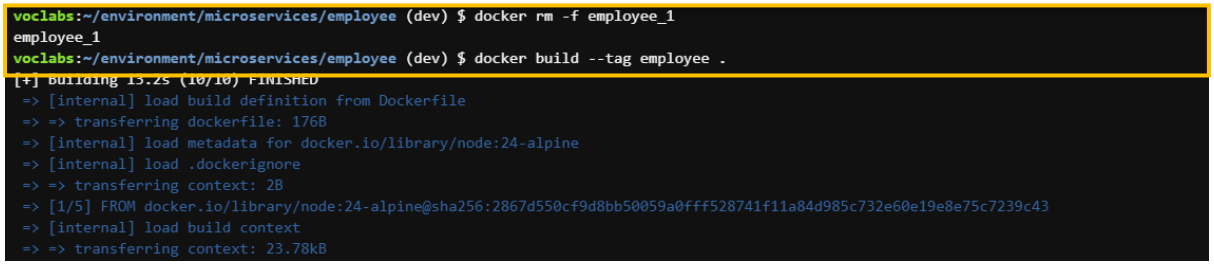


Task 4: Edit Port of the Employee Microservice & Rebuild the Image

1. Change the port from **8081** to **8080** in the **employee/index.js** and **employee/Dockerfile** files



2. Rebuild the new image for this updated source code file

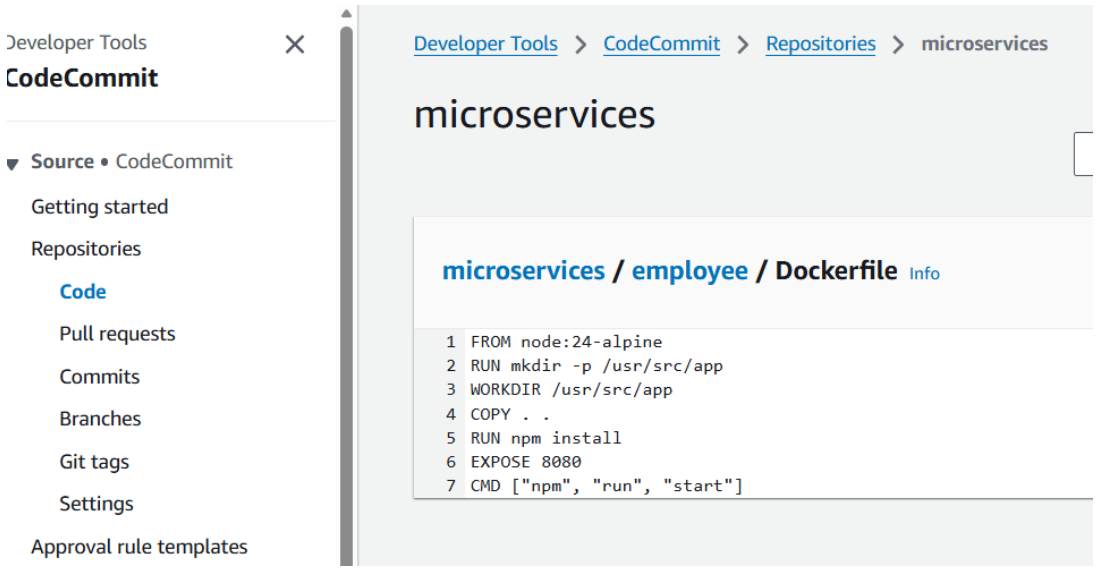


Task 5: Update the Latest Code Changes into CodeCommit

1. Commit and push the latest code to CodeCommit :

```
voclabs:~/environment/microservices (dev) $ git add .
voclabs:~/environment/microservices (dev) $ git commit -m "Update customer and employee microservices"
[dev d6b372c] Update customer and employee microservices
 2 files changed, 2 insertions(+), 2 deletions(-)
voclabs:~/environment/microservices (dev) $ git push origin dev
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 467 bytes | 467.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Validating objects: 100%
```

2. Navigate to CodeCommit page from AWS Console, then verify the code changes we made are updated. At here, the 8081 is changed to 8080.



Task 6: Create ECR Repositories and Upload Docker Images

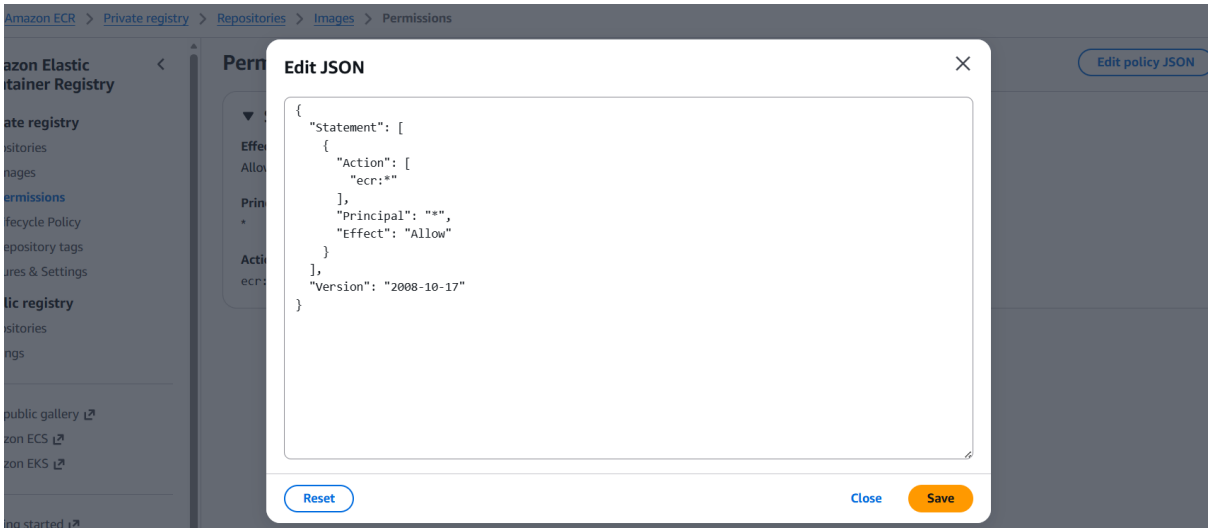
1. Connect and login to ECR service

```
voclabs:~/environment/microservices (dev) $ account_id=$(aws sts
 get-caller-identity |grep Account|cut -d '"' -f4)
voclabs:~/environment/microservices (dev) $ echo $account_id
045285622849
voclabs:~/environment/microservices (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

2. Create 2 private ECR repositories for each microservice we have

```
voclabs:~/environment/microservices (dev) $ aws ecr create-repository --repository
-name customer
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:634943294831:repository/customer",
        "registryId": "634943294831",
        "repositoryName": "customer",
        "repositoryUri": "634943294831.dkr.ecr.us-east-1.amazonaws.com/customer",
        "createdAt": "2025-11-25T10:32:43.854000+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
voclabs:~/environment/microservices (dev) $ aws ecr create-repository --repository-name employee
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:634943294831:repository/employee",
        "registryId": "634943294831",
        "repositoryName": "employee",
        "repositoryUri": "634943294831.dkr.ecr.us-east-1.amazonaws.com/employee",
        "createdAt": "2025-11-25T10:32:57.390000+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
```

3. Navigate to Amazon ECR in AWS console, then click “Repositories” → “customer”→ “Permissions” → “Edit policy JSON”. In the “Edit JSON”, edit it as the picture shown below:



4. Navigate to Amazon ECR in AWS console, then click “Repositories” → “employee”→ “Permissions” → “Edit policy JSON”. In the “Edit JSON”, edit it as the picture shown below:

5. Tag Docker images for pushing to Amazon ECR

```
voclabs:~/environment/microservices (dev) $ account_id=$(aws sts get-caller-identity |grep Account|
cut -d '"' -f4)
voclabs:~/environment/microservices (dev) $
voclabs:~/environment/microservices (dev) $ echo $account_id
634943294831
voclabs:~/environment/microservices (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-
1.amazonaws.com/customer:latest
voclabs:~/environment/microservices (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-
1.amazonaws.com/employee:latest
```

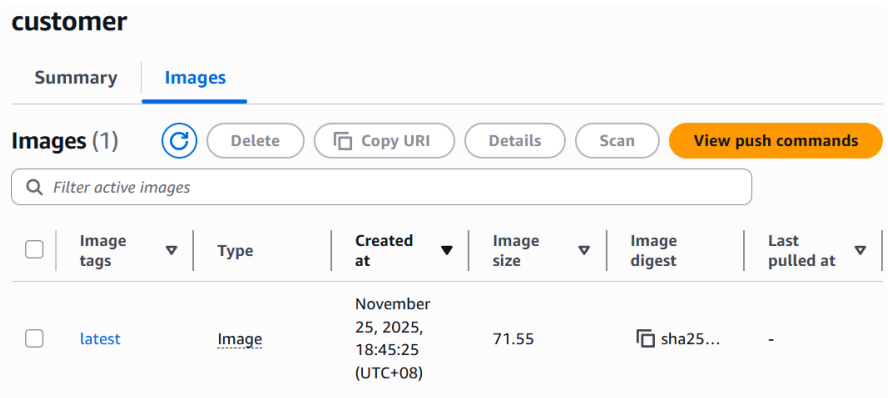
6. Verify the two images exist and tagged correctly

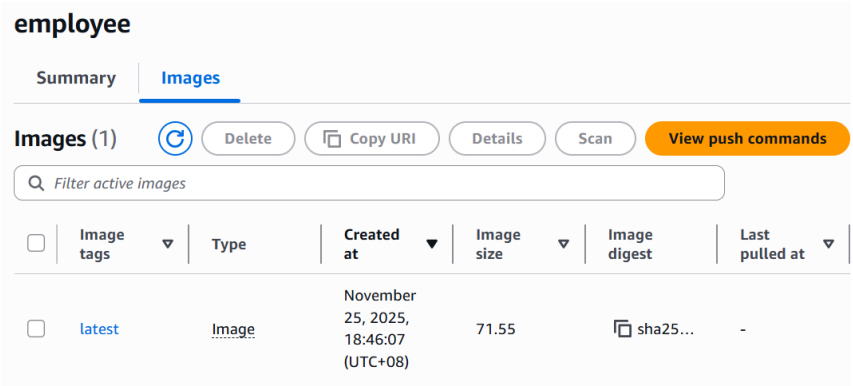
```
voclabs:~/environment/microservices (dev) $ docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
634943294831.dkr.ecr.us-east-1.amazonaws.com/employee     latest    004afdc5b5dd   14 minutes ago   219MB
employee                                                  latest    004afdc5b5dd   14 minutes ago   219MB
<none>                                                    <none>    545aa43668fd   24 minutes ago   219MB
634943294831.dkr.ecr.us-east-1.amazonaws.com/customer     latest    a8fc38ddab4f   46 minutes ago   219MB
customer                                                  latest    a8fc38ddab4f   46 minutes ago   219MB
```

7. Push the two local Docker images to Amazon ECR repository by using the repository and tag specified in the image name.

```
voclabs:~/environment/microservices (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:
latest
The push refers to repository [634943294831.dkr.ecr.us-east-1.amazonaws.com/customer]
ed73780c31d5: Pushed
84aaf00592fa: Pushed
5f70bf18a086: Pushed
54fc9e9e9165: Pushed
2497eefc5264: Pushed
d3b1ea8ff6e6: Pushed
b0f89ac7b966: Pushed
256f393e029f: Pushed
latest: digest: sha256:451ef7f49697d96f035515b269b5945a0b0d3c4b18c314365996bf67d586750f size: 1994
voclabs:~/environment/microservices (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:
latest
The push refers to repository [634943294831.dkr.ecr.us-east-1.amazonaws.com/employee]
be2328d1cea3: Pushed
919ff96fec01: Pushed
5f70bf18a086: Pushed
54fc9e9e9165: Pushed
2497eefc5264: Pushed
d3b1ea8ff6e6: Pushed
b0f89ac7b966: Pushed
256f393e029f: Pushed
latest: digest: sha256:45a5e257a3f01c41b40c742ab135e646ea034d9b05ab4ce9f11dba25683dcd19 size: 1994
```

8. Navigate “Amazon ECR” in AWS Console, then verify the two images are stored in their private repository in ECR.

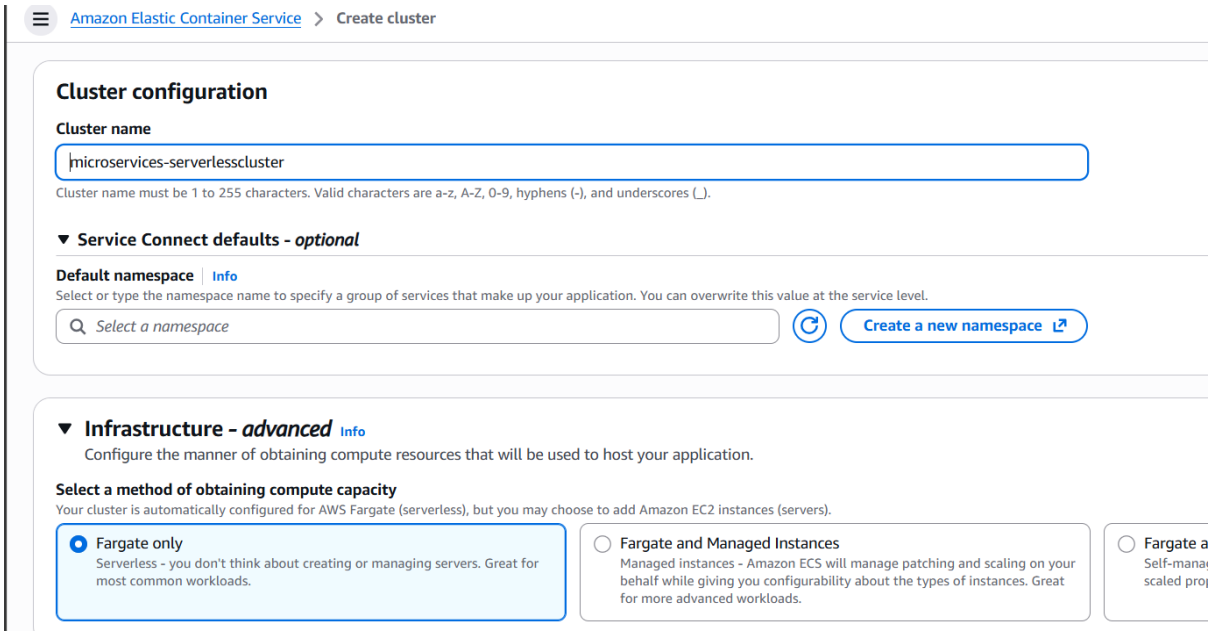



employee
Summary
Images
Images (1)
Delete
Copy URI
Details
Scan
View push commands
Filter active images
Image tags
Type
Created at
Image size
Image digest
Last pulled at
latest
Image
November 25, 2025, 18:46:07 (UTC+08)
71.55
sha25...
-

Phase 3: Creating ECS Cluster, Task Definitions, and AppSpec Files

Task 1: Create ECS Cluster

1. Create a serverless AWS Fargate cluster called **microservices-serverlesscluster**:



Task 2: Create another CodeCommit Repository to Store Deployment Files

1. Create new repository in CodeCommit, named "**deployment**".
2. Create new directory called "**deployment**" in Cloud 9.

```
voclabs:~/environment/microservices (dev) $ aws codecommit create-repository --repository-name deployment --repository-description "Repo for deployment"
{
    "repositoryMetadata": {
        "accountId": "634943294831",
        "repositoryId": "ade2c780-fee8-4e54-8834-75490469eb5a",
        "repositoryName": "deployment",
        "repositoryDescription": "Repo for deployment",
        "lastModifiedDate": "2025-11-25T10:59:51.158000+00:00",
        "creationDate": "2025-11-25T10:59:51.158000+00:00",
        "cloneUrlHttp": "https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment",
        "cloneUrlSsh": "ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment",
        "Arn": "arn:aws:codecommit:us-east-1:634943294831:deployment",
        "kmsKeyId": "arn:aws:kms:us-east-1:634943294831:key/59176070-5634-4e2f-a8b0-248417384001"
    }
}
```

```
voclabs:~/environment/microservices (dev) $ mkdir ~/environment/deployment
voclabs:~/environment/microservices (dev) $ cd ~/environment/deployment
voclabs:~/environment/deployment $ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
voclabs:~/environment/deployment (master) $ git branch -m dev
```

Task 3: Task Definitions for Customer Microservice

1. Create **taskdef-customer.json** file under the "deployment" directory. Then, write the JSON code as shown below:

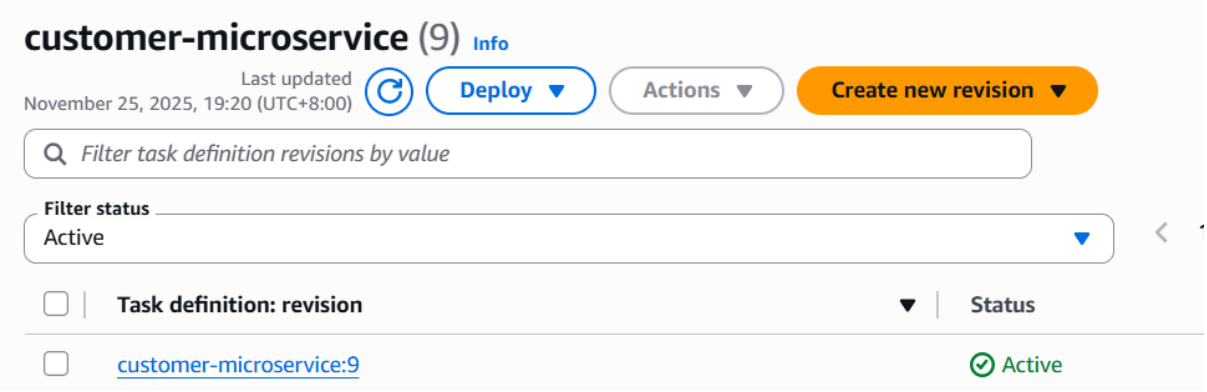

```
{
    "containerDefinitions": [
        {
            "name": "customer",
            "image": "customer",
            "environment": [
                {
                    "name": "APP_DB_HOST",
                    "value": "http://10.16.10.66:8000"
                }
            ],
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 8080,
                    "protocol": "tcp",
                    "containerPort": 8080
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-capstone",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "awslogs-capstone"
                }
            }
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "512",
    "memory": "1024",
    "executionRoleArn": "arn:aws:iam::634943294831:role/PipelineRole",
    "family": "customer-microservice"
}
```

The task definition configures the ECS container by specifying its image, environment variables, port mappings, and log settings. It defines how the container should run on Fargate, including CPU, memory, network mode, and the IAM role it uses. This JSON essentially tells ECS how to launch and manage the "customer" microservice.

2. Register the ECS task definition using the JSON configuration file we just created.

```
voclabs:~/environment/deployment (dev) $ aws ecs register-task-definition --cli-input-
json "file:///home/ec2-user/environment/deployment/taskdef-customer.json"
{
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:634943294831:task-definition/custo
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:634943294831:task-definition/custo
mer-microservice:9",
        "containerDefinitions": [
            {
                "name": "customer",
                "image": "customer",
                "cpu": 0,
                "portMappings": [
                    {
                        "containerPort": 8080,
                        "hostPort": 8080,
                        "protocol": "tcp"
                    }
                ]
```

3. Navigate to the “Amazon Elastic Container Service” in AWS Console, then go to “Task definitions” and verify the task definition for customer microservice is created.



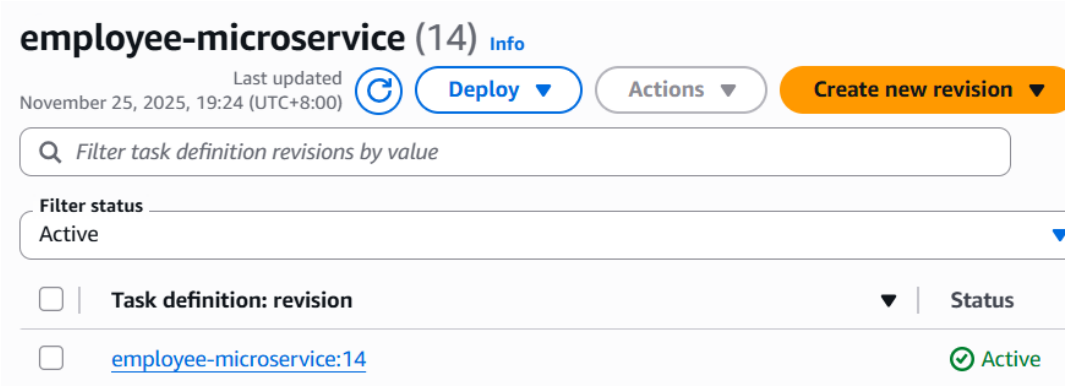
Task 4: Task Definitions for Employee Microservice

1. Create **taskdef-employee.json** file under the deployment directory. Then, write the JSON code as shown below:

2. Register the ECS task definition using the JSON configuration file we just created.

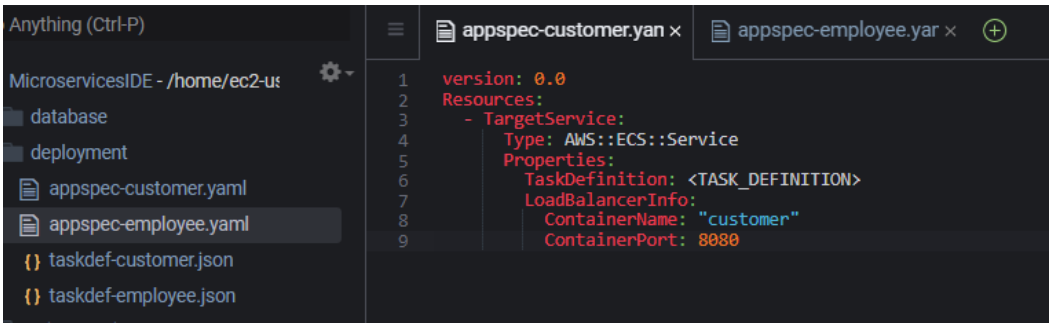
```
voclabs:~/environment/deployment (dev) $ aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-employee.json"
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:634943294831:task-definition/employee-microservice:14",
        "containerDefinitions": [
            {
                "name": "employee",
                "image": "employee",
                "cpu": 0,
                "portMappings": [
                    {
                        "containerPort": 8080,
                        "hostPort": 8080,
                        "protocol": "tcp"
!
```

3. Navigate to the “Amazon Elastic Container Service” in AWS Console, then go to “Task definitions” and verify the task definition for employee microservice is created.

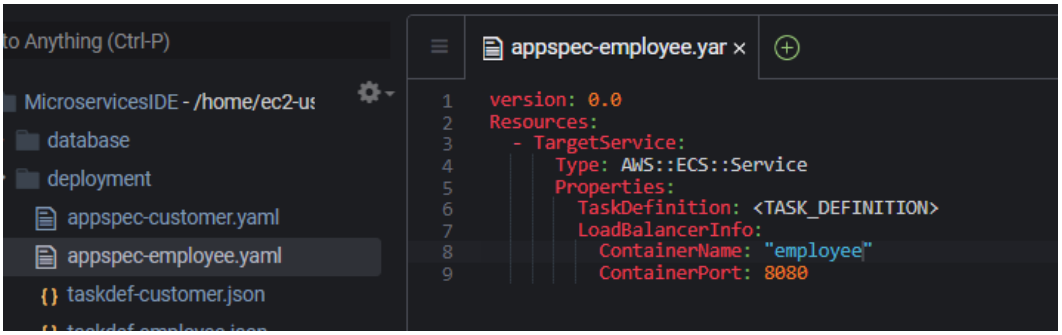


Task 5: Create AppSpec Files for CodeDeploy

1. Create new file called “appspec-customer.yaml” under “deployment” directory, then write the code into the file as shown below:

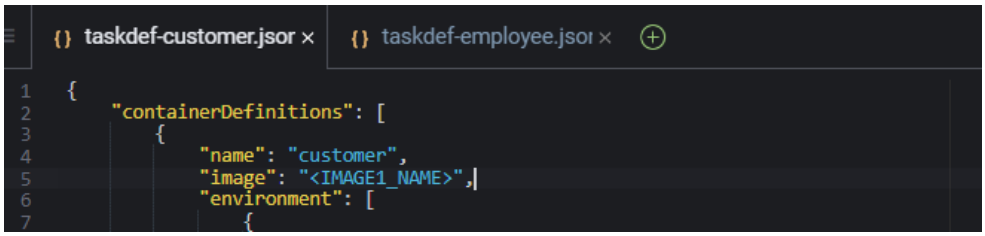


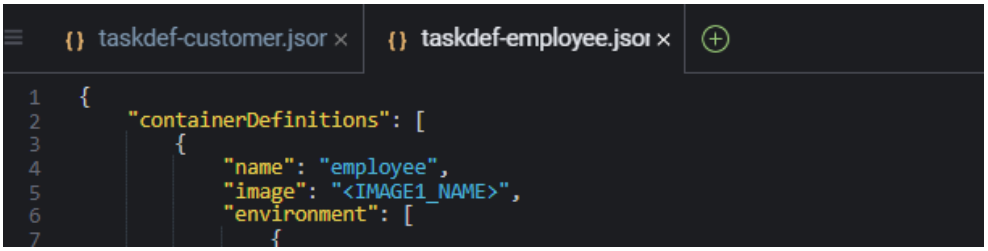
2. Create new file called “appspec-employee.yaml” under “deployment” directory, then write the code into the file as shown below:



Task 6: Update the Two "Task Definition Files" and Push into Deployment Repository

1. Change the line 5 value into a < IMAGE1_NAME> placeholder



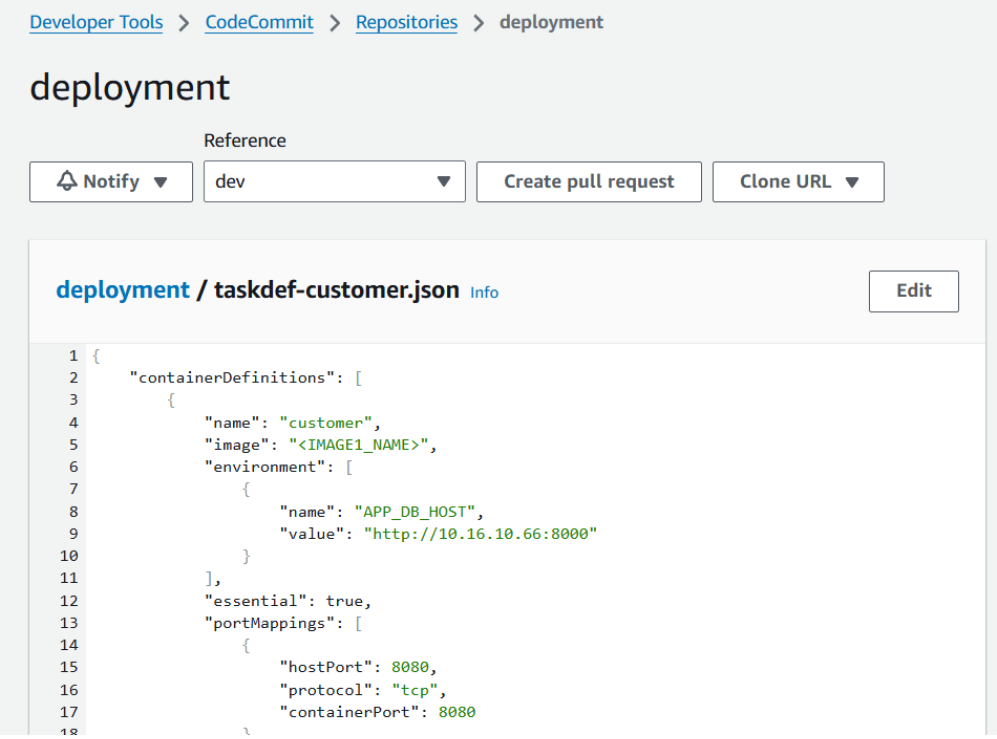


2. Connects the **local development repository** to the remote CodeCommit repository called "origin". Then, stage all changes and create commit, and then push the **local dev branch** to the **remote origin branch** in CodeCommit.

```
voclabs:~/environment/deployment (dev) $ git remote add origin https:
//git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
voclabs:~/environment/deployment (dev) $ git remote -v
origin  https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment (fetch)
origin  https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment (push)
```

```
voclabs:~/environment/deployment (dev) $ git add .
voclabs:~/environment/deployment (dev) $ git commit -m "Added task def and appspec fi;es"
[dev (root-commit) e0b680f] Added task def and appspec fi;es
 4 files changed, 100 insertions(+)
 create mode 100644 appspec-customer.yaml
 create mode 100644 appspec-employee.yaml
 create mode 100644 taskdef-customer.json
 create mode 100644 taskdef-employee.json
voclabs:~/environment/deployment (dev) $ git push -u origin dev
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.05 KiB | 1.05 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
 * [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
```

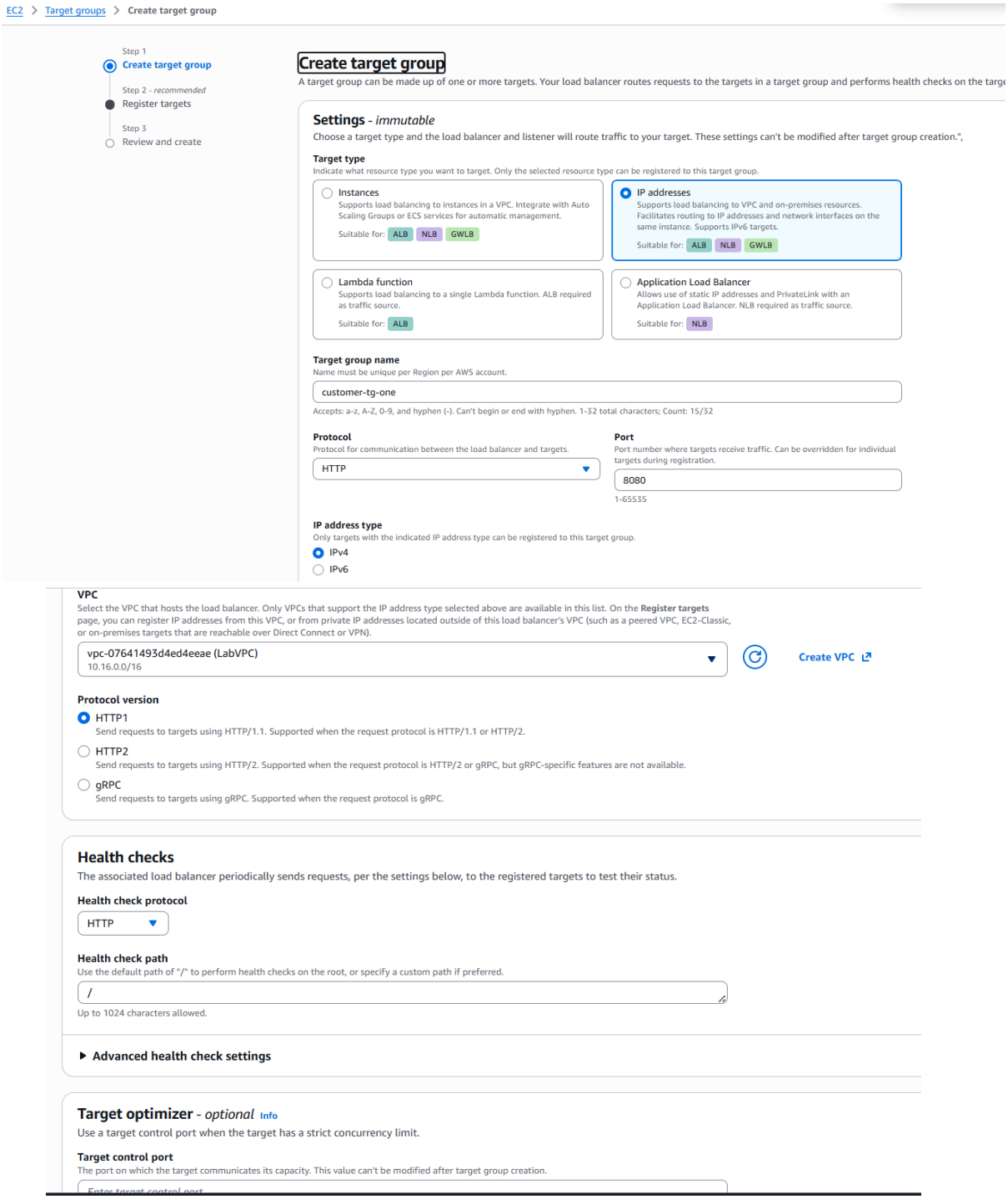
3. Verify it in AWS Console

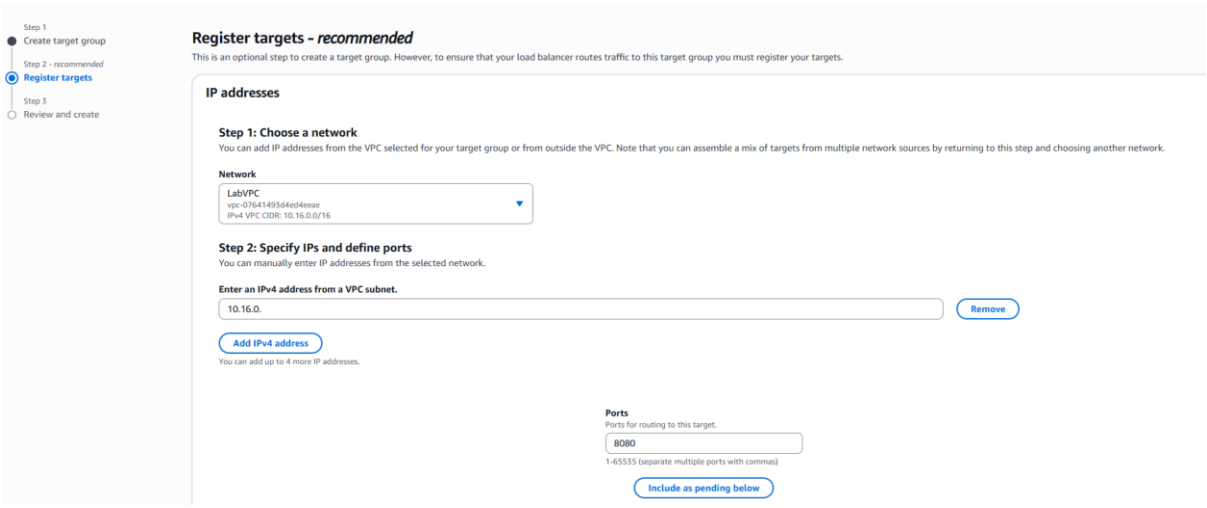


Phase 4: Creating Target Groups and Application Load Balancer

Task 1: Create the first target group for customer microservice

1. Navigate to AWS Console → EC2 → Target Groups → “Choose target group”, then create first target group “customer-tg-one” for customer as shown below :





Task 2: Create the second target group for customer microservice

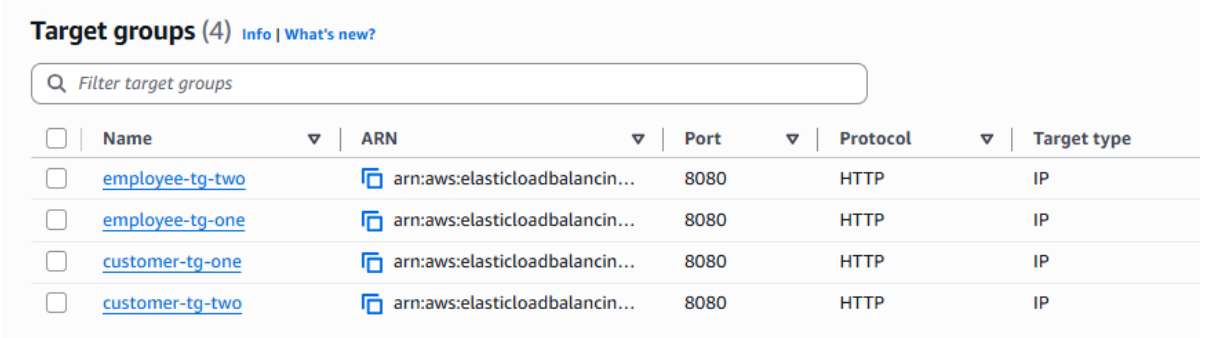
1. Repeat same step form task 1, only change the target group name to “customer-tg-two”

Task 3: Create the first target group for employee microservice

1. Repeat same step form task 1, only change the target group name to “employee-tg-one” and the heath check path to “/admin/”

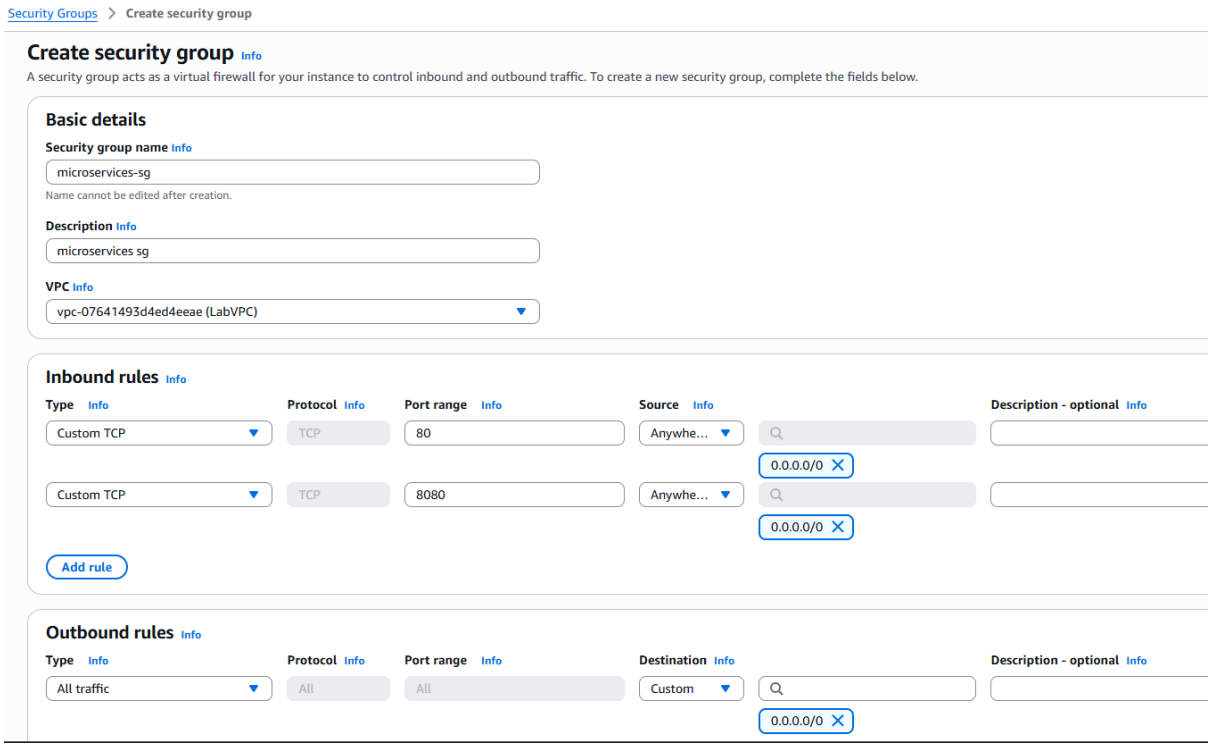
Task 4: Create the second target group for employee microservice

1. Repeat same step form task 1, only change the target group name to “employee-tg-two” and the heath check path to “/admin/”
2. After that, verify all 4 target groups are created:

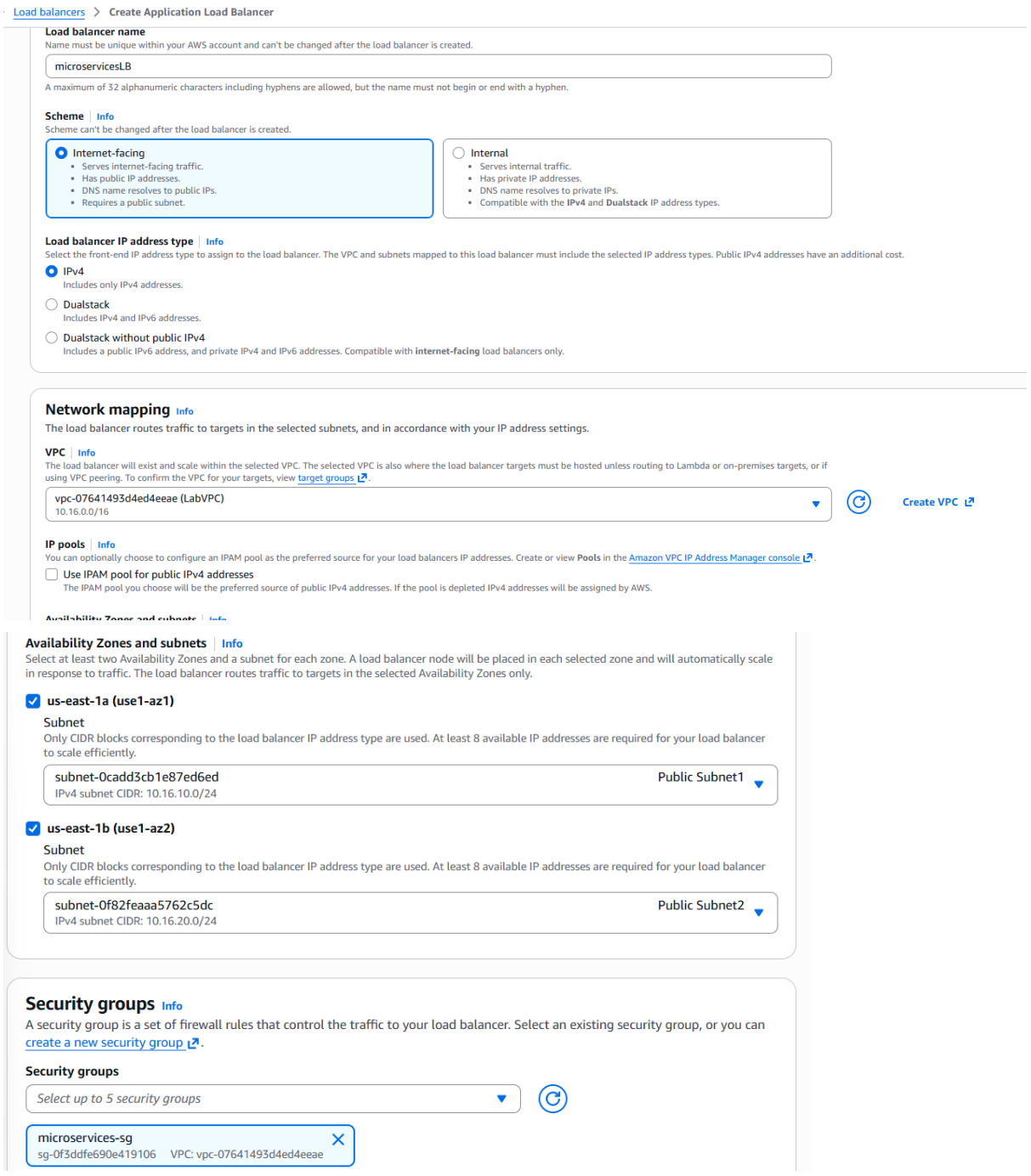


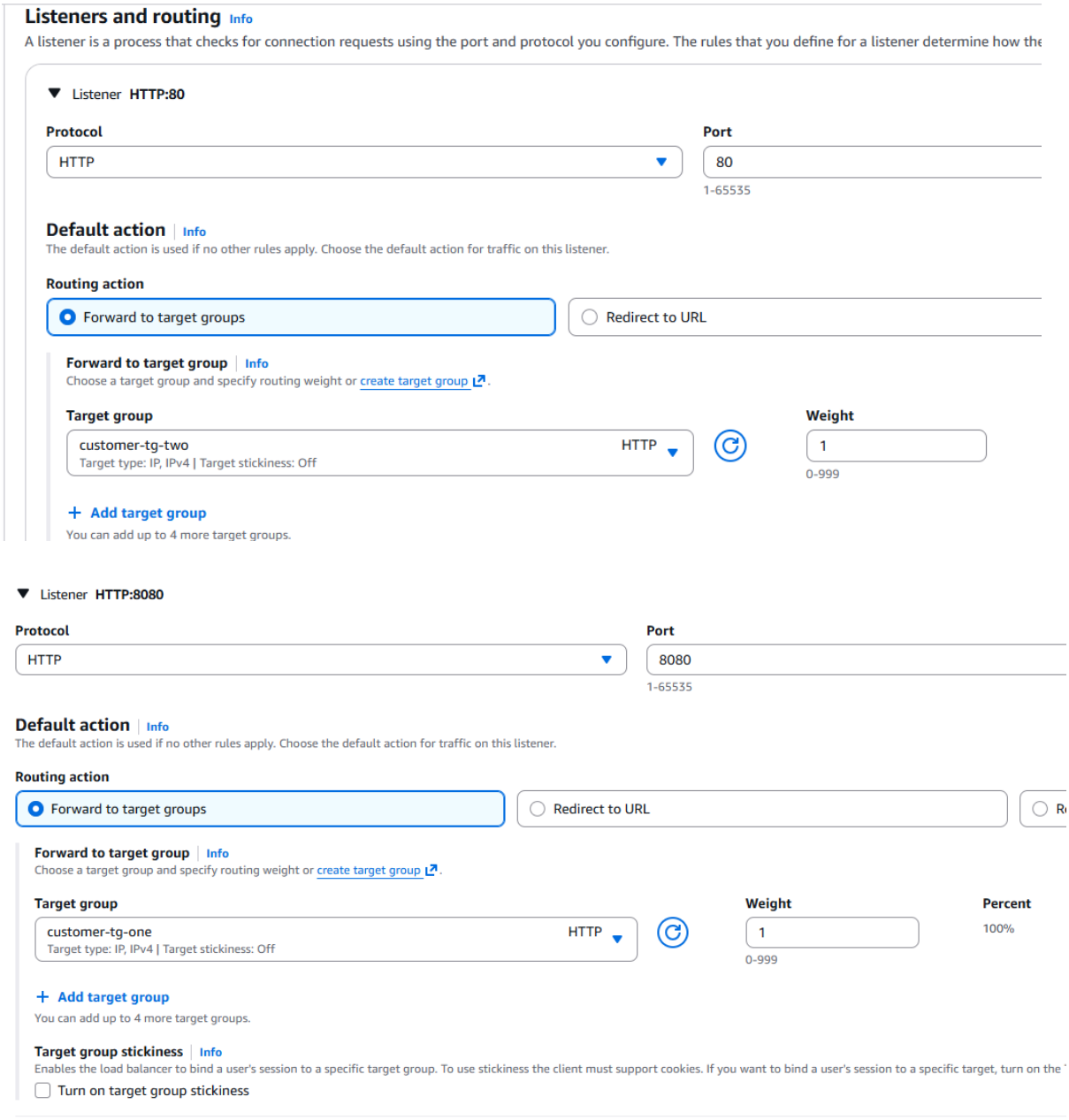
Task 5: Create Application Load Balancer

1. Navigate to AWS Console → EC2 → Security Groups → Create security group, then create a new EC2 security group called “microservices-sg” as shown below:



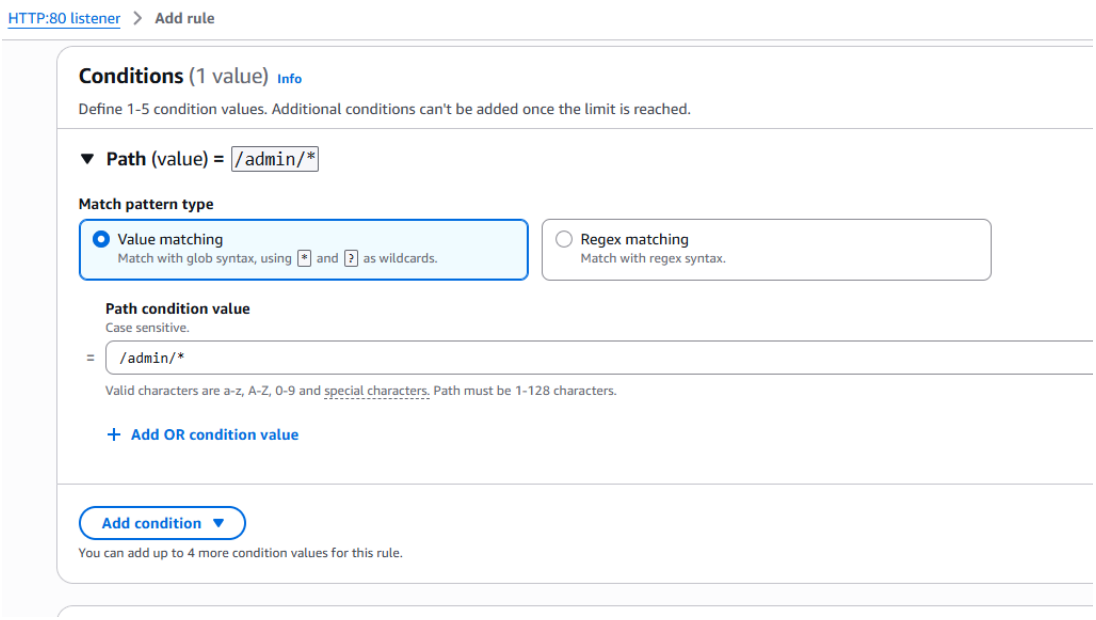
2. Navigate to AWS Console → EC2 → Load Balancers → Create Application Load Balancer, then configure as shown below:

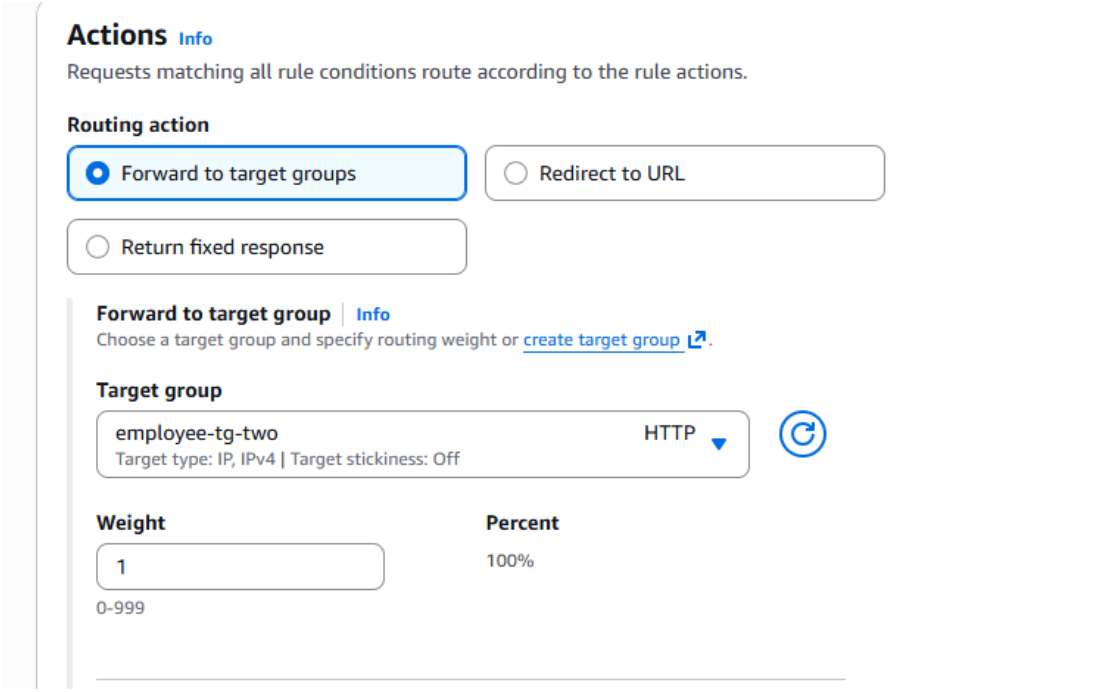




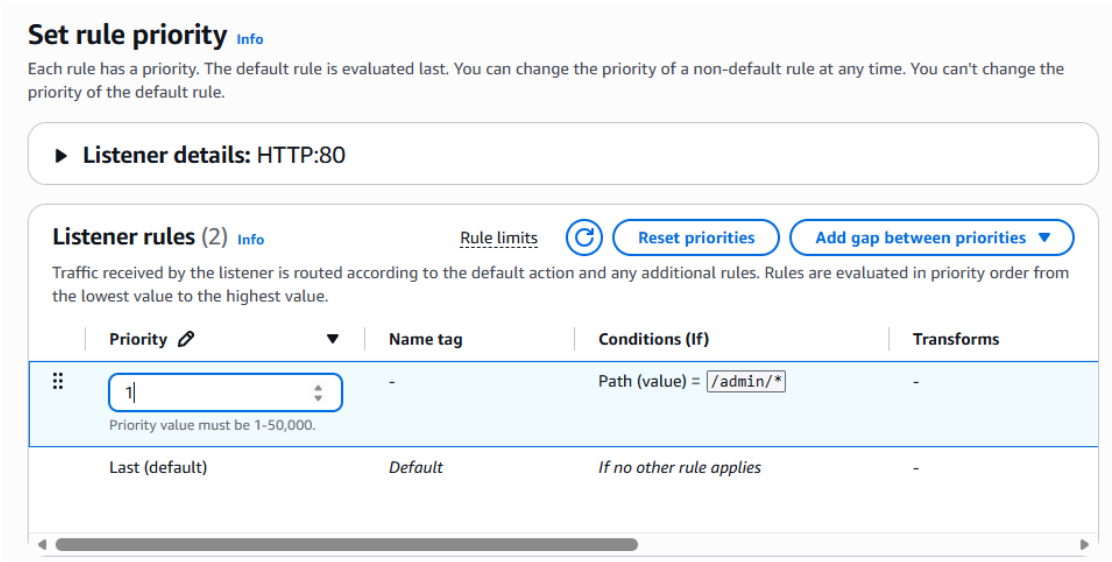
Two listeners were configured on the load balancer: the first listened on HTTP **port 80** and forwarded traffic to **customer-tg-two** by default, while the second listened on HTTP **port 8080** and forwarded traffic to **customer-tg-one** by default.

3. Navigate to HTTP:80 listener from the microservicesLB we just created. Then add rules as shown below:

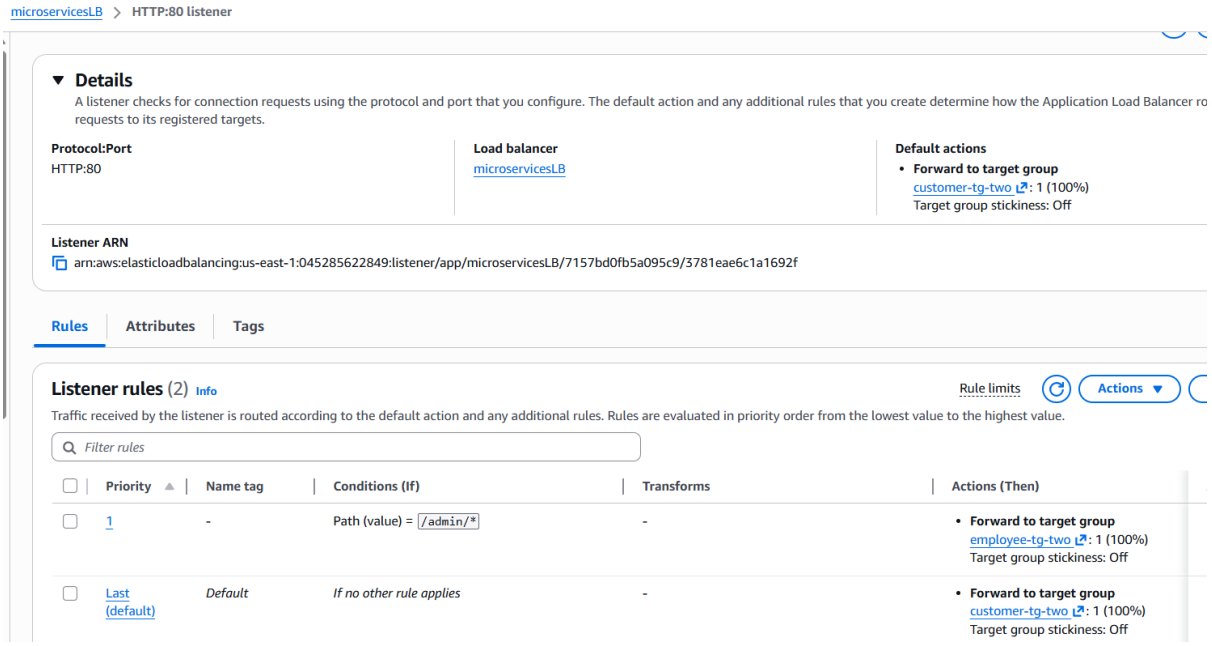




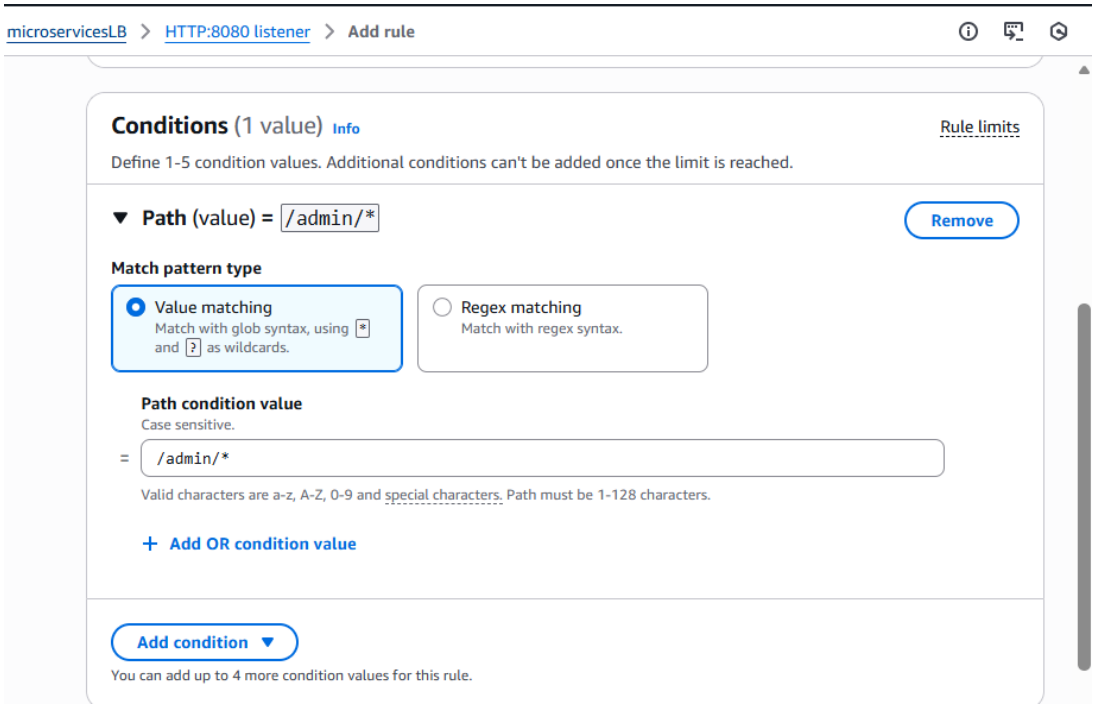
For the HTTP:80 listener, default traffic was forwarded to customer-tg-two and if path is **/admin/*** requests were forwarded to **employee-tg-two**

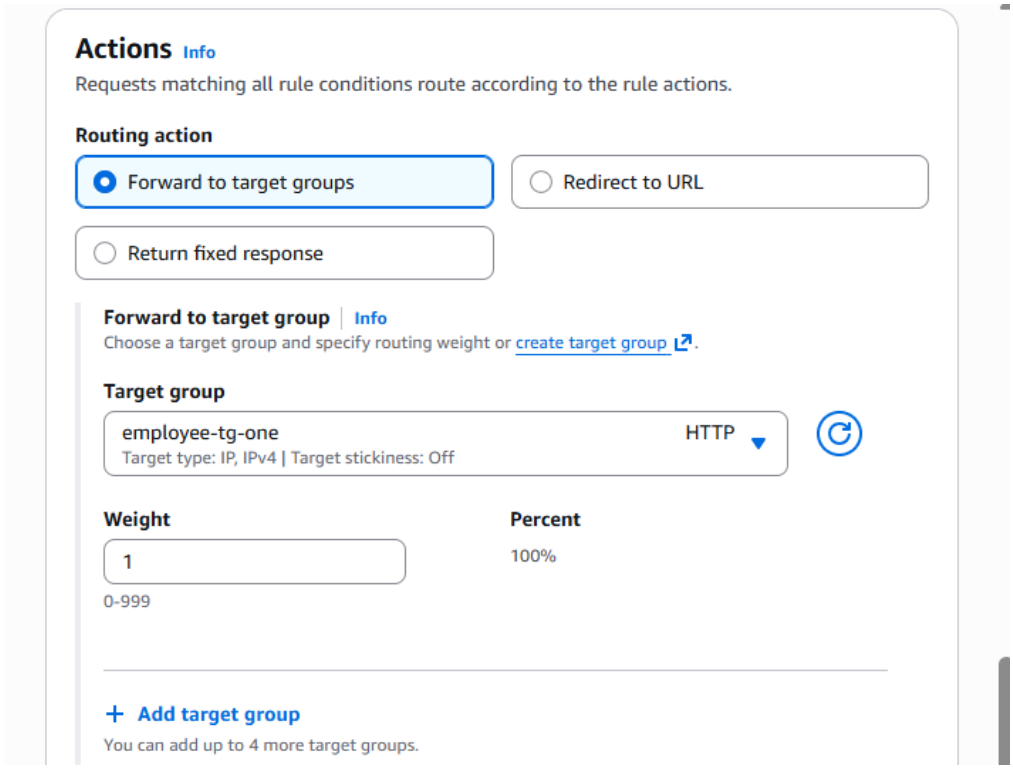


4. Verify the rule is added to 80 listener:

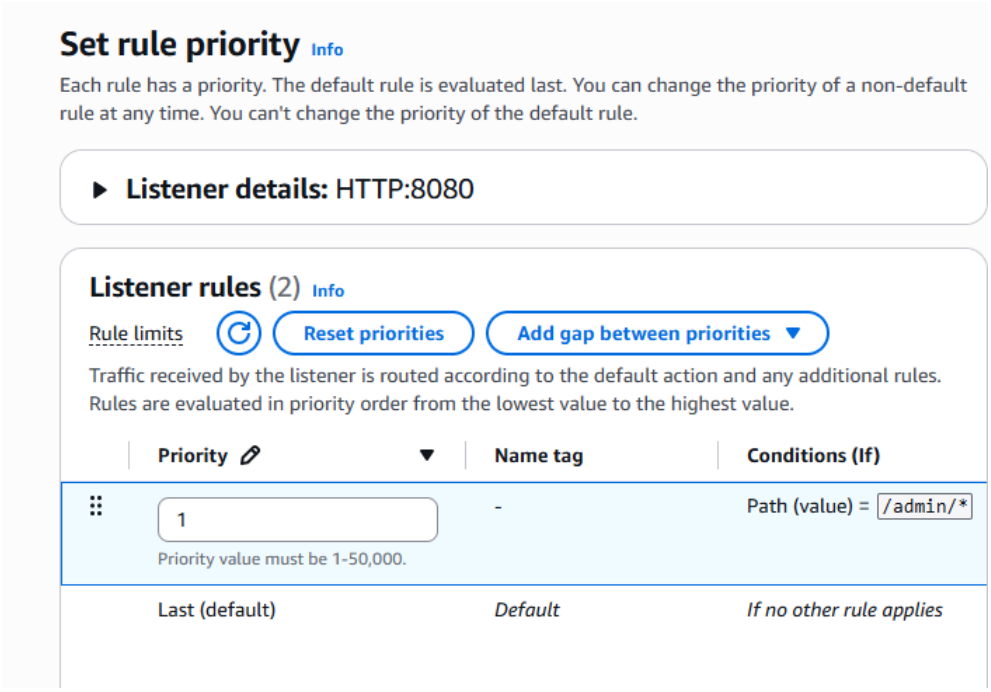


5. Navigate to HTTP:8080 listener from the microservicesLB we just created. Then add rules as shown below:

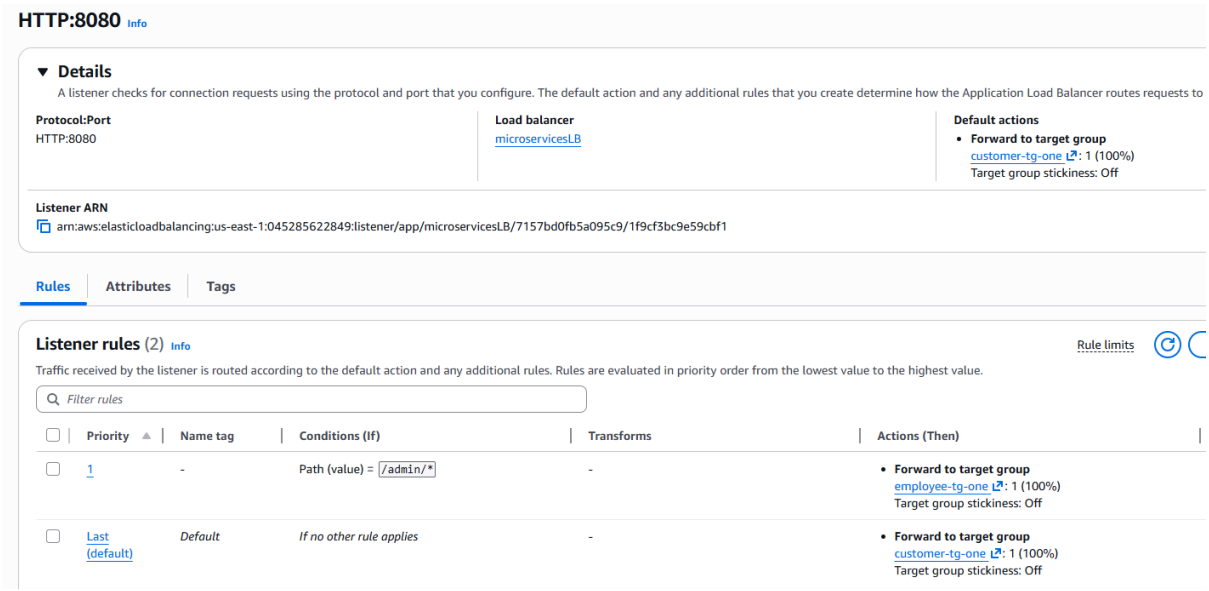




For the HTTP:8080 listener, default traffic was forwarded to customer-tg-one and if path is **/admin/*** requests were forwarded to **employee-tg-one.**



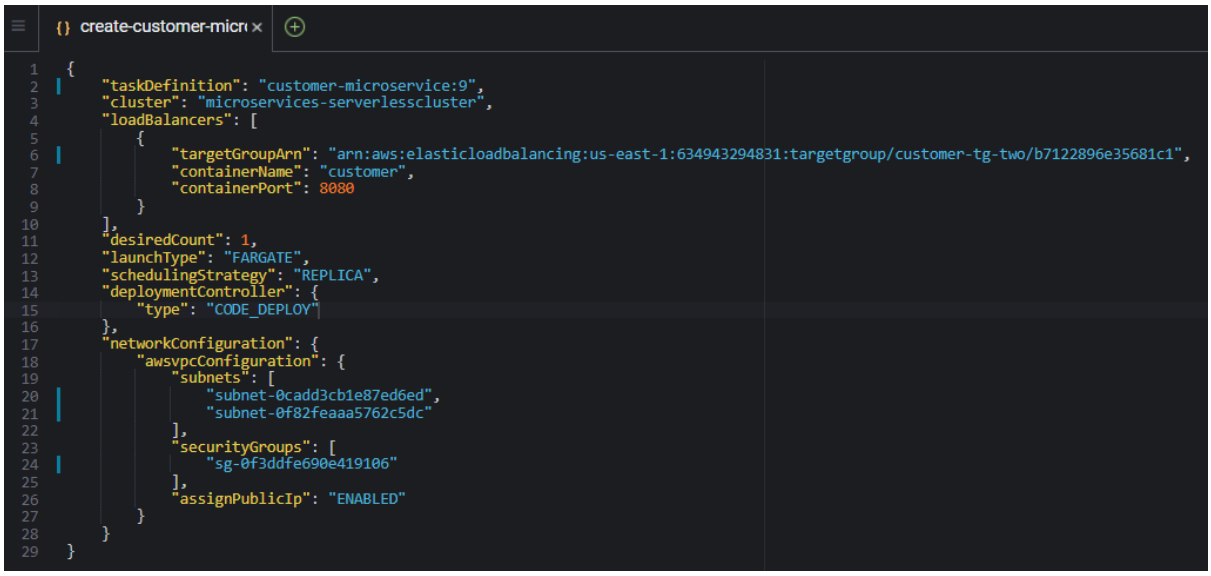
6. Verify the rule is added to 8080 listener:



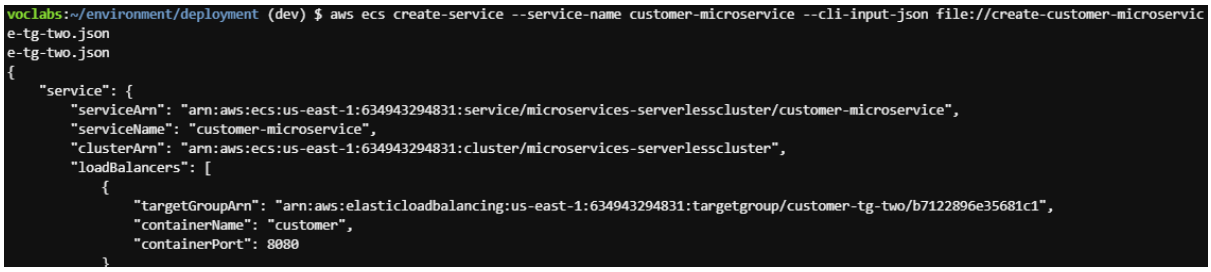
Phase 5: Creating two Amazon ECS services

Task 1: Create ECS Service for Customer Microservice

1. In Cloud 9, create new file called "**create-customer-microservice-tg-two.json**" under the deployment directory.

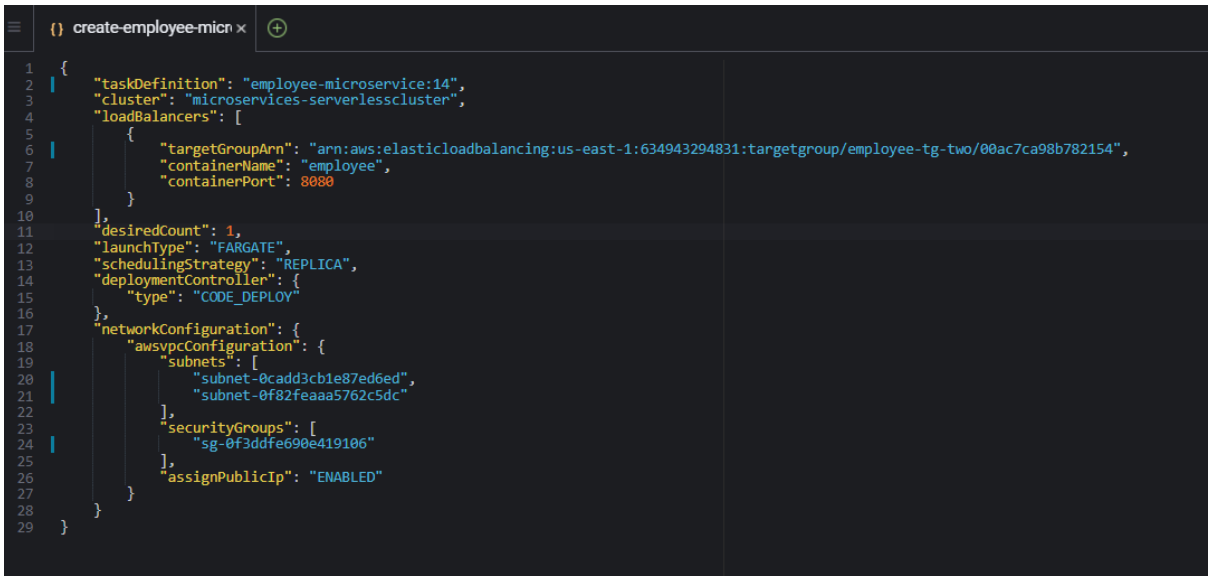


2. Create the ECS service by running this command: aws ecs create-service --service-name customer-microservice --cli-input-json file://create-customer-microservice-tg-two.json

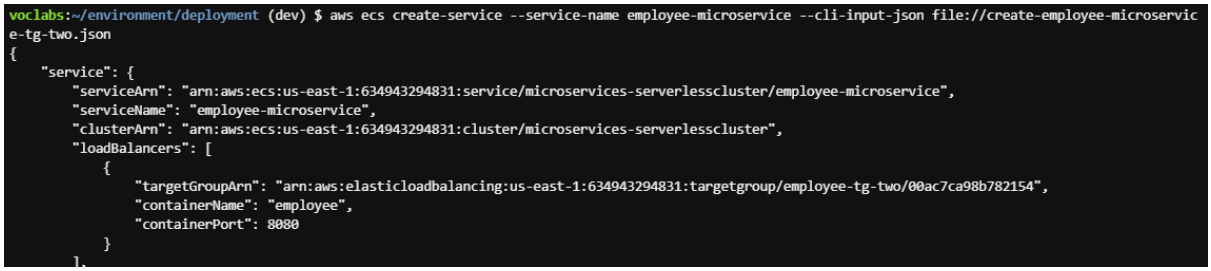


Task 2: Create ECS Service for Employee Microservice

1. In Cloud 9, create new file called "**create-employee-microservice-tg-two.json**" under the deployment directory.



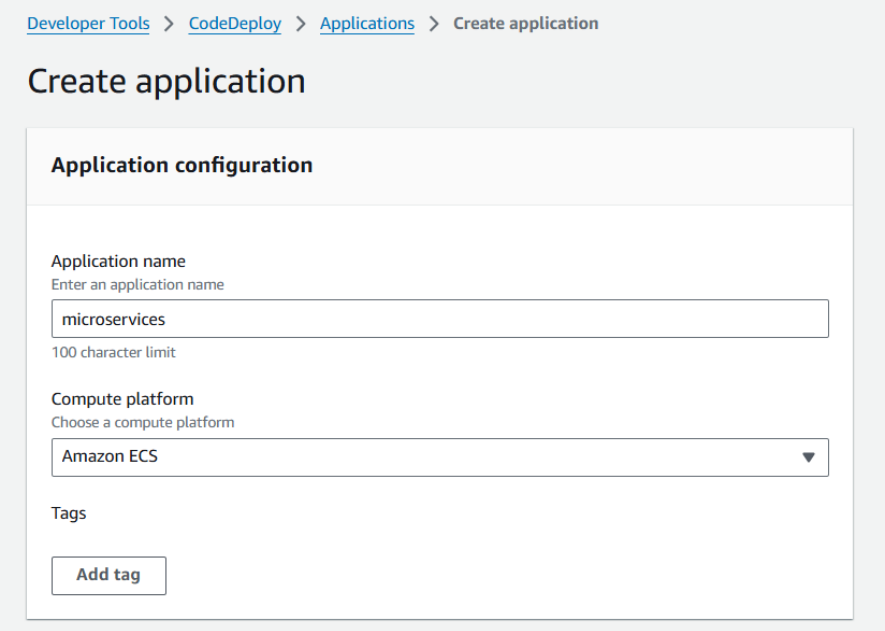
2. Create the ECS service by running this command: aws ecs create-service --service-name employee-microservice --cli-input-json file://create-employee-microservice-tg-two.json.



Phase 6: Configuring CodeDeploy and CodePipeline

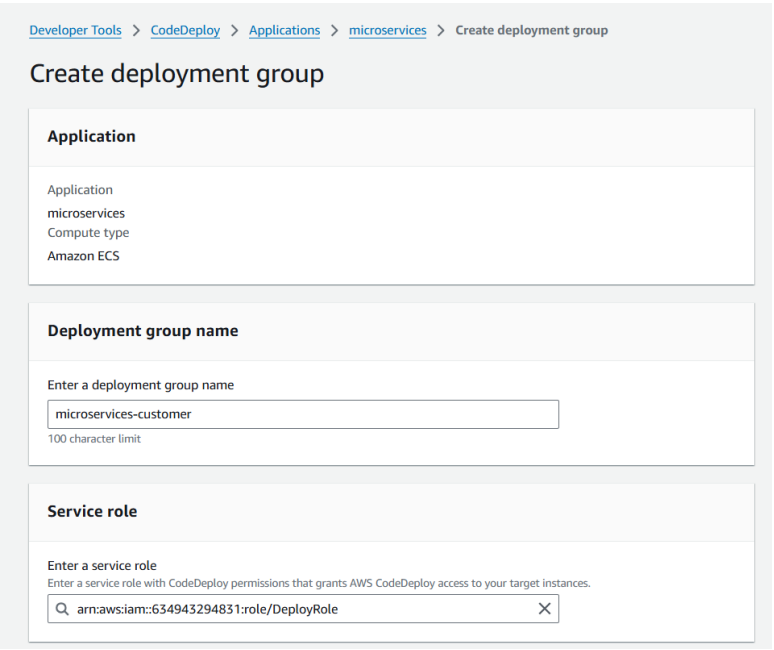
Task 1: Create a CodeDeploy application

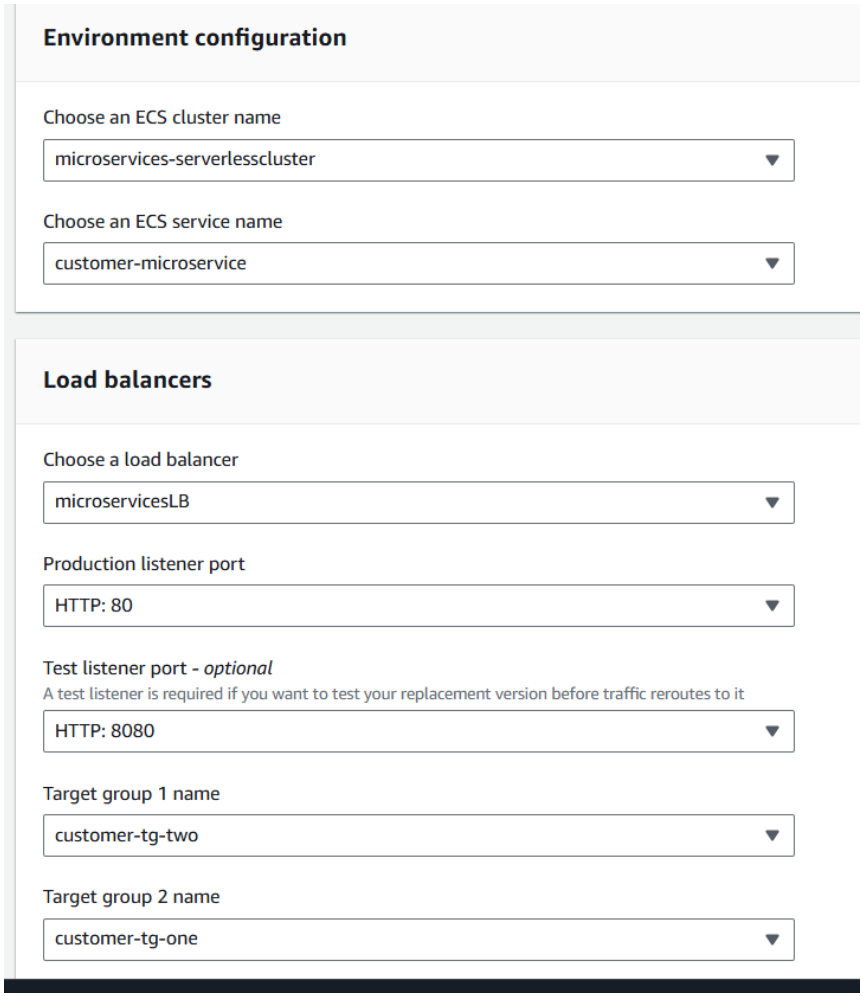
1. Create a CodeDeploy application called "**microservices**" with Amazon ECS as its compute platform

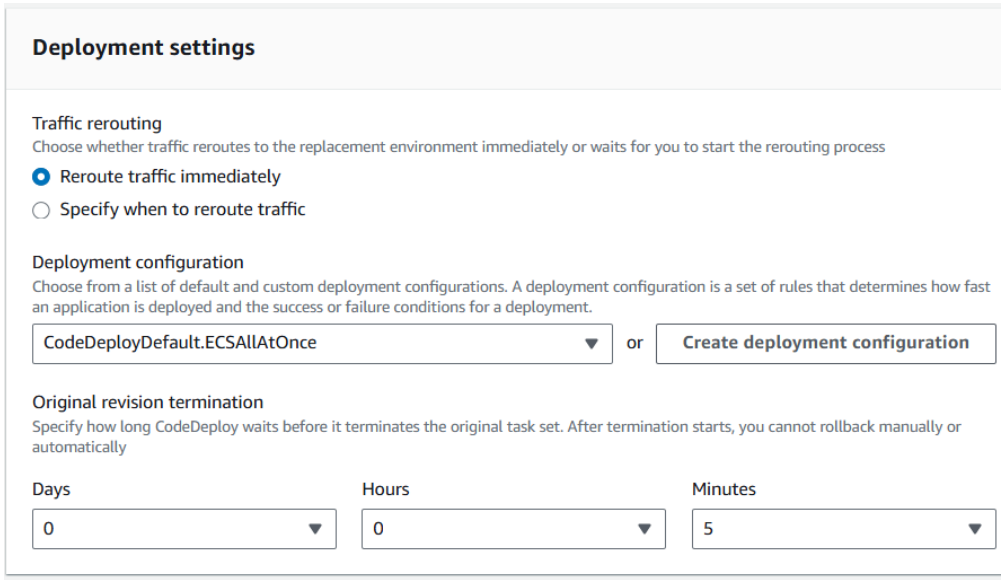


Task 2: Create a CodeDeploy Deployment Group for the Customer Microservice

1. Navigate to "Developer Tools" →CodeDeploy → Applications → microservices, then click "Create deployment group". After that, configure as shown below:

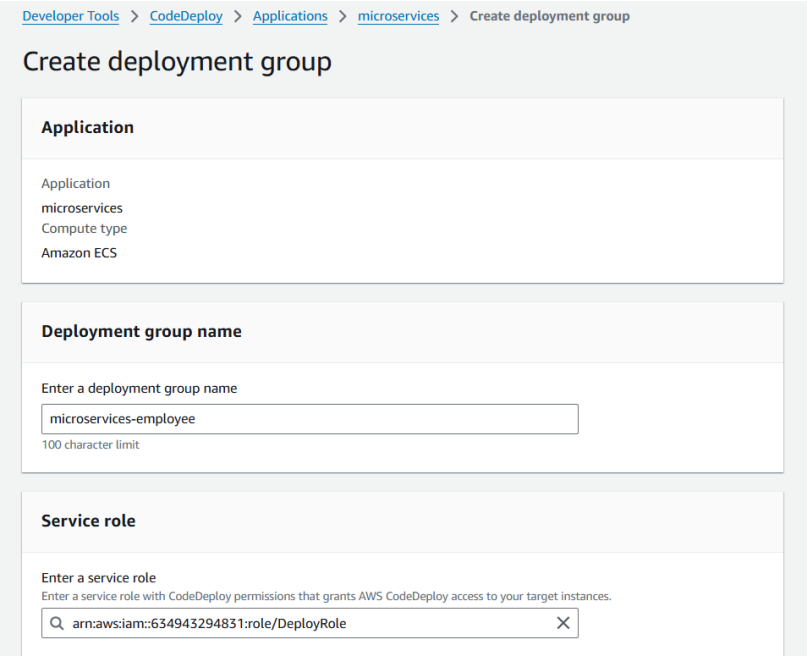


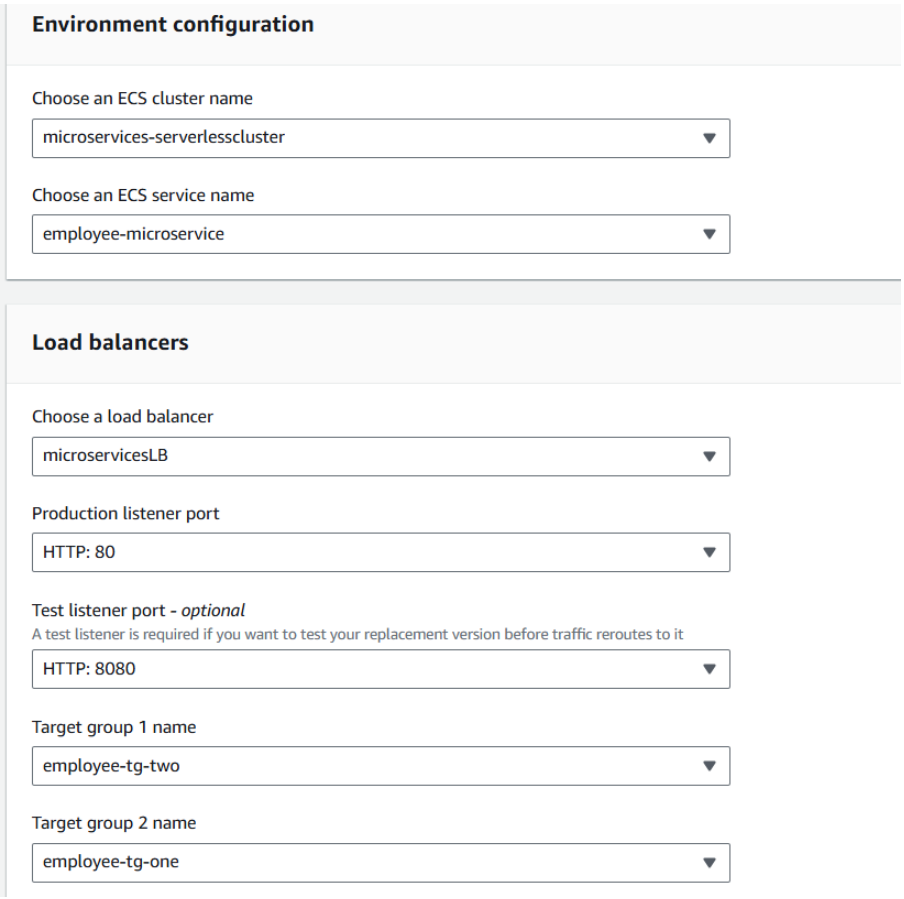

Environment configuration
Choose an ECS cluster name
microservices-serverlesscluster
Choose an ECS service name
customer-microservice
Load balancers
Choose a load balancer
microservicesLB
Production listener port
HTTP: 80
Test listener port - optional
A test listener is required if you want to test your replacement version before traffic reroutes to it
HTTP: 8080
Target group 1 name
customer-tg-two
Target group 2 name
customer-tg-one

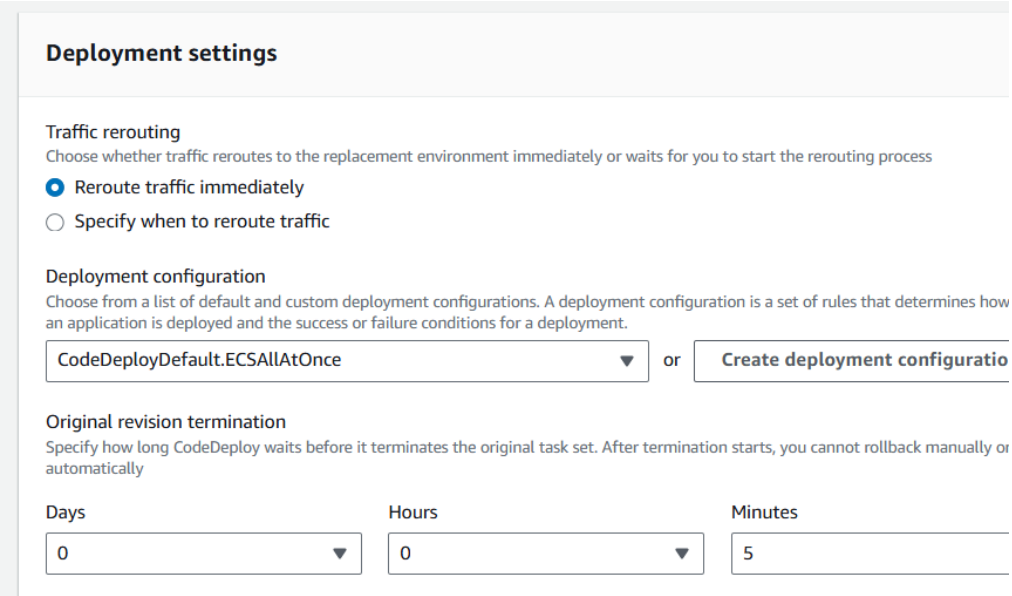

Deployment settings
Traffic rerouting
Choose whether traffic reroutes to the replacement environment immediately or waits for you to start the rerouting process
Reroute traffic immediately
Specify when to reroute traffic
Deployment configuration
Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application is deployed and the success or failure conditions for a deployment.
CodeDeployDefault.ECSAllAtOnce
or
Create deployment configuration
Original revision termination
Specify how long CodeDeploy waits before it terminates the original task set. After termination starts, you cannot rollback manually or automatically
Days
0
Hours
0
Minutes
5

Task 3: Create a CodeDeploy Deployment Group for the Employee Microservice

1. Navigate to “Developer Tools” →CodeDeploy → Applications → microservices, then click “Create deployment group”. After that, configure as shown below:

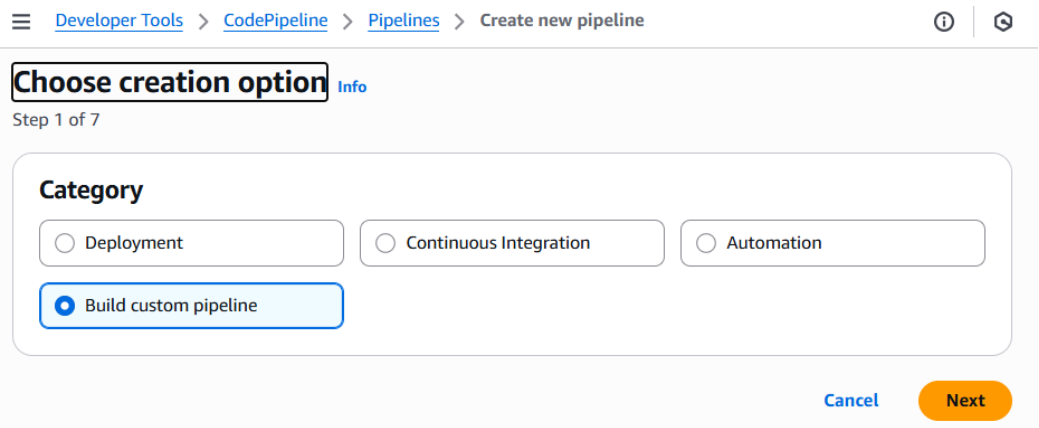


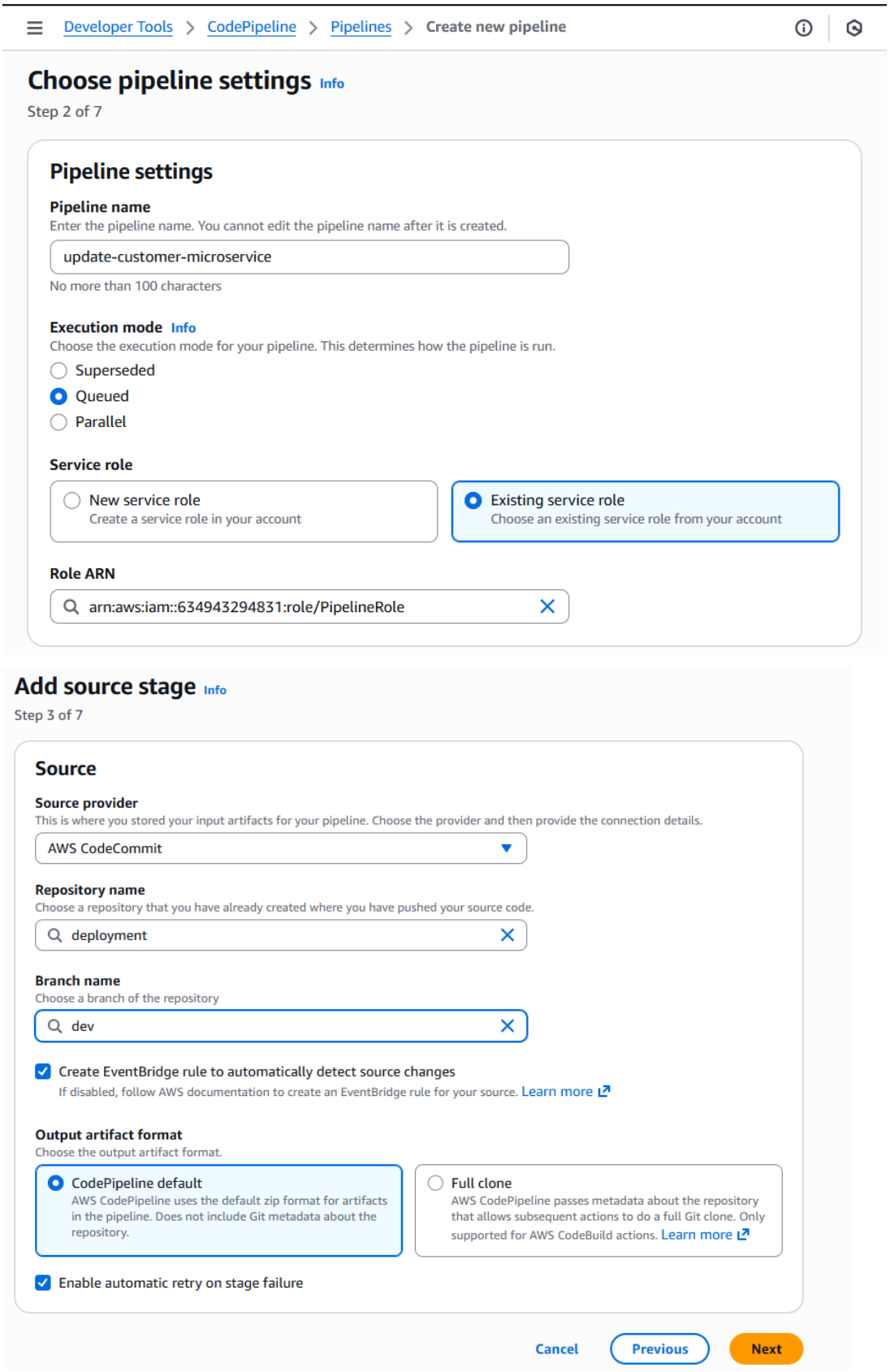




Task 4: Create Pipeline for Customer Microservice

1. Navigate to “Developer Tools” → CodePipeline → Pipelines → Create new pipeline. At here, create a pipeline called “update-customer-microservice”. Configure as shown below:





Skip step 4 “build stage” and step 5 “add test stage”.

Add deploy stage Info

Step 6 of 7

You cannot skip this stage
Pipelines must have at least two stages. Your second stage must be either a build, test or deployment stage.
Choose a provider for either the build stage, test stage or deployment stage.

Deploy

Deploy provider
Choose how you want to deploy your application or content. Choose the provider, and then provide the configuration details for that provider.

Amazon ECS (Blue/Green)

Region

United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Defined by: Source

No more than 100 characters

AWS CodeDeploy application name
Choose one of your existing applications, or create a new one in AWS CodeDeploy.

microservices | Create application

AWS CodeDeploy deployment group
Choose one of your existing deployment groups, or create a new one in AWS CodeDeploy.

mircroservices-customer

Amazon ECS task definition
Choose the input artifact where your Amazon ECS task definition file is stored. If other than the default file path, specify the path and filename of your task definition file.

SourceArtifact | taskdef-customer.json

The default path is taskdef.json.

AWS CodeDeploy AppSpec file
Choose the input artifact where your AWS CodeDeploy AppSpec file is stored. If other than the default file path, specify the path and filename of your AppSpec file.

SourceArtifact | appspec-customer.yaml

Dynamically update task definition image - *optional*
You can provide an input artifact and a placeholder name for the container definition image that will be used to dynamically update a task definition. You can specify multiple input artifacts and placeholders.

Input artifact with image details

Select input artifact

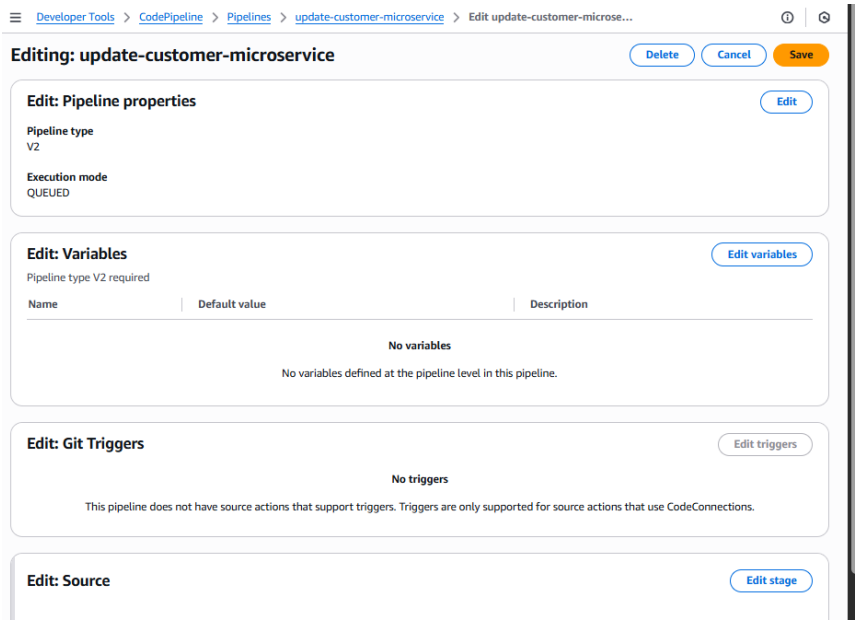
Placeholder text in the task definition

IMAGE

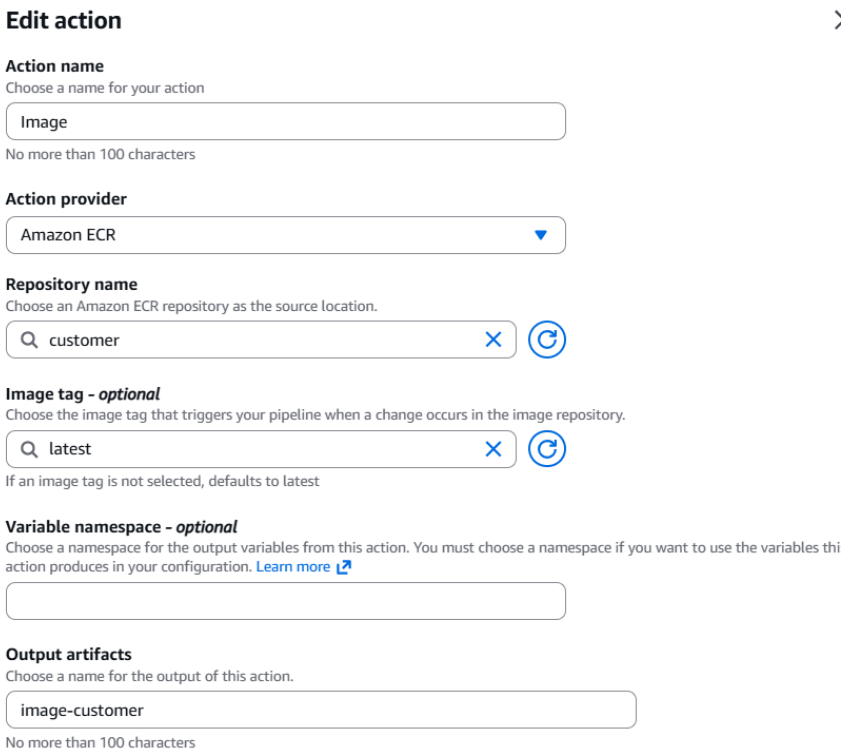
☑ Configure automatic rollback on stage failure

☐ Enable automatic retry on stage failure

2. Navigate to Developer Tools → CodePipeline → Pipelines → update-customer-microservice → Edit

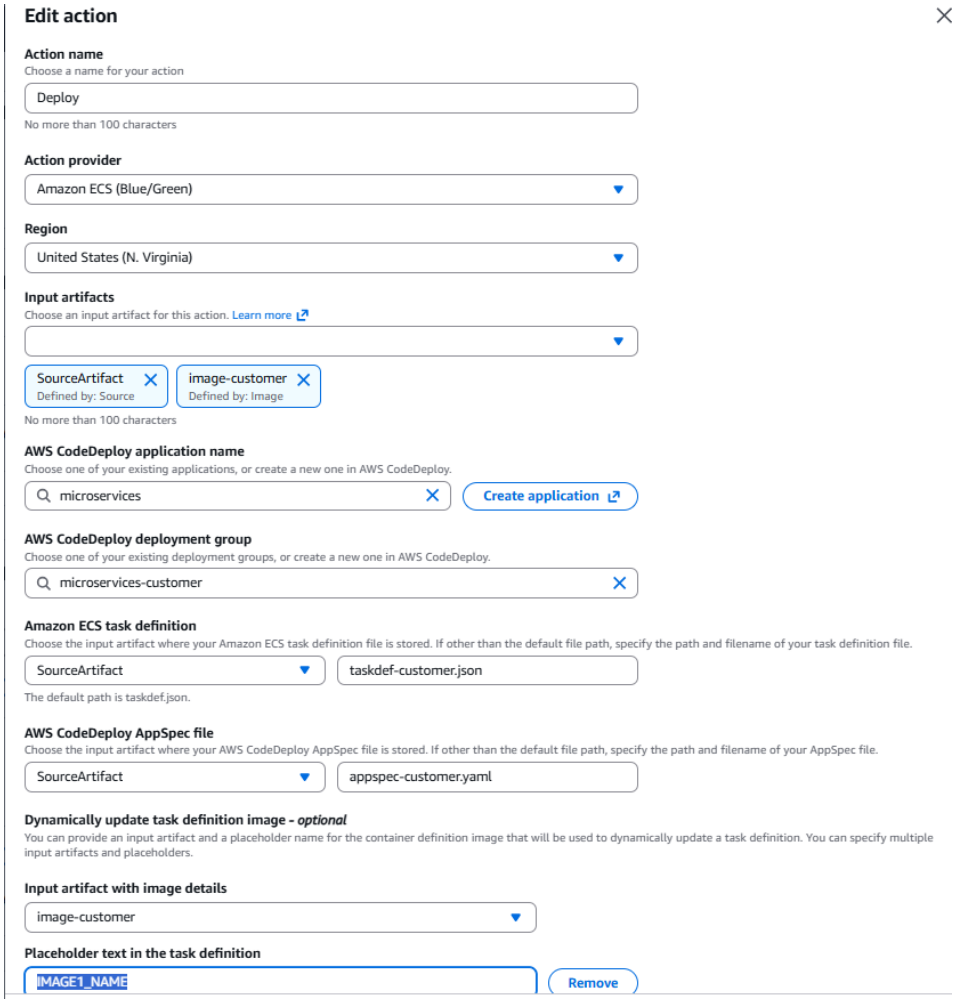


Then, update the Source section by adding a new action as shown below:



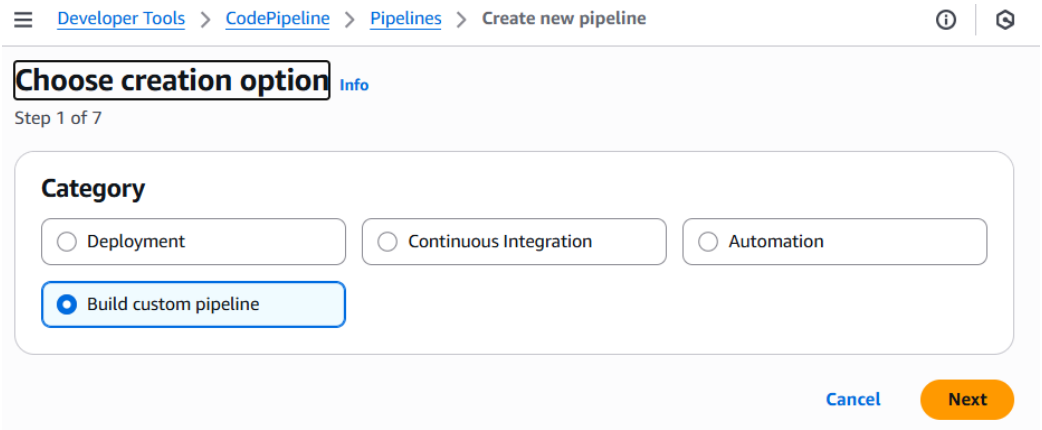
- This means that not only pushing updated source code to deployment/dev branch in CodeCommit will automatically trigger the pipeline, but also pushing a new image to ECR can trigger it as well.

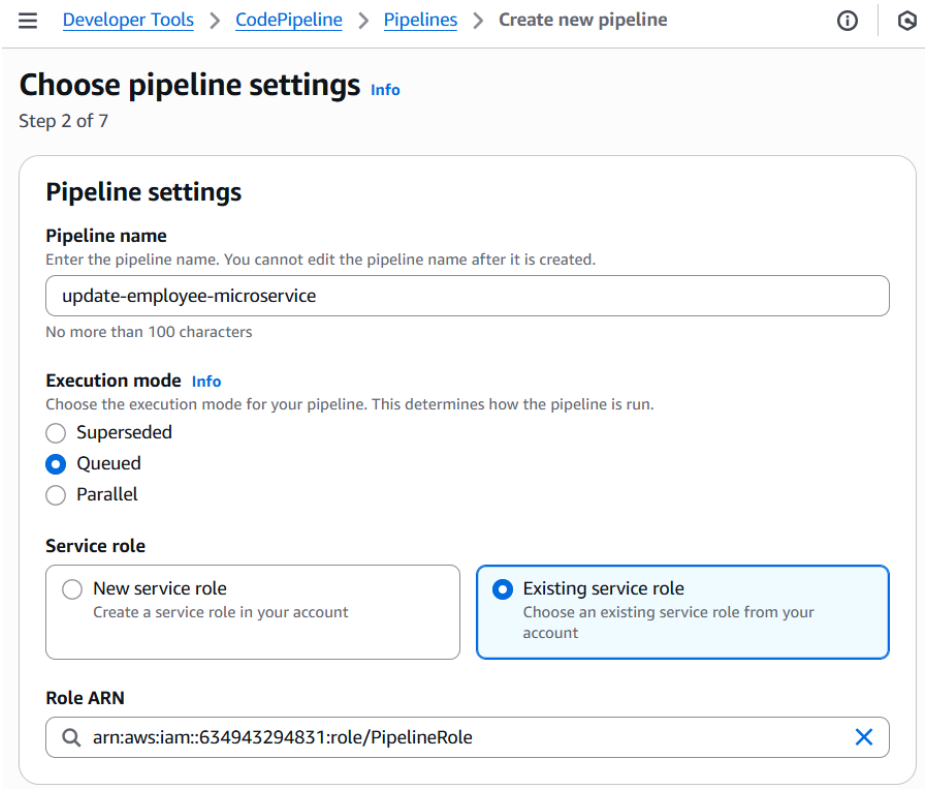
Then, update the Deploy section by adding “image-customer” for input artifacts. Then, choose “image-customer” for input artifact and type “IMAGE1_NAME” for the placeholder. Refer the configuration as shown below:

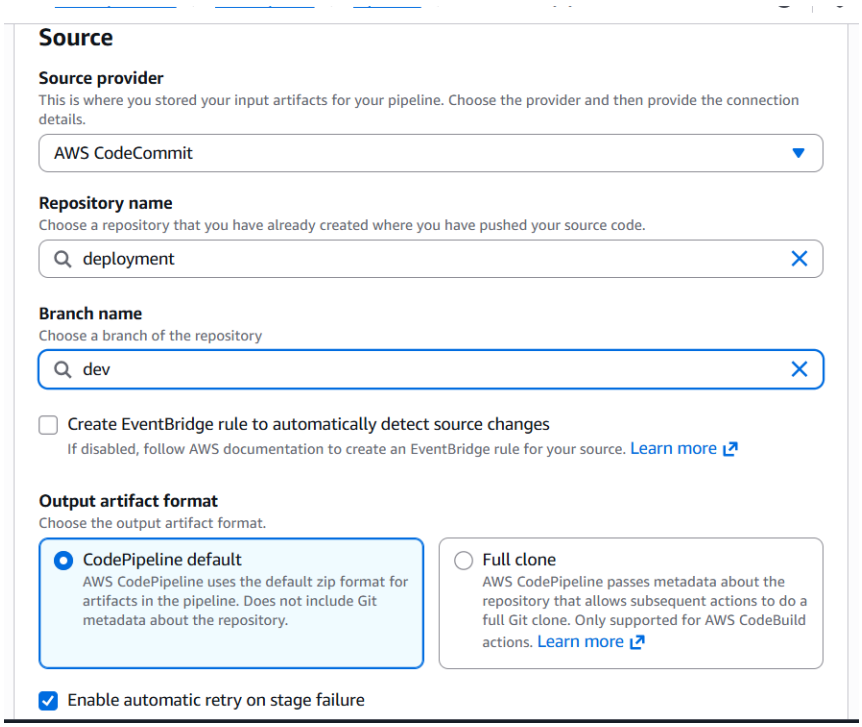


Task 5: Create Pipeline for Employee Microservice

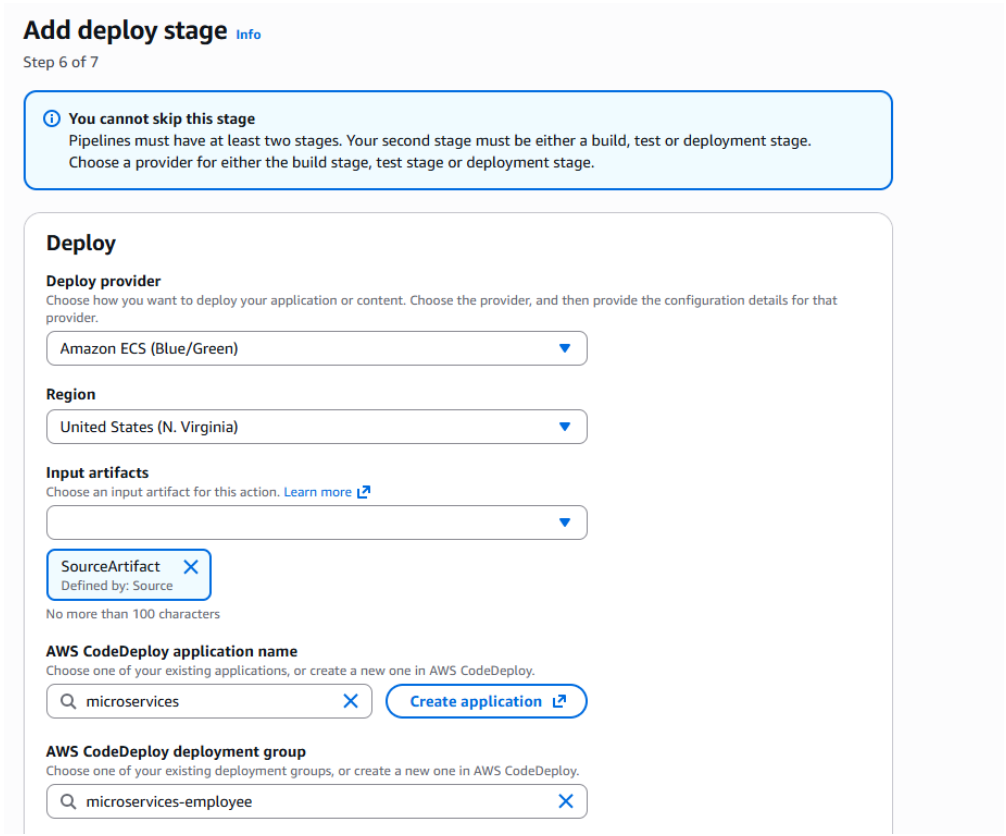
1. Navigate to “Developer Tools” → CodePipeline → Pipelines → Create new pipeline. At here, create a pipeline called “update-employee-microservice”. Configure as shown below:

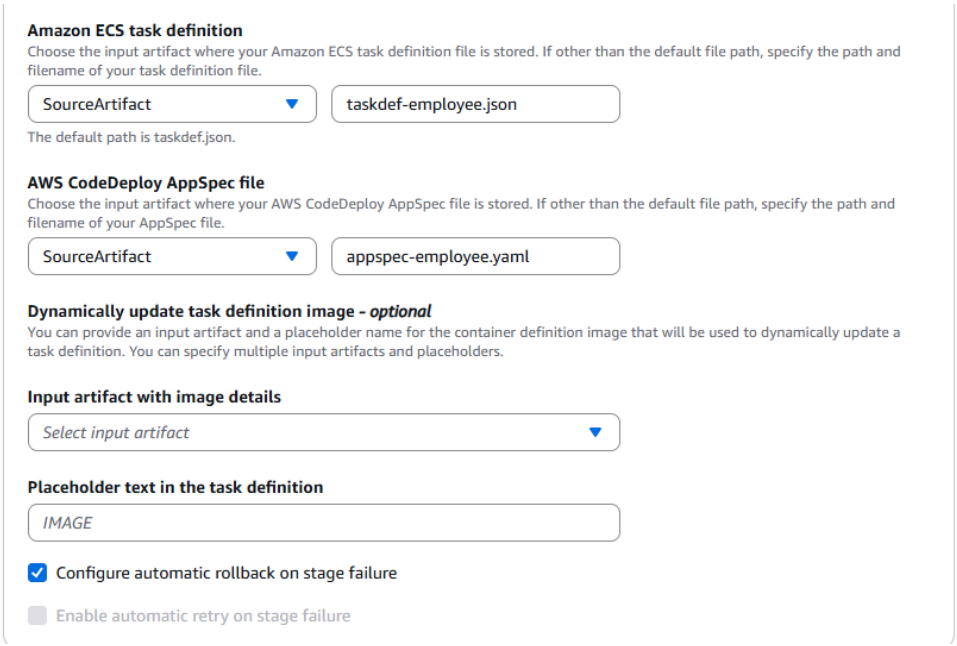




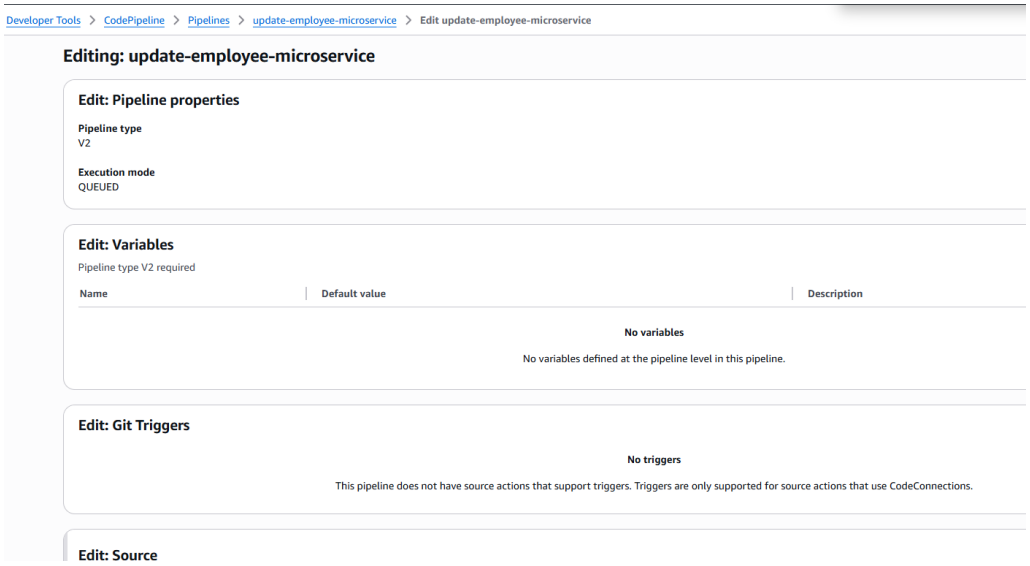


Skip step 4 “build stage” and step 5 “add test stage”.

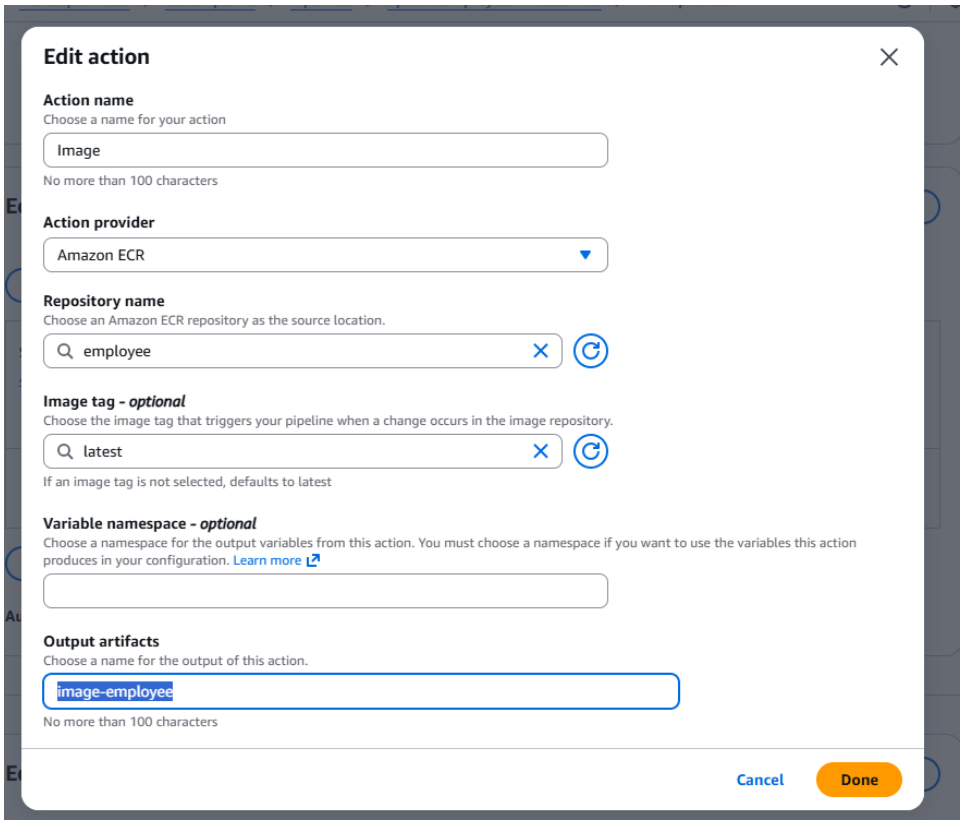




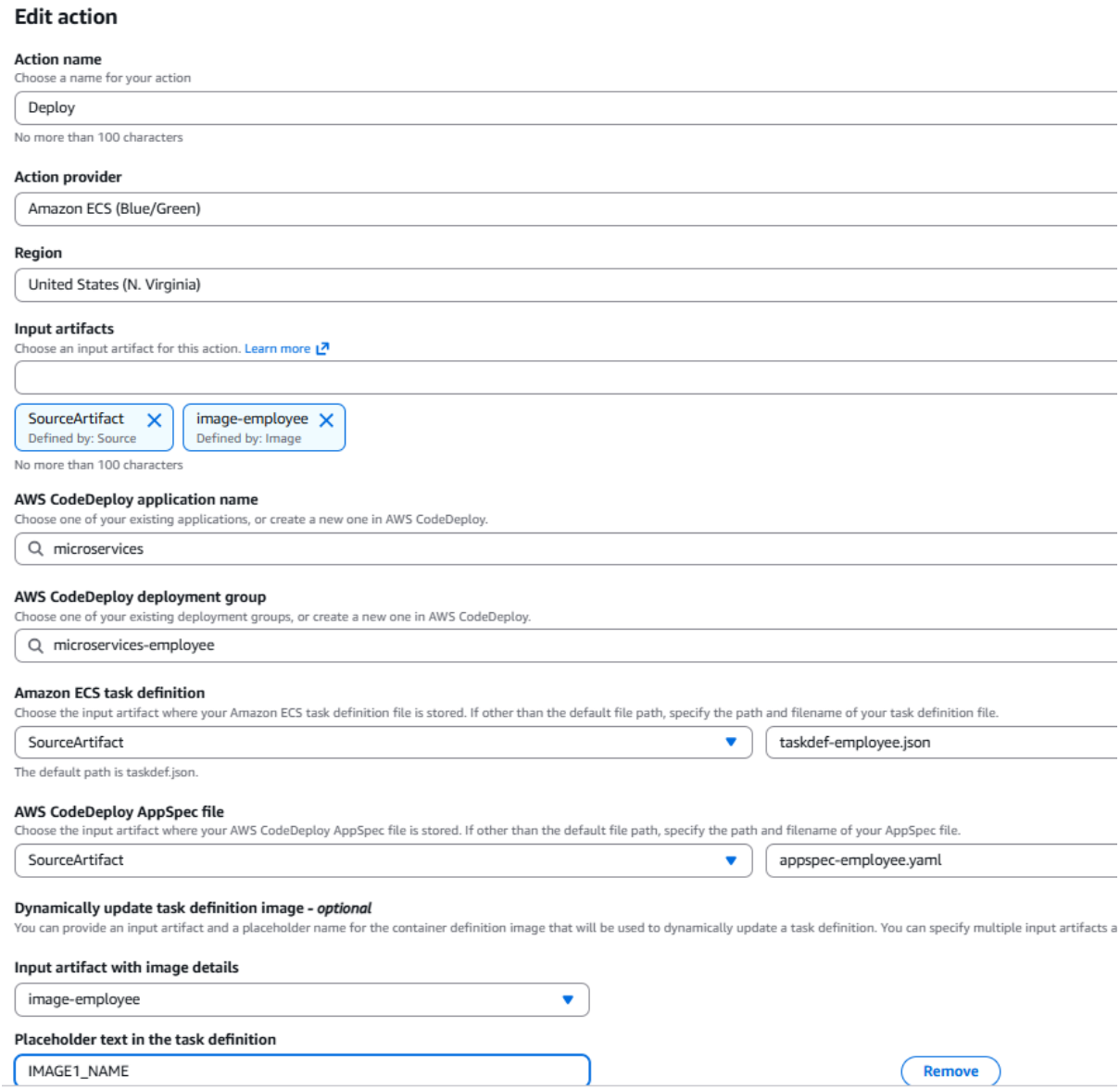
2. Navigate to Developer Tools → CodePipeline → Pipelines → update-employee-microservice → Edit



Then, update the Source section by adding a new action as shown below:

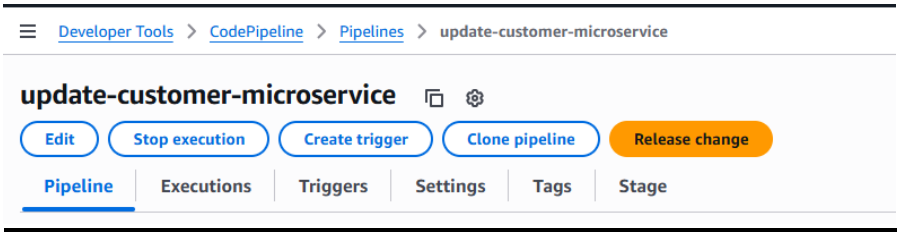


Then, update the Deploy section by adding “image-employee” for input artifacts. Then, choose “image-employee” for input artifact and type “IMAGE1_NAME” for the placeholder. Refer the configuration as shown below:

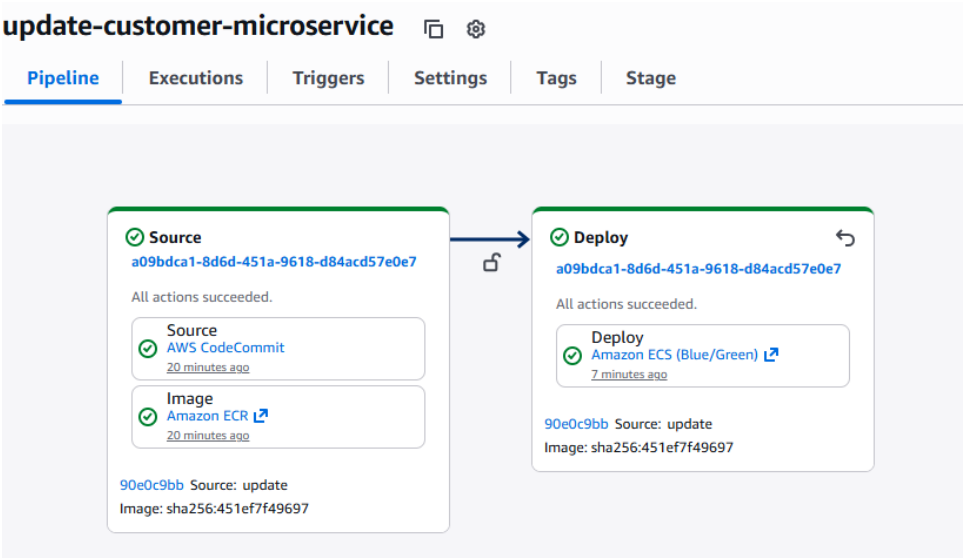


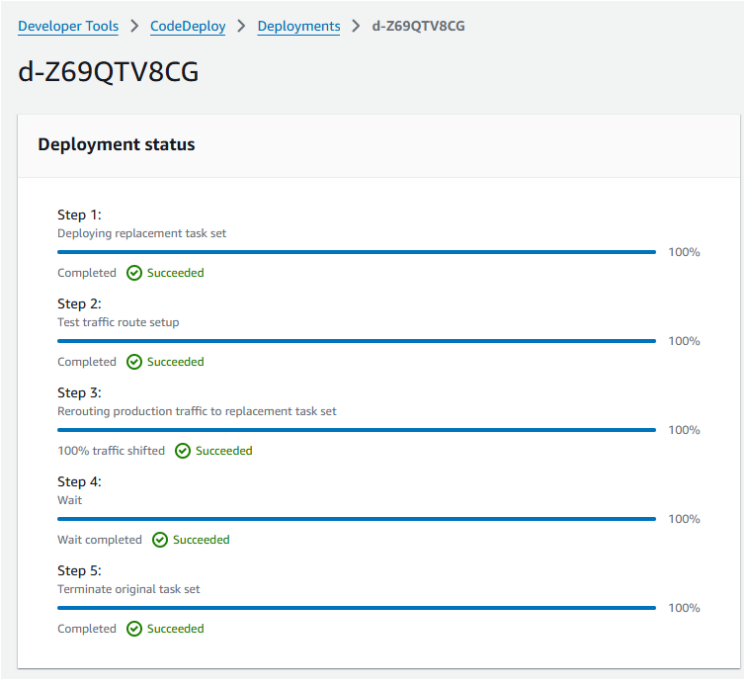
Task 6: Test CI/CD Pipeline for Customer and Employee Microservices

1. Navigate to “Developer Tools” → CodePipeline → Pipelines → update-customer-microservice, then click “Release change”:

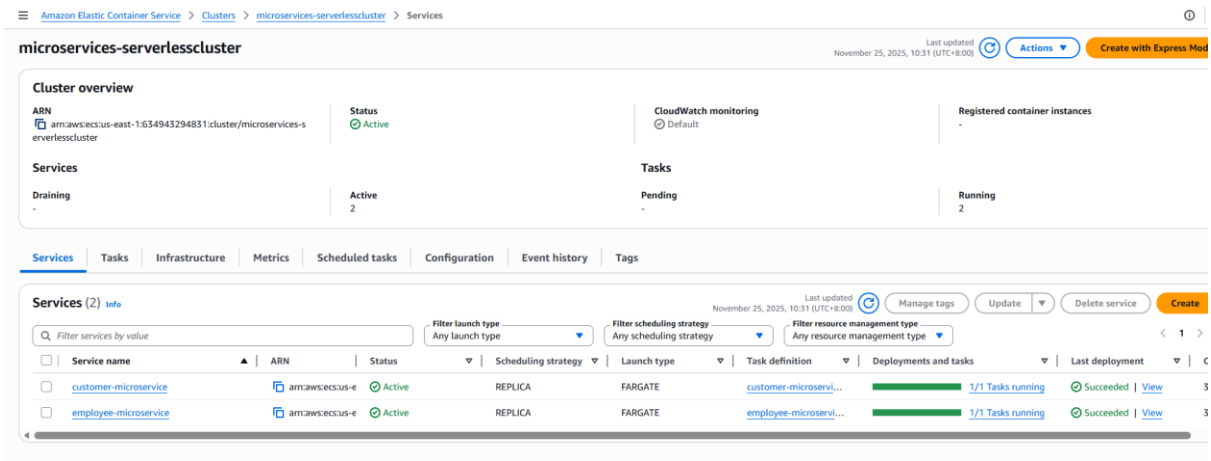


2. Wait for few minutes and observe the changes . It should shows Deploy succeeded.



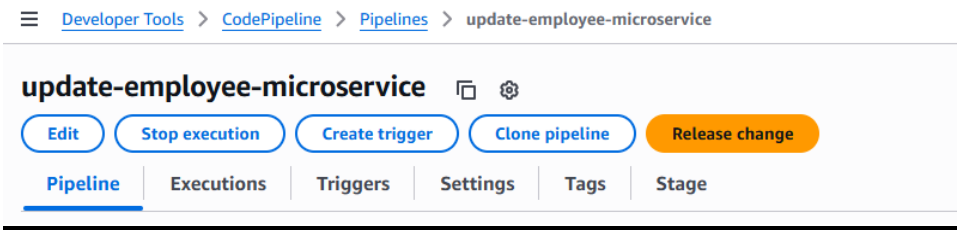


3. Load the customer microservice in browser by using the DNS name of the microservicesLB load balancer (which run on the customer microservice):

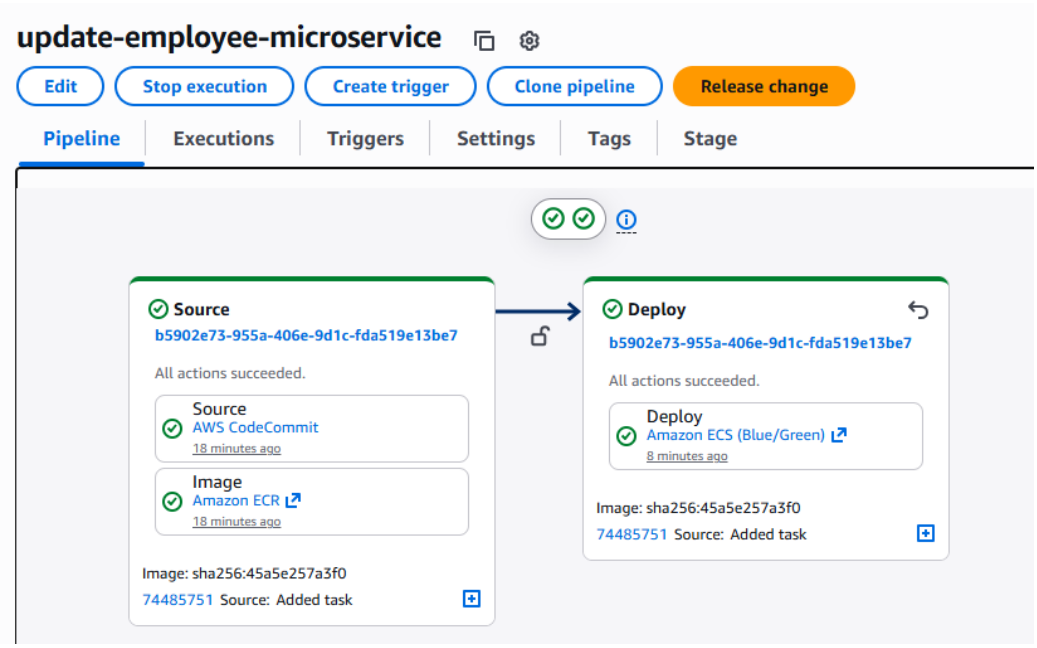


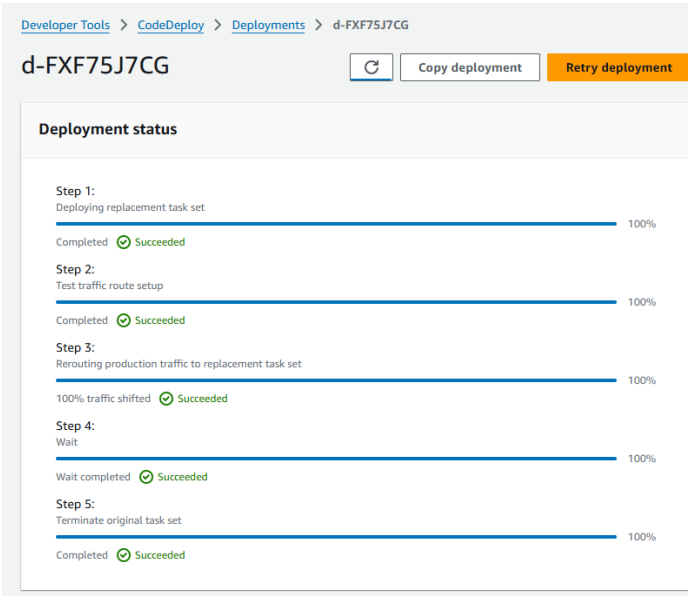
Task 7: Test CI/CD Pipeline for Employee Microservices

1. Navigate to “Developer Tools” → CodePipeline → Pipelines → update-employee-microservice, then click “Release change”:

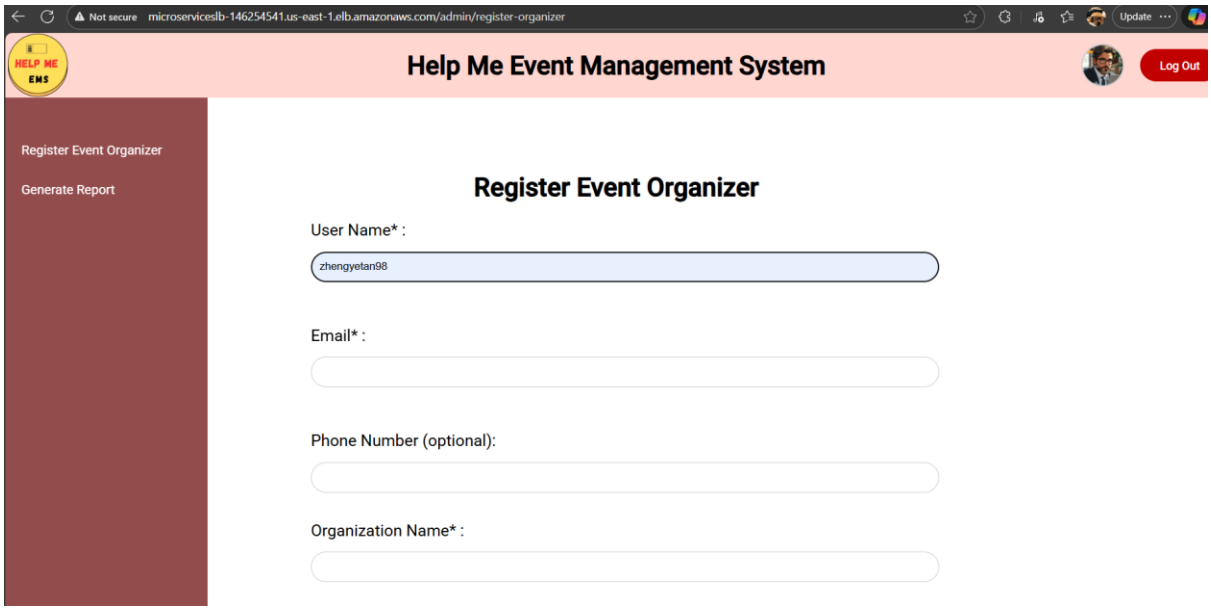


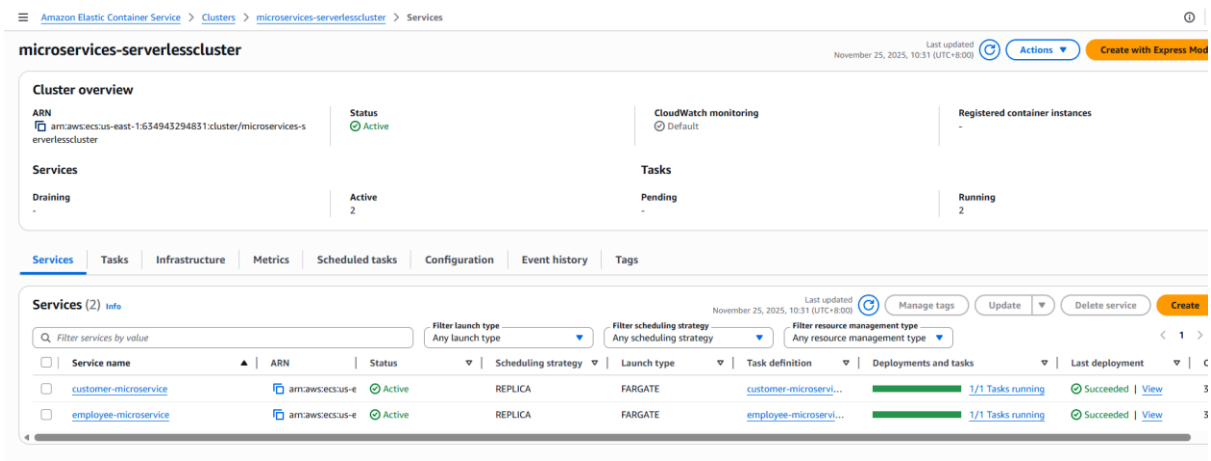
2. Wait for few minutes and observe the changes . It should shows Deploy succeeded.





3. Load the employee microservice in browser by using the DNS name of the microservicesLB load balancer with /admin/ path, then it will direct to admin page (which run on the employee microservice) :

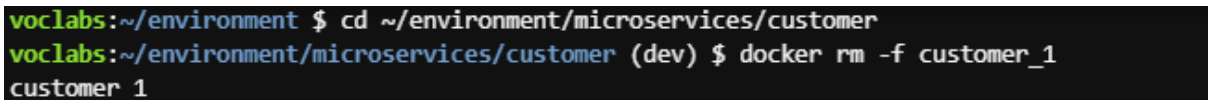


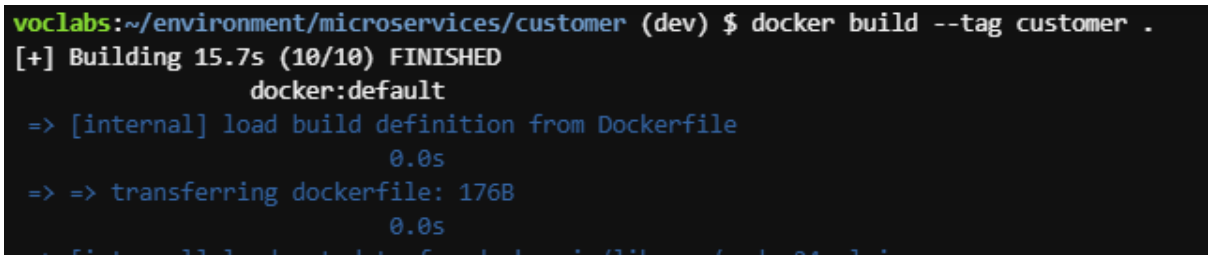


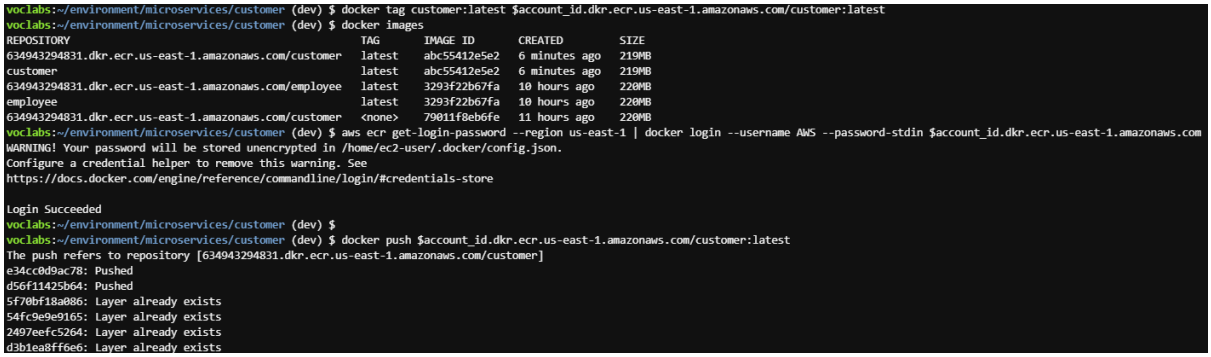
Task 8: Modify Microservices Code to Cause Pipeline Run Again

1. Modify the UI for the customer microservice. In the index.html for the customer, update the **browser tab title** to "**HELP Me EMS**".

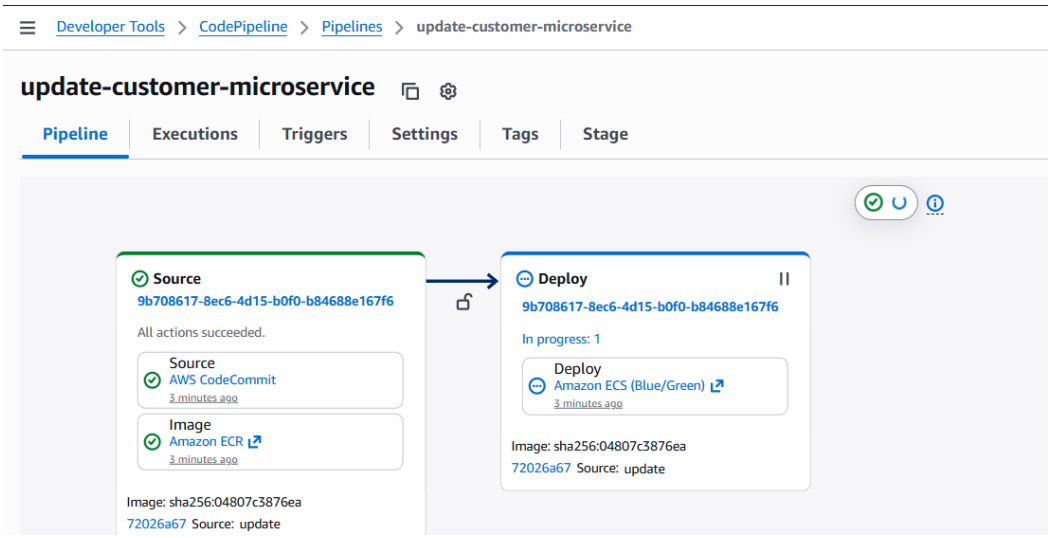
2. Regenerate a new Docker image and push it to ECR



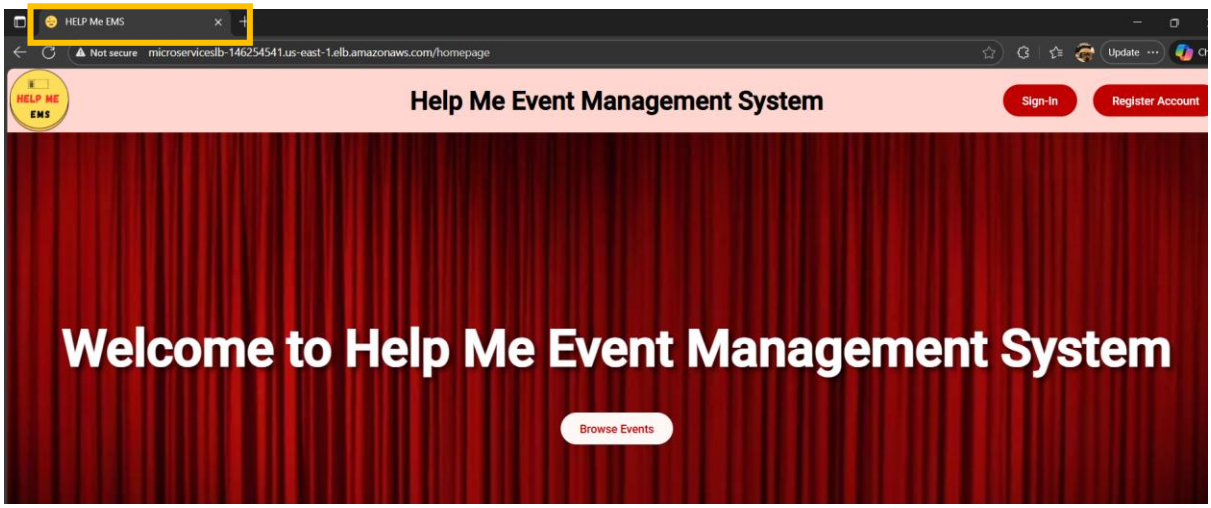


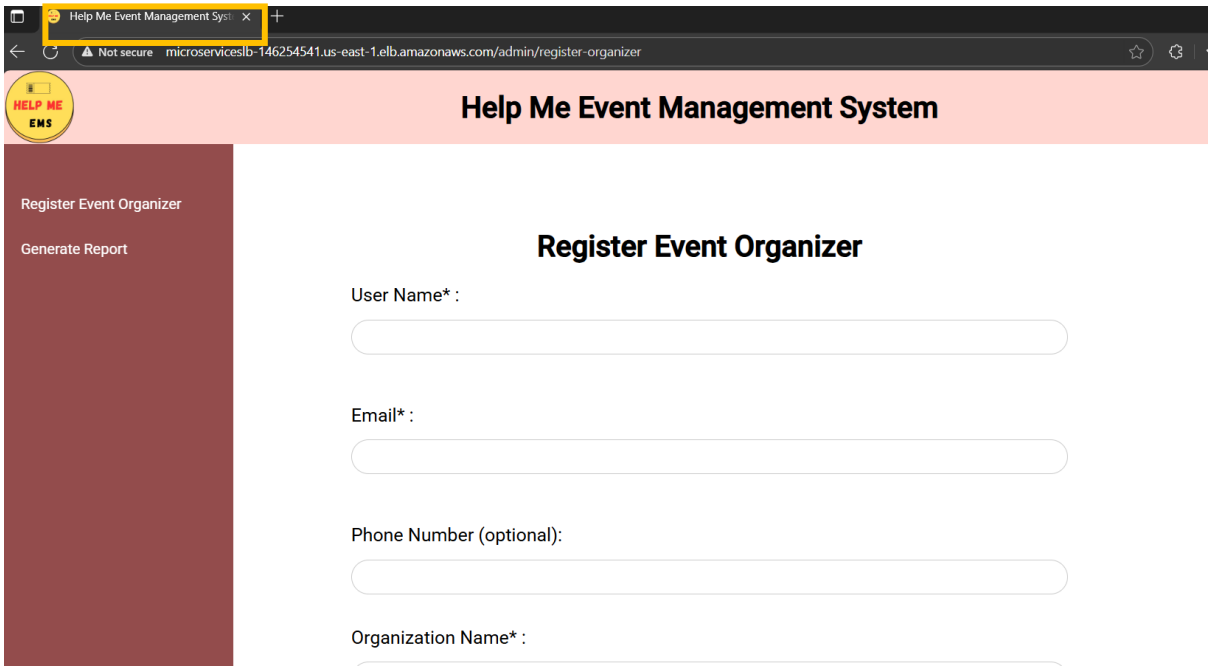


3. Notice that the pipeline for the “update-customer-microservice” is auto triggered and run



4. Test the latest change of UI for the customer page in browser. As the picture below show that the **browser tab title** of the customer view side has been changed. Now, go to the admin view side, the **browser tab title** did not change as we did not change for the employee microservice.



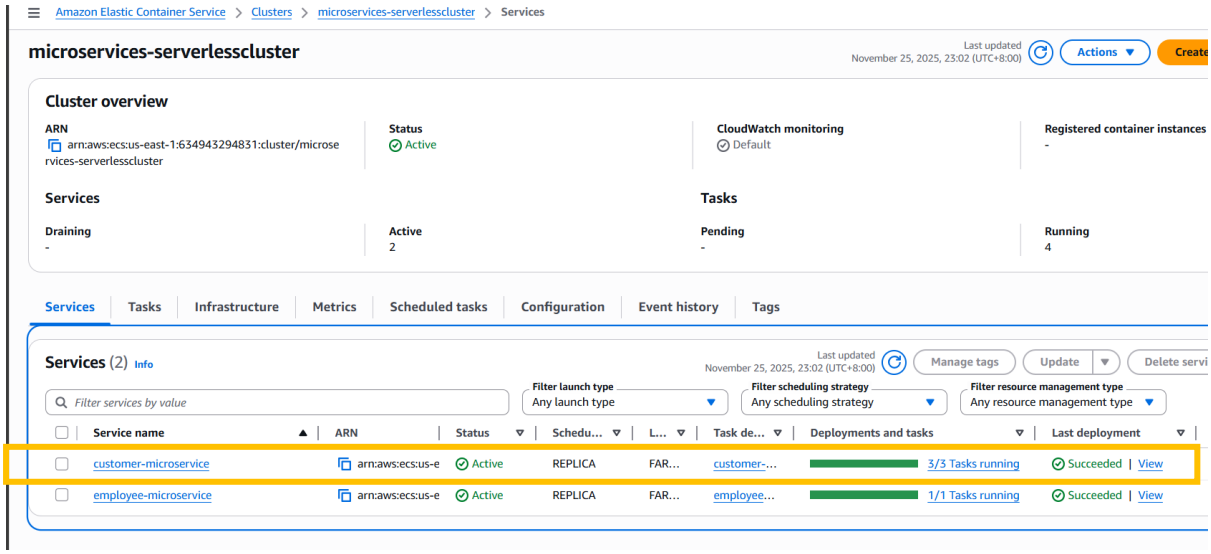

Help Me Event Management Syst
Not secure
microserviceslb-146254541.us-east-1.elb.amazonaws.com/admin/register-organizer
HELP ME EMS
Help Me Event Management System
Register Event Organizer
Generate Report
Register Event Organizer
User Name* :
Email* :
Phone Number (optional):
Organization Name* :

Task 9: Scale Customer Microservice

1. Scale up the container number for customer microservice to three containers

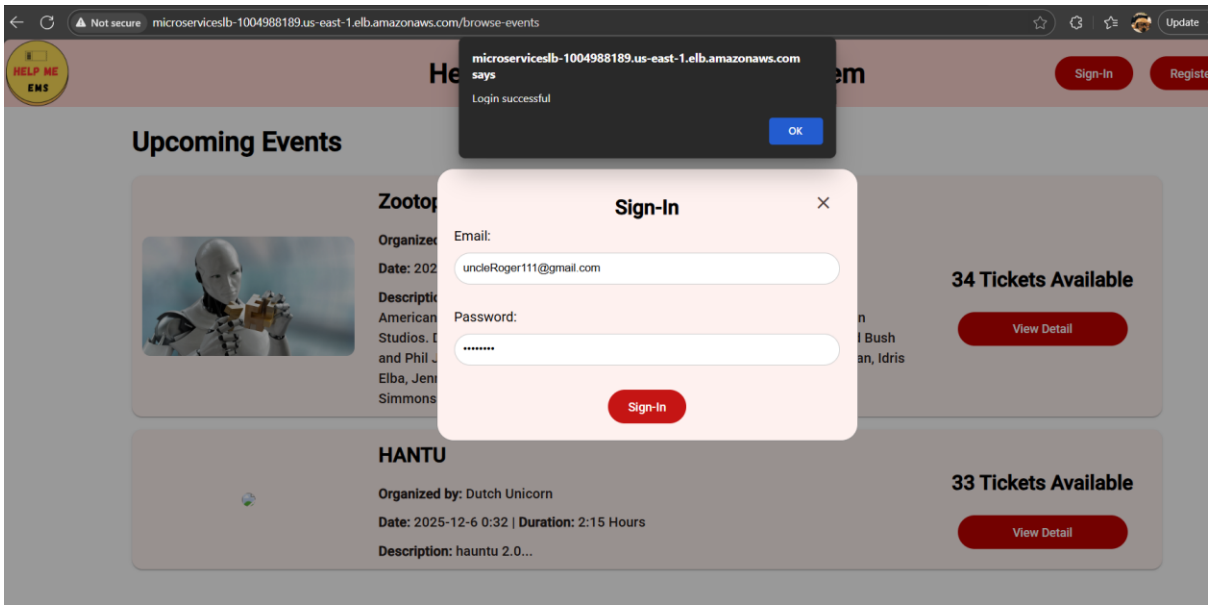
```
voclabs:~/environment/microservices/customer (dev) $ aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:634943294831:service/microservices-serverlesscluster/customer-microservice",
        "serviceName": "customer-microservice",
        "clusterArn": "arn:aws:ecs:us-east-1:634943294831:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:634943294831:targetgroup/customer-tg-two/3346420c7f51b5ae",
                "containerName": "customer",
                "containerPort": 8080
            }
        ],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 3,
        "runningCount": 1,
        "pendingCount": 0,
```

2. From here, it shows that **the total container number for customer microservice** is **3** now.

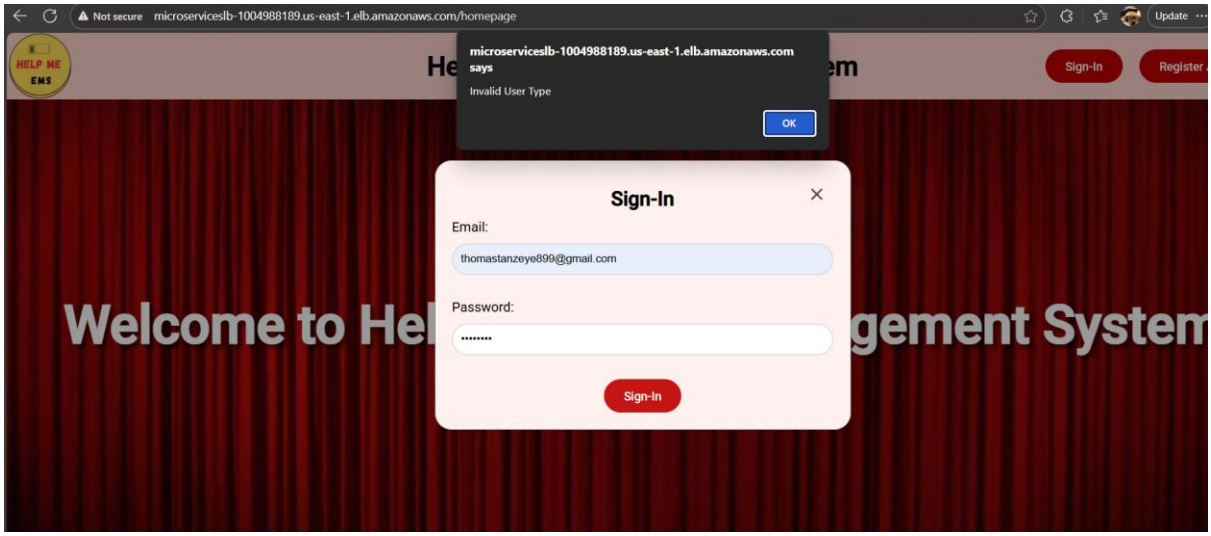


Task 10: Validate the Function of Customer Microservice

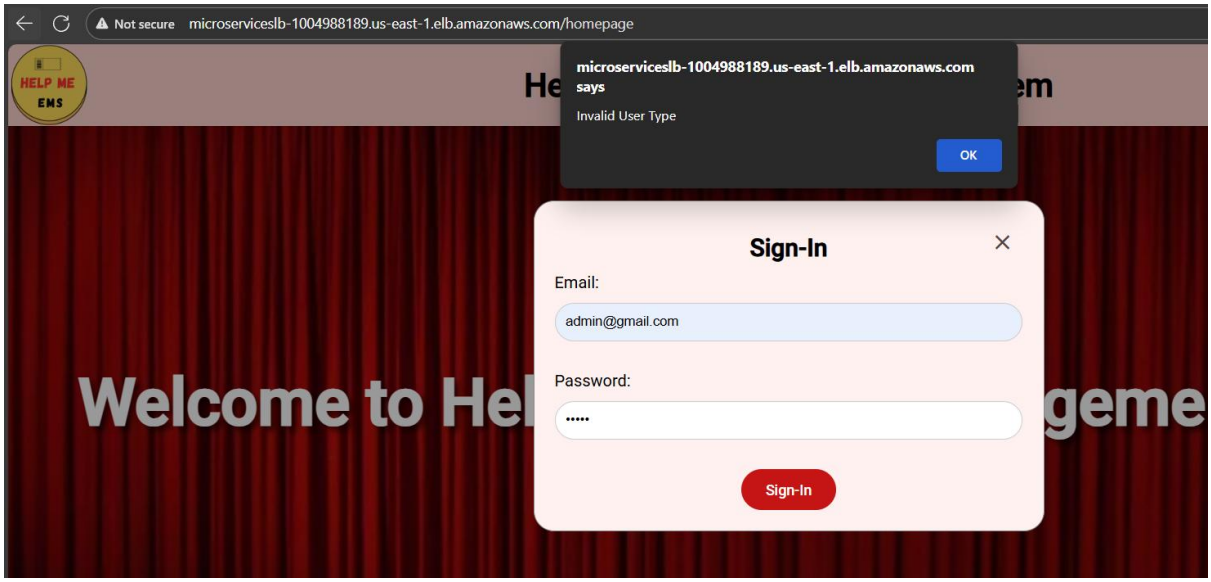
1. In Customer Microservice , login as "**attendee**" account, will be succeed. Then, they can continue browse event, book event, checkout, and add/remove event to waitlist.



2. In Customer Microservice , login as "**event organiser**" account, will be failed.

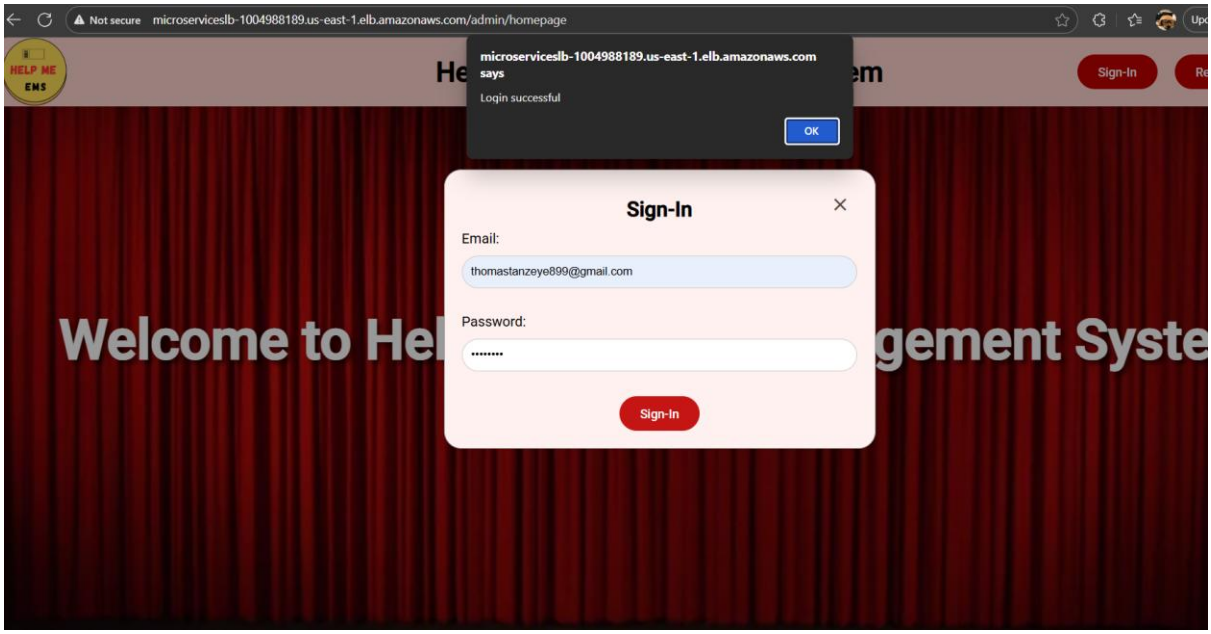


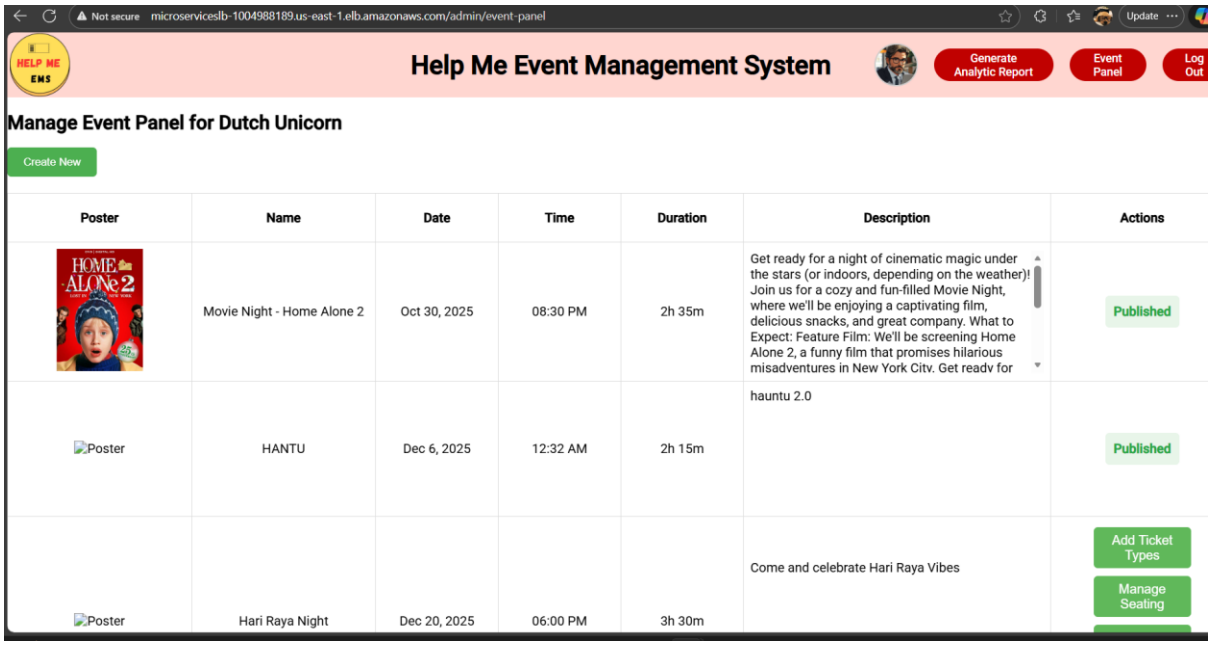
3. In Customer Microservice , login as “**admin**” account, will be failed.



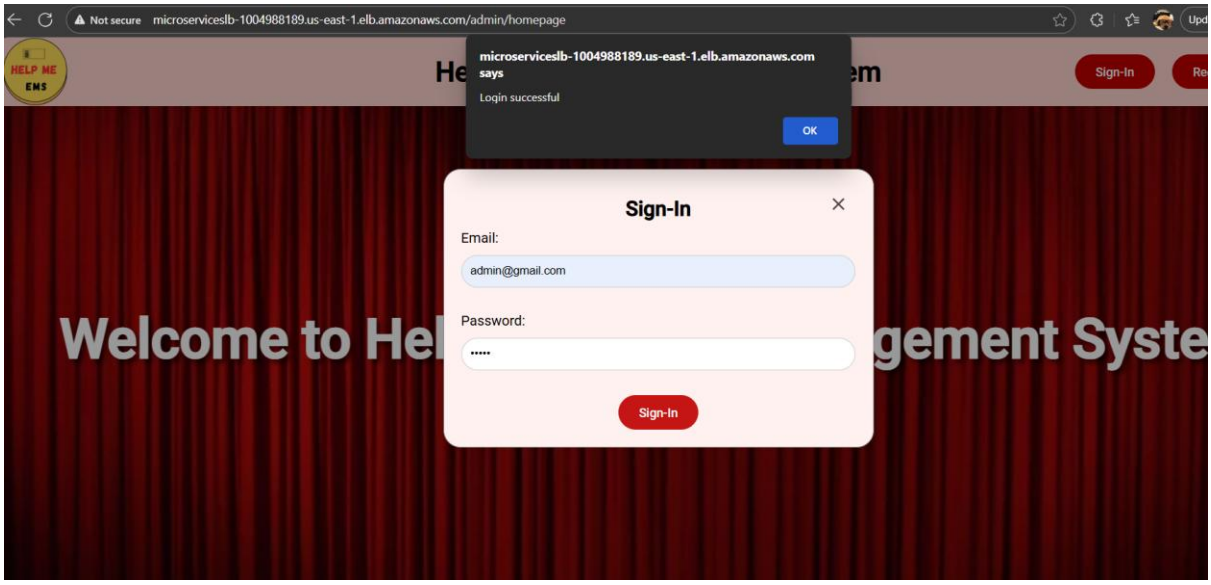
Task 11: Validate the Function of Employee Microservice

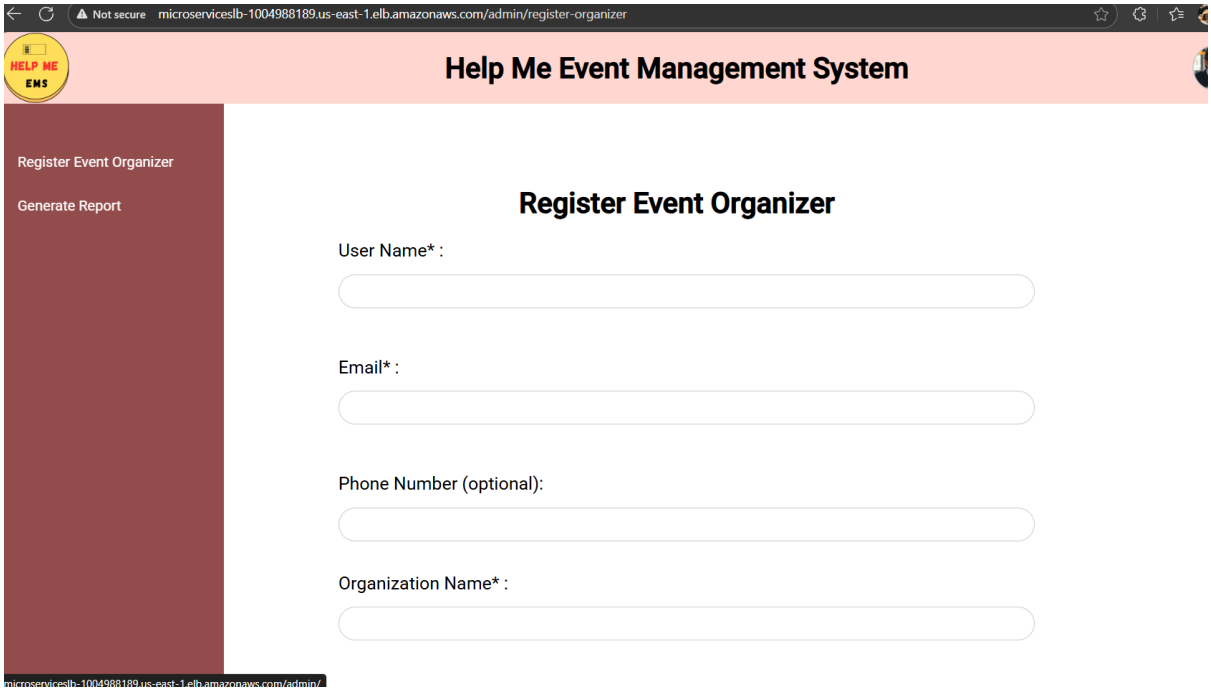
1. In Employee Microservice , login as "**event organiser**" account, will be succeed. Then, they are redirected to event management panel page for edit/create event functions.





2. In Employee Microservice , login as "**admin**" account, will be succeed. Then, they are redirected to admin page where they can register new event organiser here.

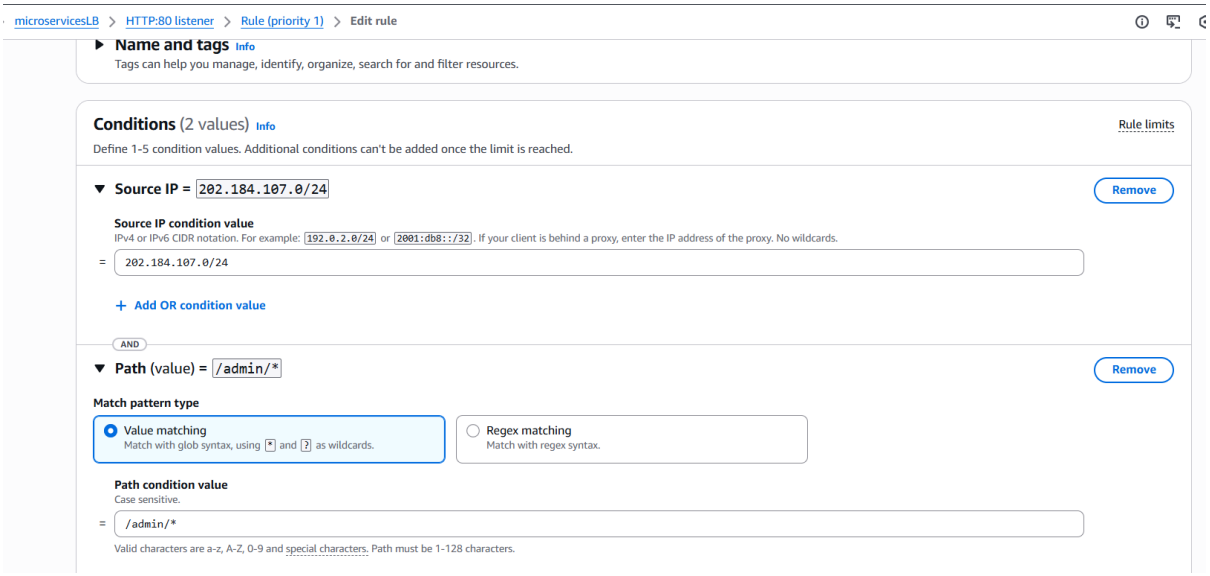


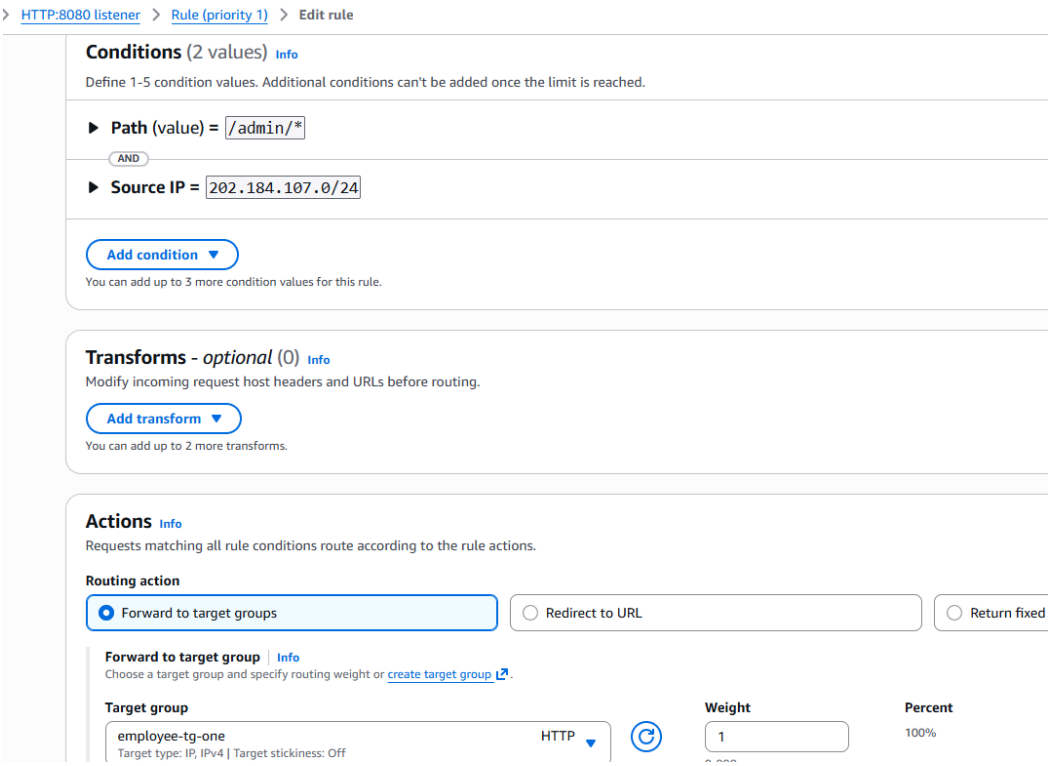


Task 12: Limit Access to the Employee Microservice

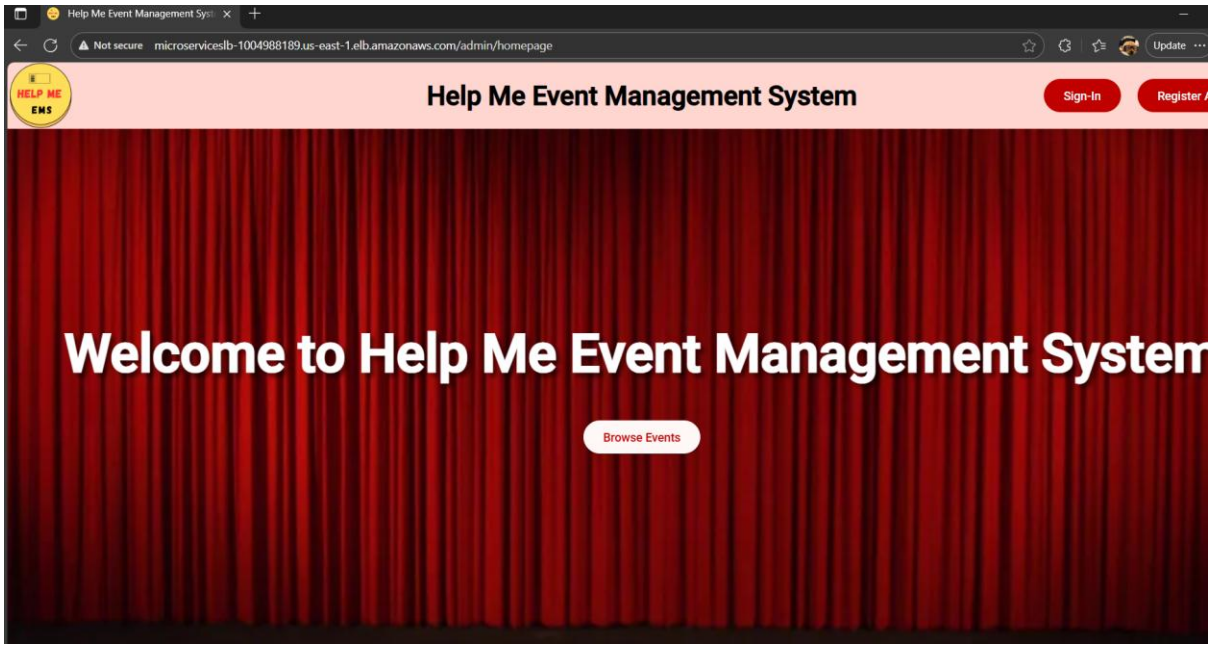
1. For the microservices load balancer, I updated the HTTP:80 & 8080 listener rule responsible for the /admin/* path. I added a Source IP condition to ensure that only users connecting from my specified subnet range can access the employee microservice.

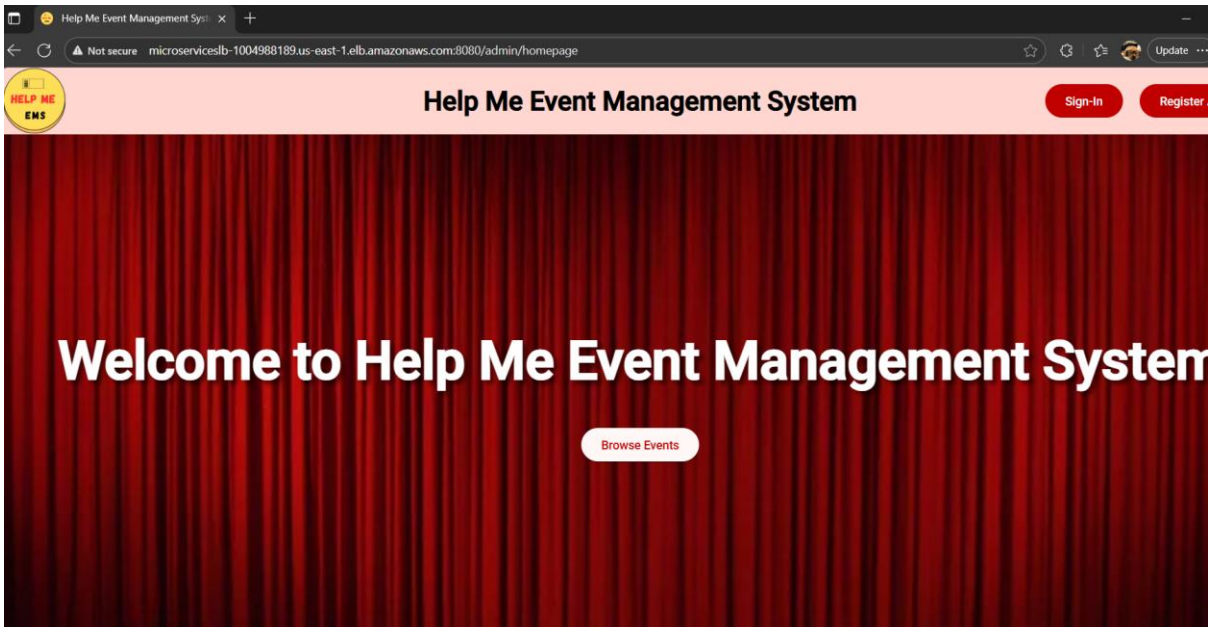
   Because the rule now contains two conditions: Path = /admin/* and Source IP = <my subnet>, both conditions must be met before the load balancer forwards the request to the employee microservice target group. If either condition fails, the request is routed to the default customer microservice instead.





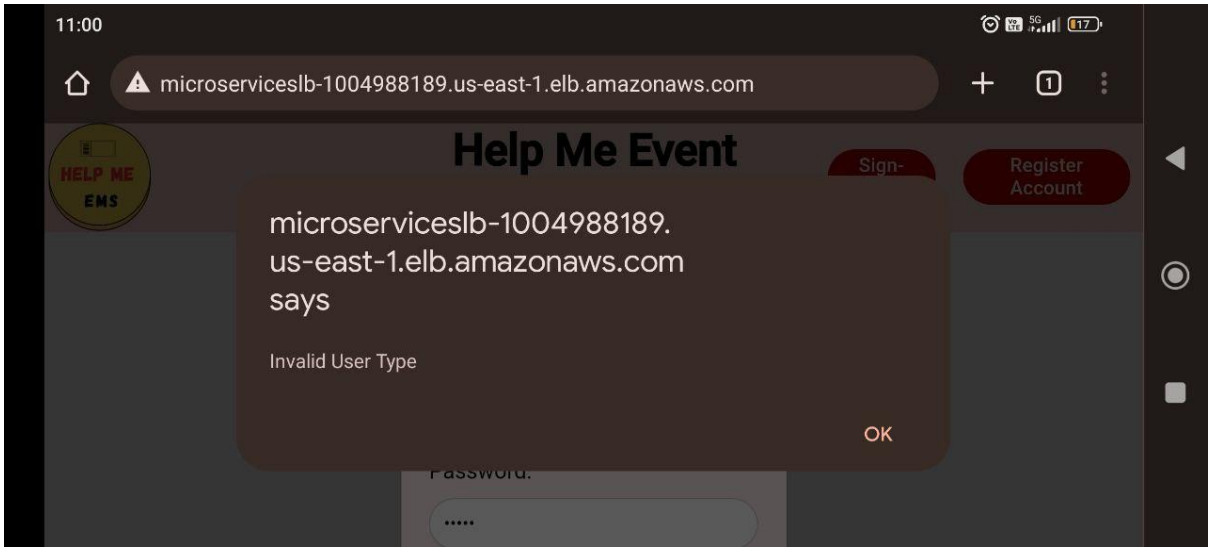
2. Use same device (laptop) to test and validate that the employee microservice (for HTTP:80 and HTTP:8080) can be accessed only if the user is connected under the same IP subnet.



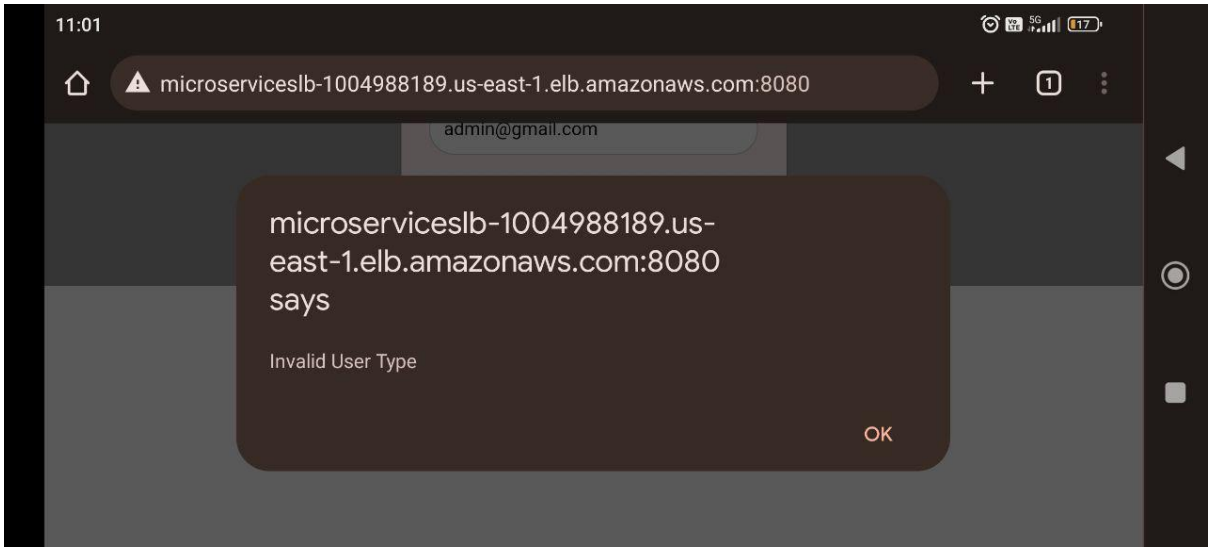


3. Use different device (mobile phone) that use mobile data to test and validate that the employee microservice (for HTTP:80 and HTTP:8080) cannot be accessed if the user is not connected under the same IP subnet that we set.

After type /admin/ to the path, the website still redirects to the customer microservice (without /admin/ path) because of the subnet condition didn't meet.


11:00
microserviceslb-1004988189.us-east-1.elb.amazonaws.com
Help Me Event
HELP ME EMS
Sign-
Register Account
microserviceslb-1004988189.
us-east-1.elb.amazonaws.com
says
Invalid User Type
OK
Password:
.....


11:00
http://microserviceslb-1004988189.us-east-1.elb.amazonaws.com:8080/admin/
HELP Me EMS
http://microserviceslb-1004988189.us-east-1.elb.amazonaws.com:8080/admin/
Help Me Event Management System
http://microserviceslb-1004988189.us-east-1.elb.amazonaws.com:8080/admin/homepage
the
I
to
1 2 3 4 5 6 7 8 9 0
@ # £ _ & - + ( ) /
=\< * " ' : ; ! ?
ABC
12 34
English


11:01
microserviceslb-1004988189.us-east-1.elb.amazonaws.com:8080
admin@gmail.com
microserviceslb-1004988189.us-
east-1.elb.amazonaws.com:8080
says
Invalid User Type
OK

\