

2.0 Analysis of Monolithic Application & Microservices Conversion

2.1 Original Monolithic Application Architecture

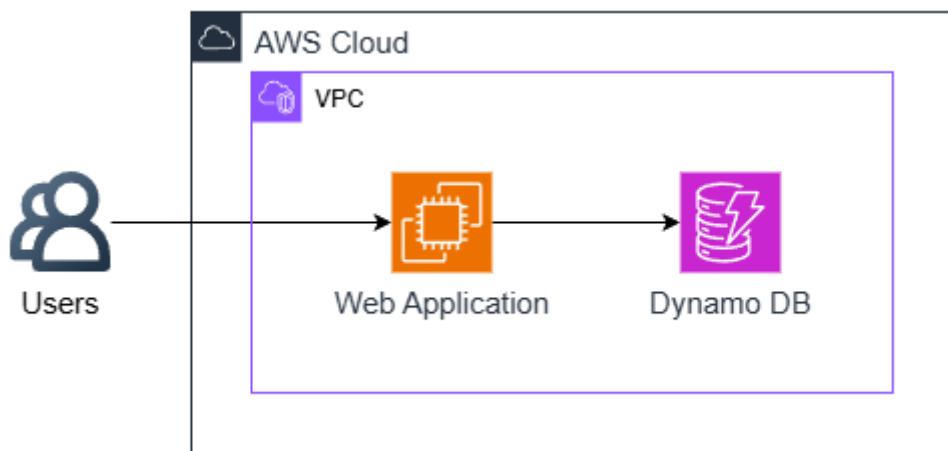


Figure 1: Original Monolithic Application Architecture

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

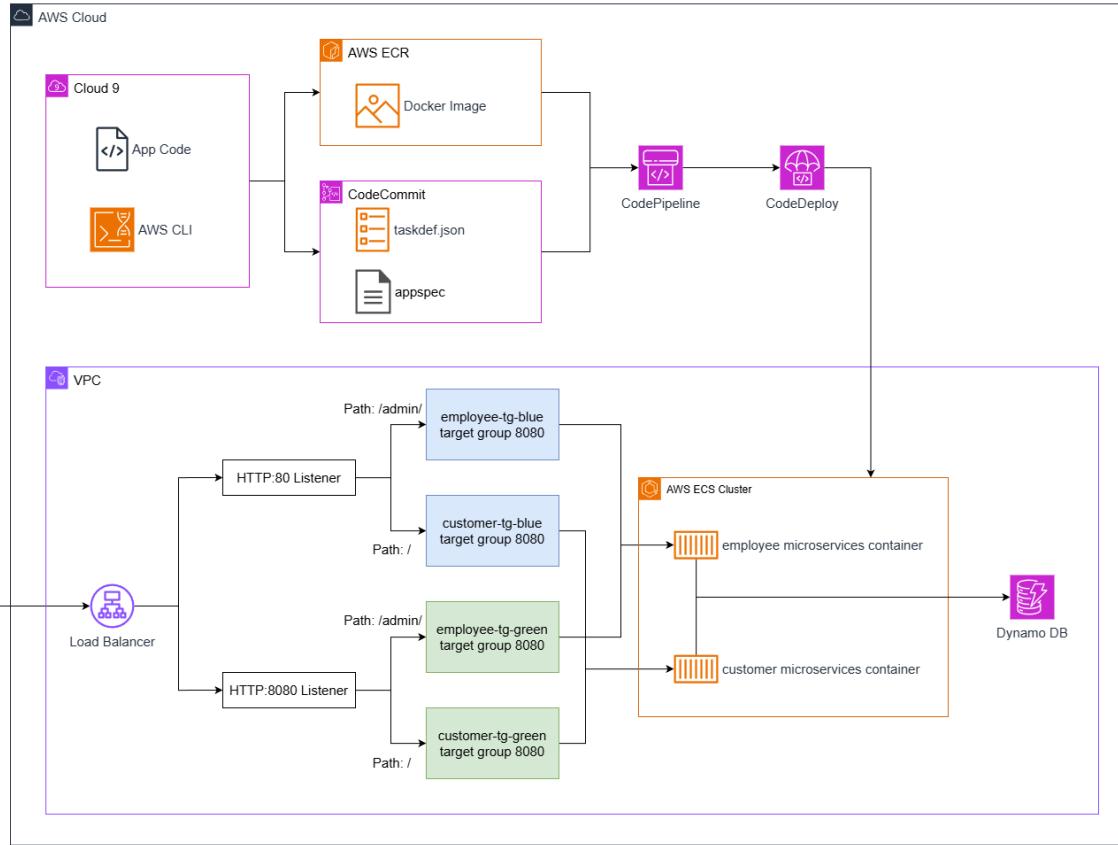


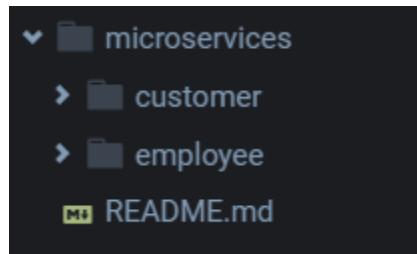
Figure 2: Microservices Application Architecture

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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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)

	<ul style="list-style-type: none">• 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:

S > Create environment

Name
MicroservicesIDE
Limit of 60 characters, alphanumeric, and unique per user.

Description - optional
Microservices IDE
Limit 200 characters.

Environment type Info
Determines what the Cloud9 IDE will run on.

New EC2 instance
Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute
You have an existing instance or server that you'd like to use.

New EC2 instance

Instance type Info
The memory and CPU of the EC2 instance that will be created for Cloud9 to run on.

t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.

t3.small (2 GiB RAM + 2 vCPU)
Recommended for small web projects.

m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and most general-purpose development.

Additional instance types
Explore additional instances to fit your need.

Platform Info
This will be installed on your EC2 instance. We recommend Amazon Linux 2023.

Amazon Linux 2

Timeout
How long Cloud9 can be inactive (no user input) before auto-hibernating. This helps prevent unnecessary charges.

1 day

Network settings Info

Connection
How your environment is accessed.

AWS Systems Manager (SSM)
Accesses environment via SSM without opening inbound ports (no ingress).

Secure Shell (SSH)
Accesses environment directly via SSH, opens inbound ports.

VPC settings Info

Amazon Virtual Private Cloud (VPC)
The VPC that your environment will access. To allow the AWS Cloud9 environment to connect to its EC2 instance, attach an internet gateway (IGW) to your VPC. [Create new VPC](#)

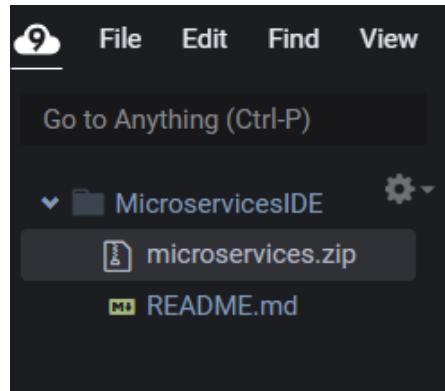
vpc-07641493d4ed4eeae
Name - LabVPC

Subnet
Used to setup your VPC configuration. To use a private subnet, select AWS Systems Manager (SSM) as the connection type. [Create new subnet](#)

No preference
Uses default subnet in any Availability Zone

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**

A screenshot of the Cloud IDE interface showing the file tree on the left and a terminal window on the right. The terminal window has tabs for 'bash - "ip-172-31-12-191.4x"' and 'Immediate'. The 'Immediate' tab shows the command 'vocabs:~/environment \$ unzip microservices.zip' and its output, which lists the creation and inflation of various files and directories within the 'microservices' folder structure. The 'microservices.zip' file is highlighted in the file tree.

3. Create a CodeCommit repository named “**microservices**”:

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.

Repository settings

Repository name

microservices

100 characters maximum. Other limits apply.

Description - optional

microservices

1,000 characters maximum

Tags

4. Initialize the Git repository and push the code:

```
vocabs:~/environment/microservices $ cd ~/environment/microservices
vocabs:~/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/
vocabs:~/environment/microservices (master) $ git branch -m dev
vocabs:~/environment/microservices (dev) $ git add .
vocabs:~/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
vocabs:~/environment/microservices (dev) $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
vocabs:~/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

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Images, Elastic Block Store, Network & Security, and more. The main area displays a table of instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP
MonolithicApp...	i-0439c6d4c92e92b9f	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-82-225-100.co...	54.82.225.100	-
aws-cloud9-MI...	i-03e06abb1f01bd1ce	Running	t3.small	3/3 checks passed	View alarms	us-east-1a	ec2-34-228-63-68.com...	34.228.63.68	-

Below the table, a detailed view is shown for the instance with ID i-0439c6d4c92e92b9f, labeled "MonolithicAppServer". The "Security" tab is selected in the navigation bar. The "Security details" section shows the IAM Role (which is empty) and Security groups (sg-0b511e01609b6012f). The "Inbound rules" section is partially visible at the bottom.

- 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.

- From the AWS Details panel on this lab instructions page, download the labsuser.pem file to your local computer.
- Upload the .pem file to AWS Cloud9 IDE, and use “chmod 400 labsuser.pem” command to set the proper permissions on the file.
- 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.
- 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"
```

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

```

1  #!/bin/bash
2  PID=$(pgrep -f "sudo node index.js")
3  sudo kill $PID
4
5  sudo apt install openjdk-21-jre-headless
6  sleep 2
7
8  wget https://d1ni2b6xgvw0s0.cloudfront.net/v2.x/dynamodb_local_latest.zip
9  unzip dynamodb_local_latest.zip
10 sleep 2
11
12 java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb &
13
14 sudo sh -c "cat <<EOF > /etc/rc.local
15 #!/bin/bash
16 cd /home/ubuntu/
17 sudo java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
18 EOF
19 "
20 sudo chmod +x /etc/rc.local

```

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

pip3 install boto3

```

ments.js
tbookings.j
ts.js
ets.js
ers.js
tlists.js
ite
age
ink-5SUUBA
ink-EQDQRF
ink-M56EKV
ink-PM3QAI

python3.8 -i ip-172-31-12-x
vclabs:~/environment $ pip3 install boto3
Defaulting to user installation because normal site-packages is not writable
Collecting boto3
  Downloading boto3-1.37.38-py3-none-any.whl (139 kB)
    |████████| 139 kB 12.3 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.8/site-packages (from boto3) (1.0.1)
Requirement already satisfied: botocore<1.38.0,>=1.37.38 in /usr/local/lib/python3.8/site-packages (from boto3) (1.37.38)
Collecting s3transfer<0.12.0,>=0.11.0
  Downloading s3transfer-0.11.5-py3-none-any.whl (84 kB)
    |████████| 84 kB 5.0 MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.8/site-packages (from boto3) (2.8.1)

```

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.

EC2 > Security Groups > sg-047810a71ba0d0d9b - aws-cloud9-MicroservicesIDE-7e7c5948585b40899bc8a61808131f70-InstanceSecurityGroup-YkVM422kfopU > Edit inbound rules

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description
sgr-062d9dd1a1030e8c8	SSH	TCP	22	Custom	35.172.155.192/27
sgr-0253e19068d0ddf62	Custom TCP	TCP	8080	Custom	0.0.0.0/0
sgr-0ec547a533f56c249	Custom TCP	TCP	8081	Custom	0.0.0.0/0
sgr-0589a57972c27f683	SSH	TCP	22	Custom	35.172.155.96/27

[Add rule](#)

Task 2: Create Docker Image for Customer Microservice

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

The screenshot shows a file tree on the left with the following structure:

- Microservices/
- database
- microservices/
- customer/
- models
- website
- JS app.js
- Dockerfile

On the right, there is a code editor window titled "Dockerfile" containing the following Dockerfile content:

```
FROM node:24-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

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

```
voclabs:~/environment/microservices (dev) $ cd ~/environment/microservices/customer
voclabs:~/environment/microservices/customer (dev) $ docker build --tag customer .
[+] Building 16.9s (10/10) FINISHED
   docker:default
=> [internal] load build definition from Dockerfile          0.0s
=> => transferring dockerfile: 176B                         0.0s
=> [internal] load metadata for docker.io/library/node:24-alpine 0.3s
=> [internal] load .dockerignore                            0.0s
=> => transferring context: 2B                           0.0s
=> [1/5] FROM docker.io/library/node:24-alpine@sha256:2867d550cf 2.8s
=> => resolve docker.io/library/node:24-alpine@sha256:2867d550cf 0.0s
=> => sha256:2d35ebdb57d9971fea0cac1582aa78935ad 3.80MB / 3.80MB 0.1s
=> => sha256:0ad6b75bfecf09f5483895f7a30ab9af8 50.91MB / 50.91MB 0.6s
=> => sha256:572392f439b9cba4dd81d87da90e1b64140 1.26MB / 1.26MB 0.1s
=> => sha256:2867d550cf9d8bb50059a0fff528741f11a 3.87kB / 3.87kB 0.0s
=> => sha256:d7a3af308e4006ba6d66bbae980fe39c118 1.72kB / 1.72kB 0.0s
=> => sha256:49e0ad1279de1597a3bf4335aa0b6097fab 6.50kB / 6.50kB 0.0s
=> => extracting sha256:2d35ebdb57d9971fea0cac1582aa78935adf8058 0.2s
=> => sha256:e731d1c17be0459197f82c0ffe5922a848f40c7 447B / 447B 0.1s
=> => extracting sha256:0ad6b75bfecf09f5483895f7a30ab9af8315d1d2 1.9s
=> => extracting sha256:572392f439b9cba4dd81d87da90e1b64140478de 0.0s
=> => extracting sha256:e731d1c17be0459197f82c0ffe5922a848f40c78 0.0s
=> [internal] load build context                          0.1s
=> => transferring context: 5.06MB                      0.1s
=> [2/5] RUN mkdir -p /usr/src/app                     0.4s
=> [3/5] WORKDIR /usr/src/app                         0.0s
=> [4/5] COPY . .                                      0.1s
=> [5/5] RUN npm install                             12.4s
=> exporting to image                                0.8s
=> => exporting layers                               0.8s
=> => writing image sha256:49d6eca2f61c2a943836e8275a91c99dfe5f 0.0s
=> => naming to docker.io/library/customer           0.0s
```

- a. Change to the customer directory
b. Create a Docker image named “customer” using the Dockerfile

- c. List out the image we just created for checking

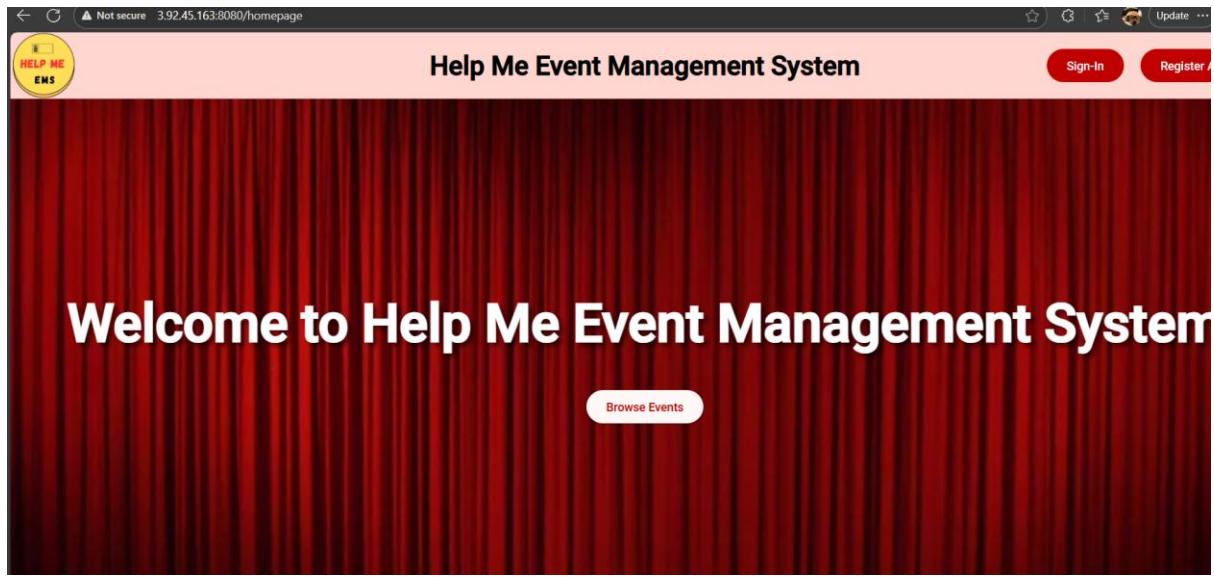
3. Runs the customer Docker image as a container named **customer_1**. It maps port 8080 from the container to the host and sets environment variables for DynamoDB (APP_DB_HOST) so the microservice can connect to our **local DynamoDB** instance at port 8000 from MonolithicAppServer.

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

```
voclabs:~/environment/microservices/customer (dev) $ docker run --name customer_1 -p 8080:8080 -e APP_DB_HOST="http://44.222.237.4:8000" customer
> aws-microservices@1.0.0 start
> node index.js

Server is running on port 8080
```

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



Task 3: Create Docker Image for Employee Microservice

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

A screenshot of a terminal window titled "Dockerfile" showing the contents of a Dockerfile. The file contains the following code:

```
FROM node:24-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY .
RUN npm install
EXPOSE 8081
CMD ["npm", "run", "start"]
```

The terminal also shows a file tree on the left with directories like "MicroservicesIDE", "database", "microservices", "customer", "employee", "models", "website", "app.js", and "Dockerfile".

- 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
      => [internal] load build definition from Dockerfile
      => => transferring dockerfile: 176B
docker:default
          0.0s
          0.0s
```

- 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
```

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

The screenshot shows a web browser window with the following details:

- Address Bar:** Shows the URL `3.92.45.163:8081/admin/register-organizer`.
- Header:** The title bar reads "Help Me Event Management System".
- Left Sidebar:** A dark red sidebar with two buttons: "Register Event Organizer" and "Generate Report".
- Main Content:** The main area is titled "Register Event Organizer". It contains four input fields:
 - "User Name*": An input field with a placeholder.
 - "Email*": An input field with a placeholder.
 - "Phone Number (optional)": An input field with a placeholder.
 - "Organization Name*": An input field with a placeholder.
- Right Side:** A user profile picture and a "Log Out" button.

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

```
FROM node:24-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

```
const express = require('express');
const cors = require('cors');
const path = require('node:path');

const app = require('./app');

const main = express();
main.use(cors());

// api routes
main.use("/admin/api/", app);

// client routes
main.use('/admin/', express.static(path.join(__dirname, './website')));

main.get(('/:.*'), (req, res) => {
    res.sendFile(path.join(__dirname, './website/index.html'));
});

main.listen(8080, () => {
    console.log('Server is running on port 8080');
});
```

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

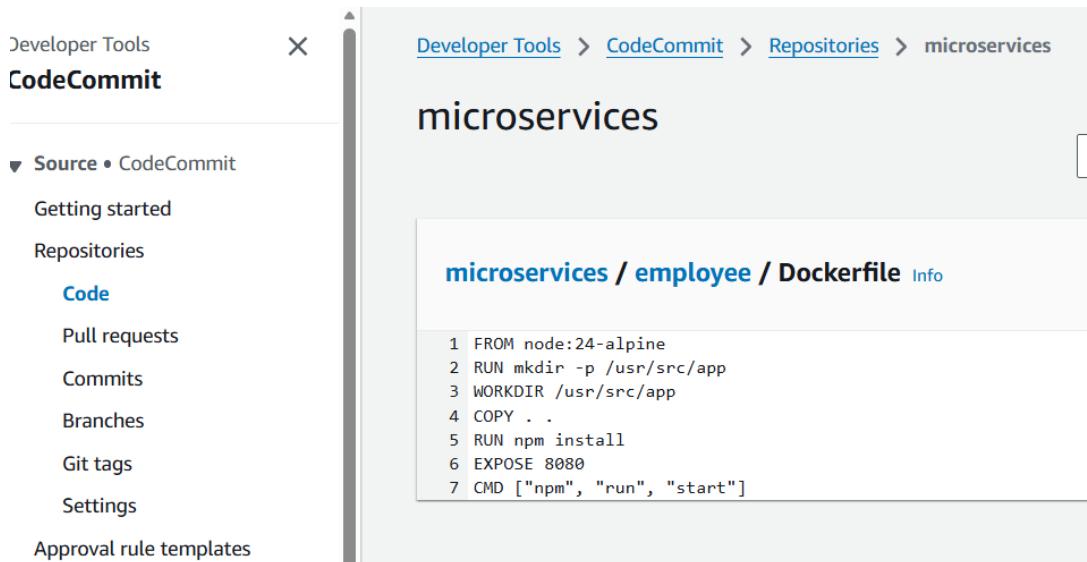
```
voclabs:~/environment/microservices/employee (dev) $ docker rm -f employee_1
employee_1
voclabs:~/environment/microservices/employee (dev) $ docker build --tag employee .
[*] Building 13.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 176B
=> [internal] load metadata for docker.io/library/node:24-alpine
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:24-alpine@sha256:2867d550cf9d8bb50059a0fff528741f11a84d985c732e60e19e8e75c7239c43
=> [internal] load build context
=> => transferring context: 23.78kB
```

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

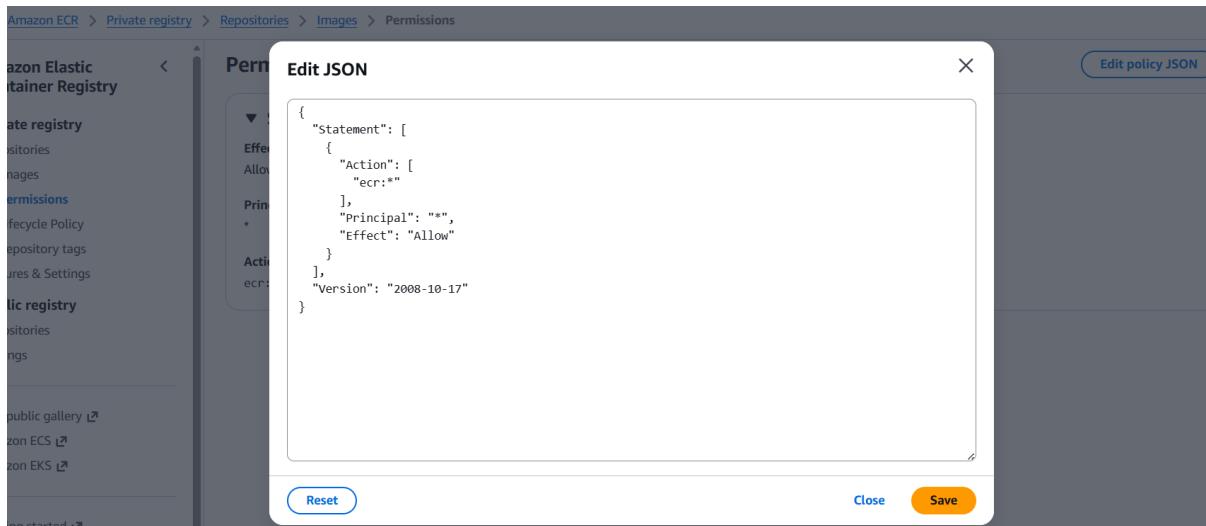
1. Connect and login to ECR service

```
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
```

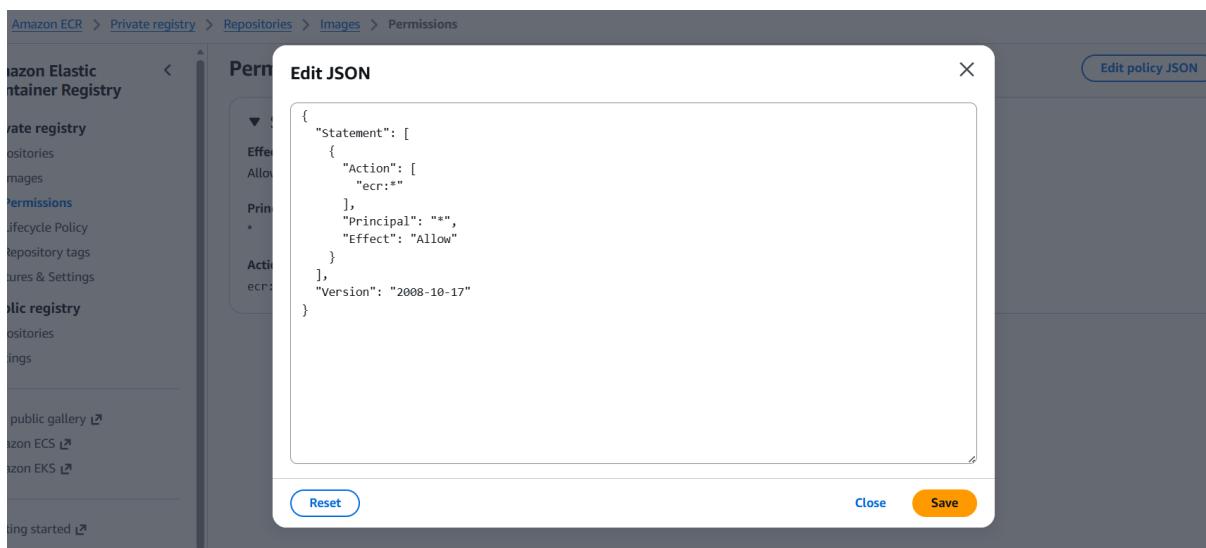
2. Create 2 private ECR repositories for each microservice we have

```
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

```

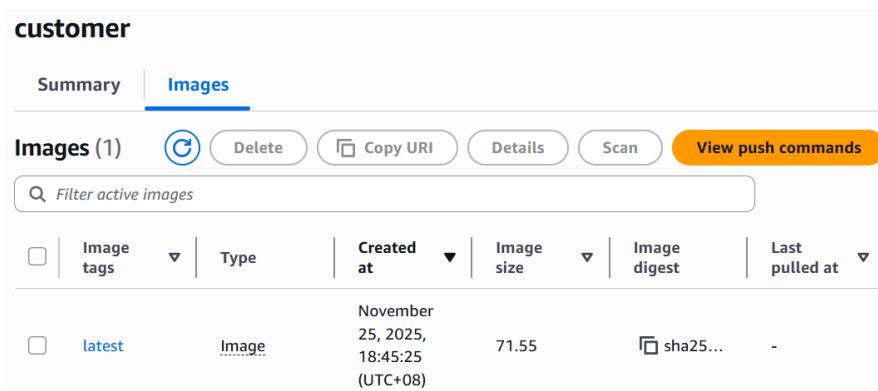
6. Verify the two images exist and tagged correctly

```
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
84aa00592fa: Pushed
5f70bf18a086: Pushed
54fc9e9e9165: Pushed
2497eefc5264: Pushed
d3b1ea8fff6e6: 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
d3b1ea8fff6e6: 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
<input type="text"/> Filter active images						
	Image tags	Type	Created at	Image size	Image digest	Last pulled at
<input type="checkbox"/>	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**:

Amazon Elastic Container Service > Create cluster

Cluster configuration

Cluster name

Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

▼ Service Connect defaults - *optional*

Default namespace | Info
Select or type the namespace name to specify a group of services that make up your application. You can overwrite this value at the service level.

Select a namespace Create a new namespace ↗

▼ Infrastructure - *advanced* Info
Configure the manner of obtaining compute resources that will be used to host your application.

Select a method of obtaining compute capacity
Your cluster is automatically configured for AWS Fargate (serverless), but you may choose to add Amazon EC2 instances (servers).

Fargate only
Serverless - you don't think about creating or managing servers. Great for most common workloads.

Fargate and Managed Instances
Managed instances - Amazon ECS will manage patching and scaling on your behalf while giving you configurability about the types of instances. Great for more advanced workloads.

Fargate a Self-managed scaled proj

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.

```
vocabs:~/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"
    }
}

vocabs:~/environment/microservices (dev) $ mkdir ~/environment/deployment
vocabs:~/environment/microservices (dev) $ cd ~/environment/deployment
vocabs:~/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/
vocabs:~/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:

```
taskdef-customer.json x +  
1 {  
2     "containerDefinitions": [  
3         {  
4             "name": "customer",  
5             "image": "customer",  
6             "environment": [  
7                 {  
8                     "name": "APP_DB_HOST",  
9                     "value": "http://10.16.10.66:8000"  
10                }  
11            ],  
12            "essential": true,  
13            "portMappings": [  
14                {  
15                    "hostPort": 8080,  
16                    "protocol": "tcp",  
17                    "containerPort": 8080  
18                }  
19            ],  
20            "logConfiguration": {  
21                "logDriver": "awslogs",  
22                "options": {  
23                    "awslogs-create-group": "true",  
24                    "awslogs-group": "awslogs-capstone",  
25                    "awslogs-region": "us-east-1",  
26                    "awslogs-stream-prefix": "awslogs-capstone"  
27                }  
28            }  
29        }  
30    ],  
31    "requiresCompatibilities": [  
32        "FARGATE"  
33    ],  
34    "networkMode": "awsvpc",  
35    "cpu": "512",  
36    "memory": "1024",  
37    "executionRoleArn": "arn:aws:iam::634943294831:role/PipelineRole",  
38    "family": "customer-microservice"  
39 }
```

Local DynamoDB endpoint

Account ID

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.

```
vocabs:~/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/customer-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.

The screenshot shows the AWS Lambda Task Definitions page. The title is "customer-microservice (9)" with an "info" link. Below the title, there are buttons for "Last updated" (November 25, 2025, 19:20 (UTC+8:00)), "Deploy", "Actions", and "Create new revision". A search bar says "Filter task definition revisions by value". A dropdown menu for "Filter status" shows "Active". Below these are two filter rows: "Task definition: revision" with a dropdown and "Status" with a dropdown showing "Active".

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:

```
taskdef-employee.json x +  
1 {  
2     "containerDefinitions": [  
3         {  
4             "name": "employee",  
5             "image": "employee",  
6             "environment": [  
7                 {  
8                     "name": "APP_DB_HOST",  
9                     "value": "http://10.16.10.66:8000"  
10                }  
11            ],  
12            "essential": true,  
13            "portMappings": [  
14                {  
15                    "hostPort": 8080,  
16                    "protocol": "tcp",  
17                    "containerPort": 8080  
18                }  
19            ],  
20            "logConfiguration": {  
21                "logDriver": "awslogs",  
22                "options": {  
23                    "awslogs-create-group": "true",  
24                    "awslogs-group": "awslogs-capstone",  
25                    "awslogs-region": "us-east-1",  
26                    "awslogs-stream-prefix": "awslogs-capstone"  
27                }  
28            }  
29        },  
30    ],  
31    "requiresCompatibilities": [  
32        "FARGATE"  
33    ],  
34    "networkMode": "awsvpc",  
35    "cpu": "512",  
36    "memory": "1024",  
37    "executionRoleArn": "arn:aws:iam::634943294831:role/PipelineRole",  
38    "family": "employee-microservice"  
39}
```

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

```
vclabs:~/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.

employee-microservice (14) Info

Last updated
November 25, 2025, 19:24 (UTC+8:00)

Deploy ▾ **Actions** ▾ **Create new revision** ▾

Filter task definition revisions by value

Filter status
Active

<input type="checkbox"/> Task definition: revision	Status
<input type="checkbox"/> employee-microservice:14	<input checked="" type="checkbox"/> Active

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:

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "customer"
          ContainerPort: 8080
```

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

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "employee"
          ContainerPort: 8080
```

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

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

```
taskdef-customer.json taskdef-employee.json +  
1 {  
2     "containerDefinitions": [  
3         {  
4             "name": "customer",  
5             "image": "<IMAGE1_NAME>",  
6             "environment": [  
7                 {
```

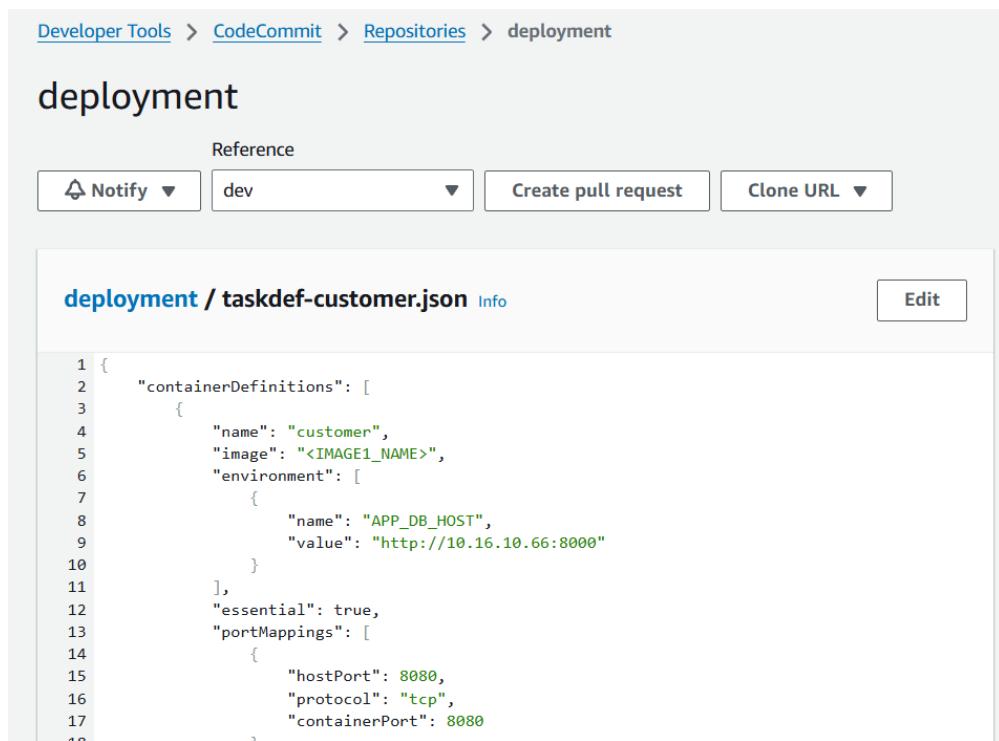
```
taskdef-customer.json taskdef-employee.json +  
1 {  
2     "containerDefinitions": [  
3         {  
4             "name": "employee",  
5             "image": "<IMAGE1_NAME>",  
6             "environment": [  
7                 {
```

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 files"  
[dev (root-commit) e0b680f] Added task def and appspec files  
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



The screenshot shows the AWS CodeCommit interface for the 'deployment' repository. The path 'Developer Tools > CodeCommit > Repositories > deployment' is visible at the top. The repository name 'deployment' is displayed prominently. Below the repository name are several buttons: 'Notify' (with a dropdown arrow), a dropdown menu set to 'dev', 'Create pull request', and 'Clone URL' (with a dropdown arrow). A 'Reference' link is also present. The main content area displays the file 'taskdef-customer.json' with the following JSON code:

```
1 {
2     "containerDefinitions": [
3         {
4             "name": "customer",
5             "image": "<IMAGE1_NAME>",
6             "environment": [
7                 {
8                     "name": "APP_DB_HOST",
9                     "value": "http://10.16.10.66:8000"
10                }
11            ],
12            "essential": true,
13            "portMappings": [
14                {
15                    "hostPort": 8080,
16                    "protocol": "tcp",
17                    "containerPort": 8080
18                }
19            ]
20        }
21    ]
22}
```

An 'Edit' button is located in the top right corner of the code editor.

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 :

EC2 > Target groups > Create target group

Step 1
 Create target group
 Step 2 - recommended
 Register targets
 Step 3
 Review and create

Create target group

A target group can be made up of one or more targets. Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Settings - immutable

Choose a target type and the load balancer and listener will route traffic to your target. These settings can't be modified after target group creation.

Target type

Indicate what resource type you want to target. Only the selected resource type can be registered to this target group.

Instances
Supports load balancing to instances in a VPC. Integrate with Auto Scaling Groups or ECS services for automatic management.
Suitable for: **ALB** **NLB** **GWL**

IP addresses
Supports load balancing to VPC and on-premises resources. Facilitates routing to IP addresses and network interfaces on the same instance. Supports IPv6 targets.
Suitable for: **ALB** **NLB** **GWL**

Lambda function
Supports load balancing to a single Lambda function. ALB required as traffic source.
Suitable for: **ALB**

Application Load Balancer
Allows use of static IP addresses and PrivateLink with an Application Load Balancer. NLB required as traffic source.
Suitable for: **NLB**

Target group name

Name must be unique per Region per AWS account.

customer-tg-one

Accepts: a-z, A-Z, 0-9, and hyphen (-). Can't begin or end with hyphen. 1-32 total characters; Count: 15/32

Protocol

Protocol for communication between the load balancer and targets.

Port

Port number where targets receive traffic. Can be overridden for individual targets during registration.

HTTP **8080**
1-65535

IP address type

Only targets with the indicated IP address type can be registered to this target group.

IPv4
 IPv6

VPC

Select the VPC that hosts the load balancer. Only VPCs that support the IP address type selected above are available in this list. On the [Register targets](#) page, you can register IP addresses from this VPC, or from private IP addresses located outside of this load balancer's VPC (such as a peered VPC, EC2-Classic, or on-premises targets that are reachable over Direct Connect or VPN).

vpc-07641493d4ed4eeae (LabVPC)
10.16.0.0/16

Create VPC

Protocol version

HTTP1
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.

HTTP2
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.

gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.

Health checks

The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.

Health check protocol

HTTP

Health check path

Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.

/

Up to 1024 characters allowed.

Advanced health check settings

Target optimizer - optional [Info](#)

Use a target control port when the target has a strict concurrency limit.

Target control port

The port on which the target communicates its capacity. This value can't be modified after target group creation.

Enter target control port

Step 1
 Create target group
 Step 2 - recommended
 Register targets
 Step 3
 Review and create

Register targets - recommended

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

IP addresses

Step 1: Choose a network
 You can add IP addresses from the VPC selected for your target group or from outside the VPC. Note that you can assemble a mix of targets from multiple network sources by returning to this step and choosing another network.

Network

LabVPC
 vpc-07fe4149354ed4eeae
 IPv4 VPC CIDR: 10.16.0.0/16

Step 2: Specify IPs and define ports
 You can manually enter IP addresses from the selected network.

Enter an IPv4 address from a VPC subnet.
 10.16.0.1

 You can add up to 4 more IP addresses.

Ports
 Ports for routing to this target.
 8080
 1-65535 (separate multiple ports with commas)

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:

Target groups (4) [Info](#) | [What's new?](#)

<input type="checkbox"/>	Name	ARN	Port	Protocol	Target type
<input type="checkbox"/>	employee-tg-two	arn:aws:elasticloadbalancin...	8080	HTTP	IP
<input type="checkbox"/>	employee-tg-one	arn:aws:elasticloadbalancin...	8080	HTTP	IP
<input type="checkbox"/>	customer-tg-one	arn:aws:elasticloadbalancin...	8080	HTTP	IP
<input type="checkbox"/>	customer-tg-two	arn:aws:elasticloadbalancin...	8080	HTTP	IP

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:

Security Groups > Create security group

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name Info
microservices-sg
Name cannot be edited after creation.

Description Info
microservices sg

VPC Info
vpc-07641493d4ed4eeae (LabVPC)

Inbound rules Info

Type	Protocol	Port range	Source	Description - optional
Custom TCP	TCP	80	Anywhere	0.0.0.0/0
Custom TCP	TCP	8080	Anywhere	0.0.0.0/0

Add rule

Outbound rules Info

Type	Protocol	Port range	Destination	Description - optional
All traffic	All	All	Custom	0.0.0.0/0

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

The screenshot shows the 'Create Application Load Balancer' wizard. Step 1: Set Load Balancer Name to 'microservicesLB'. Step 2: Choose 'Internet-facing' Scheme. Step 3: Select 'IPv4' for Load balancer IP address type. Step 4: Under Network mapping, choose VPC 'vpc-07641493d4ed4eeae (LabVPC)'. Step 5: Under IP pools, enable 'Use IPAM pool for public IPv4 addresses'. Step 6: Under Availability Zones and subnets, select 'us-east-1a (use1-az1)' and 'us-east-1b (use1-az2)'. Step 7: Under Security groups, select 'microservices-sg'. The final step is to review and create the load balancer.

Load balancer name
Name must be unique within your AWS account and can't be changed after the load balancer is created.

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Scheme | **Info**
Scheme can't be changed after the load balancer is created.

Internet-facing

- Serves internet-facing traffic.
- Has public IP addresses.
- DNS name resolves to public IPs.
- Requires a public subnet.

Internal

- Serves internal traffic.
- Has private IP addresses.
- DNS name resolves to private IPs.
- Compatible with the **IPv4** and **Dualstack** IP address types.

Load balancer IP address type | **Info**
Select the front-end IP address type to assign to the load balancer. The VPC and subnets mapped to this load balancer must include the selected IP address types. Public IPv4 addresses have an additional cost.

IPv4
Includes only IPv4 addresses.

Dualstack
Includes IPv4 and IPv6 addresses.

Dualstack without public IPv4
Includes a public IPv6 address, and private IPv4 and IPv6 addresses. Compatible with **internet-facing** load balancers only.

Network mapping | **Info**
The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC | **Info**
The load balancer will exist and scale within the selected VPC. The selected VPC is also where the load balancer targets must be hosted unless routing to Lambda or on-premises targets, or if using VPC peering. To confirm the VPC for your targets, view [target groups](#).

vpc-07641493d4ed4eeae (LabVPC)
10.16.0.0/16

Create VPC

IP pools | **Info**
You can optionally choose to configure an IPAM pool as the preferred source for your load balancer's IP addresses. Create or view [Pools](#) in the [Amazon VPC IP Address Manager console](#).

Use IPAM pool for public IPv4 addresses
The IPAM pool you choose will be the preferred source of public IPv4 addresses. If the pool is depleted, IPv4 addresses will be assigned by AWS.

Availability Zones and subnets | **Info**
Select at least two Availability Zones and a subnet for each zone. A load balancer node will be placed in each selected zone and will automatically scale in response to traffic. The load balancer routes traffic to targets in the selected Availability Zones only.

us-east-1a (use1-az1)
Subnet
Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.
subnet-0cad3cb1e87ed6ed
IPv4 subnet CIDR: 10.16.10.0/24

us-east-1b (use1-az2)
Subnet
Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.
subnet-0f82feaaa5762c5dc
IPv4 subnet CIDR: 10.16.20.0/24

Public Subnet1

Public Subnet2

Security groups | **Info**
A security group is a set of firewall rules that control the traffic to your load balancer. Select an existing security group, or you can [create a new security group](#).

Security groups

Select up to 5 security groups

microservices-sg
sg-0f3ddfe690e419106 VPC: vpc-07641493d4ed4eeae

Listeners and routing Info

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the traffic is handled.

▼ Listener HTTP:80

Protocol

HTTP

Port

80

1-65535

Default action | Info

The default action is used if no other rules apply. Choose the default action for traffic on this listener.

Routing action

Forward to target groups

Redirect to URL

Forward to target group | Info

Choose a target group and specify routing weight or [create target group](#).

Target group

customer-tg-two

Target type: IP, IPv4 | Target stickiness: Off

HTTP



Weight

1

0-999

+ Add target group

You can add up to 4 more target groups.

▼ Listener HTTP:8080

Protocol

HTTP

Port

8080

1-65535

Default action | Info

The default action is used if no other rules apply. Choose the default action for traffic on this listener.

Routing action

Forward to target groups

Redirect to URL

Rewrite

Forward to target group | Info

Choose a target group and specify routing weight or [create target group](#).

Target group

customer-tg-one

Target type: IP, IPv4 | Target stickiness: Off

HTTP



Weight

1

Percent

100%

0-999

+ Add target group

You can add up to 4 more target groups.

Target group stickiness | Info

Enables the load balancer to bind a user's session to a specific target group. To use stickiness the client must support cookies. If you want to bind a user's session to a specific target, turn on the 'Turn on target group stickiness' checkbox.

Turn on target group stickiness

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:

[HTTP:80 listener](#) > [Add rule](#)

Conditions (1 value) Info

Define 1-5 condition values. Additional conditions can't be added once the limit is reached.

Path (value) = /admin/*

Match pattern type

- Value matching**
Match with glob syntax, using `*` and `?` as wildcards.
- Regex matching**
Match with regex syntax.

Path condition value
Case sensitive.
= `/admin/*`
Valid characters are a-z, A-Z, 0-9 and special characters. Path must be 1-128 characters.

[+ Add OR condition value](#)

[Add condition ▾](#)
You can add up to 4 more condition values for this rule.

Actions Info

Requests matching all rule conditions route according to the rule actions.

Routing action

- Forward to target groups**
- Redirect to URL**
- Return fixed response**

Forward to target group Info
Choose a target group and specify routing weight or [create target group](#).

Target group

employee-tg-two	HTTP ▾	
Target type: IP, IPv4 Target stickiness: Off		

Weight	Percent
1	100%
0-999	

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**

Set rule priority Info

Each rule has a priority. The default rule is evaluated last. You can change the priority of a non-default rule at any time. You can't change the priority of the default rule.

► Listener details: HTTP:80

Listener rules (2) Info

Rule limits Reset priorities

Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.

Priority	Name tag	Conditions (If)	Transforms
1	-	Path (value) = /admin/*	-
Last (default)	Default	If no other rule applies	-

Priority value must be 1-50,000.

4. Verify the rule is added to 80 listener:

microservicesLB > HTTP:80 listener

▼ Details
A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.

Protocol:Port	Load balancer	Default actions
HTTP:80	microservicesLB	<ul style="list-style-type: none">Forward to target group customer-tg-two: 1 (100%) Target group stickiness: Off

Listener ARN
 arn:aws:elasticloadbalancing:us-east-1:045285622849:listener/app/microservicesLB/7157bd0fb5a095c9/3781eae6c1a1692f

Rules Attributes Tags

Listener rules (2) Info

Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.

Priority	Name tag	Conditions (If)	Transforms	Actions (Then)
1	-	Path (value) = /admin/*	-	<ul style="list-style-type: none">Forward to target group employee-tg-two: 1 (100%) Target group stickiness: Off
Last (default)	Default	If no other rule applies	-	<ul style="list-style-type: none">Forward to target group customer-tg-two: 1 (100%) Target group stickiness: Off

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

The screenshot shows the 'Add rule' configuration for a microservicesLB listener. It is divided into two main sections: 'Conditions' and 'Actions'.

Conditions (1 value)

Path (value) = /admin/*

Match pattern type

- Value matching: Match with glob syntax, using `*` and `?` as wildcards.
- Regex matching: Match with regex syntax.

Path condition value
Case sensitive.
= `/admin/*`

Valid characters are a-z, A-Z, 0-9 and special characters. Path must be 1-128 characters.

Add OR condition value

Add condition ▾
You can add up to 4 more condition values for this rule.

Actions

Requests matching all rule conditions route according to the rule actions.

Routing action

- Forward to target groups
- Redirect to URL
- Return fixed response

Forward to target group | Info
Choose a target group and specify routing weight or [create target group](#).

Target group

employee-tg-one	HTTP	
Target type: IP, IPv4 Target stickiness: Off		

Weight
1
0-999

Percent
100%

+ Add target group
You can add up to 4 more target groups.

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**.

Set rule priority Info

Each rule has a priority. The default rule is evaluated last. You can change the priority of a non-default rule at any time. You can't change the priority of the default rule.

► Listener details: HTTP:8080

Listener rules (2) Info

Rule limits



Reset priorities

Add gap between priorities ▾

Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.

Priority	Name tag	Conditions (If)
1	-	Path (value) = /admin/*
Last (default)	Default	If no other rule applies

6. Verify the rule is added to 8080 listener:

HTTP:8080 Info

▼ Details

A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to

Protocol:Port

HTTP:8080

Load balancer

microservicesLB

Default actions

- Forward to target group
[customer-tg-one](#): 1 (100%)

Target group stickiness: Off

Listener ARN

[arn:aws:elasticloadbalancing:us-east-1:045285622849:listener/app/microservicesLB/7157bd0fb5a095c9/1f9cf3bc9e59cbf1](#)

Rules Attributes Tags

Listener rules (2) Info

Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.

Rule limits



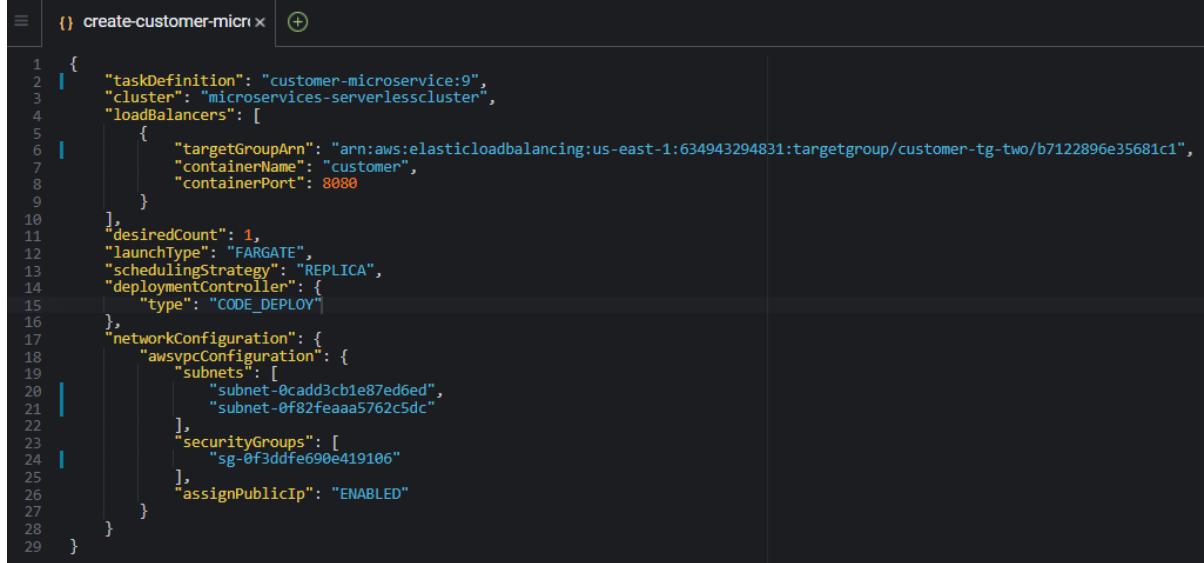
Filter rules

Priority	Name tag	Conditions (If)	Transforms	Actions (Then)
1	-	Path (value) = /admin/*	-	<ul style="list-style-type: none">Forward to target group employee-tg-one: 1 (100%)
Last (default)	Default	If no other rule applies	-	<ul style="list-style-type: none">Forward to target group customer-tg-one: 1 (100%)

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.



```
1 {  
2     "taskDefinition": "customer-microservice:9",  
3     "cluster": "microservices-serverlesscluster",  
4     "loadBalancers": [  
5         {  
6             "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:634943294831:targetgroup/customer-tg-two/b7122896e35681c1",  
7             "containerName": "customer",  
8             "containerPort": 8080  
9         }  
10    ],  
11    "desiredCount": 1,  
12    "launchType": "FARGATE",  
13    "schedulingStrategy": "REPLICA",  
14    "deploymentController": {  
15        "type": "CODE_DEPLOY"  
16    },  
17    "networkConfiguration": {  
18        "awsvpcConfiguration": {  
19            "subnets": [  
20                "subnet-0cadd3cb1e87ed6ed",  
21                "subnet-0f82feaaa5762c5dc"  
22            ],  
23            "securityGroups": [  
24                "sg-0f3ddfe690e419106"  
25            ],  
26            "assignPublicIp": "ENABLED"  
27        }  
28    }  
29}
```

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`

```
vclabs:~/environment/deployment (dev) $ aws ecs create-service --service-name customer-microservice --cli-input-json file://create-customer-microservice-tg-two.json  
e-tg-two.json  
{  
  "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/b7122896e35681c1",  
        "containerName": "customer",  
        "containerPort": 8080  
      }  
    ]  
  }  
}
```

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.

```

1  {
2     "taskDefinition": "employee-microservice:14",
3     "cluster": "microservices-serverlesscluster",
4     "loadBalancers": [
5         {
6             "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:634943294831:targetgroup/employee-tg-two/00ac7ca98b782154",
7             "containerName": "employee",
8             "containerPort": 8080
9         }
10    ],
11    "desiredCount": 1,
12    "launchType": "FARGATE",
13    "schedulingStrategy": "REPLICA",
14    "deploymentController": {
15        "type": "CODE_DEPLOY"
16    },
17    "networkConfiguration": {
18        "awsvpcConfiguration": {
19            "subnets": [
20                "subnet-0cadd3cb1e87ed6ed",
21                "subnet-0f82feaaa5762c5dc"
22            ],
23            "securityGroups": [
24                "sg-0f3ddfe690e419106"
25            ],
26            "assignPublicIp": "ENABLED"
27        }
28    }
29 }

```

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.

```

vclabs:~/environment/deployment (dev) $ aws ecs create-service --service-name employee-microservice --cli-input-json file://create-employee-microservice-tg-two.json
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:634943294831:service/microservices-serverlesscluster/employee-microservice",
        "serviceName": "employee-microservice",
        "clusterArn": "arn:aws:ecs:us-east-1:634943294831:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:634943294831:targetgroup/employee-tg-two/00ac7ca98b782154",
                "containerName": "employee",
                "containerPort": 8080
            }
        ],
        "desiredCount": 1,
        "launchType": "FARGATE",
        "schedulingStrategy": "REPLICA",
        "deploymentController": {
            "type": "CODE_DEPLOY"
        },
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-0cadd3cb1e87ed6ed",
                    "subnet-0f82feaaa5762c5dc"
                ],
                "securityGroups": [
                    "sg-0f3ddfe690e419106"
                ],
                "assignPublicIp": "ENABLED"
            }
        }
    }
}

```

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

Developer Tools > CodeDeploy > Applications > Create application

Create application

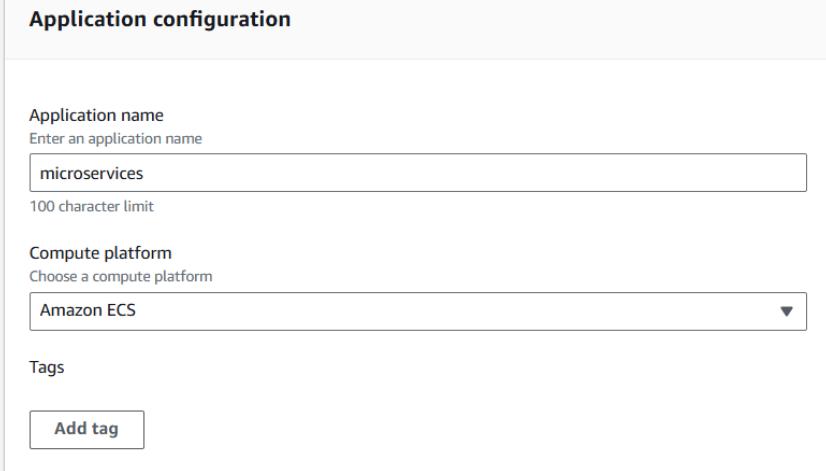
Application configuration

Application name
Enter an application name

100 character limit

Compute platform
Choose a compute platform

Tags



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:

Developer Tools > CodeDeploy > Applications > microservices > Create deployment group

Create deployment group

Application

Application
microservices
Compute type
Amazon ECS

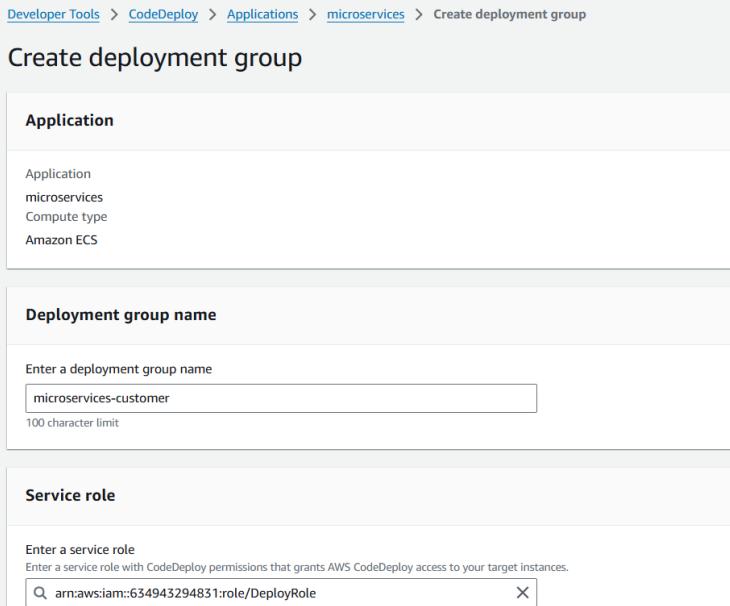
Deployment group name

Enter a deployment group name

100 character limit

Service role

Enter a service role
Enter a service role with CodeDeploy permissions that grants AWS CodeDeploy access to your target instances.



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:

Developer Tools > CodeDeploy > Applications > microservices > Create deployment group

Create deployment group

Application

Application
microservices
Compute type
Amazon ECS

Deployment group name

Enter a deployment group name
microservices-employee
100 character limit

Service role

Enter a service role
Enter a service role with CodeDeploy permissions that grants AWS CodeDeploy access to your target instances.
arn:aws:iam::634943294831:role/DeployRole

Environment configuration

Choose an ECS cluster name
microservices-serverlesscluster

Choose an ECS service name
employee-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
employee-tg-two

Target group 2 name
employee-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 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	Hours	Minutes
0	0	5

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:

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Choose creation option Info

Step 1 of 7

Category

Deployment Continuous Integration Automation

Build custom pipeline

Cancel Next

Choose pipeline settings [Info](#)

Step 2 of 7

Pipeline settings

Pipeline name

Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Execution mode [Info](#)

Choose the execution mode for your pipeline. This determines how the pipeline is run.

- Superseded
- Queued
- Parallel

Service role

New service role
Create a service role in your account

Existing service role
Choose an existing service role from your account

Role ARN



Add source stage [Info](#)

Step 3 of 7

Source

Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.



Repository name

Choose a repository that you have already created where you have pushed your source code.



Branch name

Choose a branch of the repository



Create EventBridge rule to automatically detect source changes

If disabled, follow AWS documentation to create an EventBridge rule for your source. [Learn more](#)

Output artifact format

Choose the output artifact format.

CodePipeline default

AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.

Full clone

AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. [Learn more](#)

Enable automatic retry on stage failure

[Cancel](#)

[Previous](#)

[Next](#)

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

Add deploy stage Info

Step 6 of 7

i 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 X

No more than 100 characters
Defined by: Source

AWS CodeDeploy application name

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

microservices



Create application i

AWS CodeDeploy deployment group

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

mircoservices-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

The screenshot shows the 'Edit' page for the 'update-customer-microservice' pipeline. At the top, there are tabs for 'Developer Tools', 'CodePipeline', 'Pipelines', 'update-customer-microservice', and 'Edit update-customer-micro...'. Below the tabs are three main sections:

- Edit: Pipeline properties**: Shows 'Pipeline type V2' and 'Execution mode QUEUED'. There is a blue 'Edit' button.
- Edit: Variables**: Shows 'Pipeline type V2 required'. It has columns for 'Name', 'Default value', and 'Description'. A blue 'Edit variables' button is at the top right. Below it says 'No variables' and 'No variables defined at the pipeline level in this pipeline.'
- Edit: Git Triggers**: Shows 'No triggers'. It says 'This pipeline does not have source actions that support triggers. Triggers are only supported for source actions that use CodeConnections.' There is a blue 'Edit triggers' button.
- Edit: Source**: This section is currently empty. There is a blue 'Edit stage' button.

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

Edit action

Action name
Choose a name for your action

No more than 100 characters

Action provider

Repository name
Choose an Amazon ECR repository as the source location.
 X

Image tag - optional
Choose the image tag that triggers your pipeline when a change occurs in the image repository.
 X
If an image tag is not selected, defaults to latest

Variable namespace - optional
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. [Learn more](#)

Output artifacts
Choose a name for the output of this action.

No more than 100 characters

- 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:

Edit action

Action name
Choose a name for your action

No more than 100 characters

Action provider

Region

Input artifacts
Choose an input artifact for this action. [Learn more](#)

 Defined by: Source Defined by: Image
No more than 100 characters

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

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

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.

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.

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

Placeholder text in the task definition

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:

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Choose creation option Info

Step 1 of 7

Category

Deployment Continuous Integration Automation
 Build custom pipeline

Cancel **Next**

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Choose pipeline settings Info

Step 2 of 7

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Execution mode Info
Choose the execution mode for your pipeline. This determines how the pipeline is run.
 Superseded
 Queued
 Parallel

Service role
 New service role
Create a service role in your account
 Existing service role
Choose an existing service role from your account

Role ARN

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

AWS CodeCommit ▾

Repository name
Choose a repository that you have already created where you have pushed your source code.

deployment X

Branch name
Choose a branch of the repository

dev X

Create EventBridge rule to automatically detect source changes
If disabled, follow AWS documentation to create an EventBridge rule for your source. [Learn more ↗](#)

Output artifact format
Choose the output artifact format.

CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.

Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. [Learn more ↗](#)

Enable automatic retry on stage failure

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 X
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 X Create application ↗

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

microservices-employee X

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-employee.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-employee.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-employee-microservice → Edit

Developer Tools > CodePipeline > Pipelines > update-employee-microservice > Edit update-employee-microservice

Editing: update-employee-microservice

Edit: Pipeline properties

Pipeline type: V2
 Execution mode: QUEUED

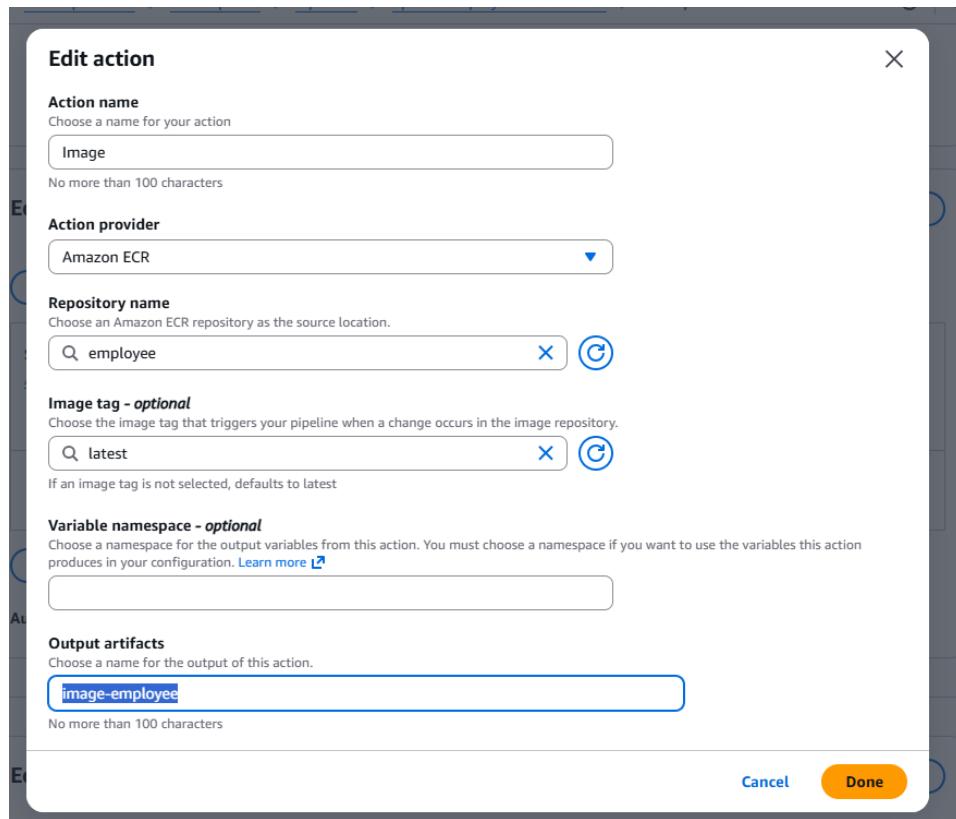
Edit: Variables
 Pipeline type V2 required

Name	Default value	Description
No variables		
No variables defined at the pipeline level in this pipeline.		

Edit: Git Triggers
 No triggers
This pipeline does not have source actions that support triggers. Triggers are only supported for source actions that use CodeConnections.

Edit: Source

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:

Edit action

Action name

Choose a name for your action

Deploy

No more than 100 characters

Action provider

Amazon ECS (Blue/Green)

Region

United States (N. Virginia)

Input artifacts

Choose an input artifact for this action. [Learn more](#)

SourceArtifact X
Defined by: Source

image-employee X
Defined by: Image

No more than 100 characters

AWS CodeDeploy application name

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

microservices

AWS CodeDeploy deployment group

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

microservices-employee

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-employee.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-employee.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 a

Input artifact with image details

image-employee



Placeholder text in the task definition

IMAGE1_NAME

[Remove](#)

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”:

The screenshot shows the AWS CodePipeline console with the pipeline name "update-customer-microservice". The "Release change" button is highlighted in orange. Other buttons include "Edit", "Stop execution", "Create trigger", and "Clone pipeline". Below the buttons are tabs for "Pipeline", "Executions", "Triggers", "Settings", "Tags", and "Stage".

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

The screenshot shows the AWS CodePipeline console displaying the execution details for the "update-customer-microservice" pipeline. It shows two stages: "Source" and "Deploy". Both stages are marked as "All actions succeeded". The "Source" stage includes actions for "AWS CodeCommit" and "Amazon ECR". The "Deploy" stage includes an action for "Amazon ECS (Blue/Green)". The overall status is "90e0c9bb Source: update Image: sha256:451ef7f49697".

The screenshot shows the AWS CodeDeploy console displaying the deployment status for deployment "d-Z69QTV8CG". The deployment consists of five steps: Step 1 (Deploying replacement task set), Step 2 (Test traffic route setup), Step 3 (Rerouting production traffic to replacement task set), Step 4 (Wait), and Step 5 (Terminate original task set). All steps are marked as "Completed" with a green checkmark and labeled "Succeeded". Progress bars indicate 100% completion for each step.

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



Amazon Elastic Container Service > Clusters > microservices-serverlesscluster > Services

Last updated November 25, 2025, 10:31 (UTC+8:00) Actions Create with Express Mod

microservices-serverlesscluster

Cluster overview		CloudWatch monitoring		Registered container instances			
ARN arn:aws:ecs:us-east-1:634943294831:cluster/microservices-serverlesscluster	Status Active		CloudWatch monitoring Default		Registered container instances		
Services		Tasks					
Draining -	Active 2	Pending -	Running 2				
Services	Tasks	Infrastructure	Metrics	Scheduled tasks	Configuration	Event history	Tags

Services (2) Info

Filter services by value		Filter launch type	Filter scheduling strategy	Filter resource management type
customer-microservice	employee-microservice	Any launch type	Any scheduling strategy	Any resource management type
arn:aws:ecs:us-east-1:634943294831:service/microservices-serverlesscluster/customer-microservice	arn:aws:ecs:us-east-1:634943294831:service/microservices-serverlesscluster/employee-microservice	Active	REPLICA	FARGATE
customer-microservice	employee-microservice	Active	REPLICA	FARGATE

Task 7: Test CI/CD Pipeline for Employee Microservices

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

The screenshot shows the AWS CodePipeline console. At the top, there's a navigation bar with 'Developer Tools > CodePipeline > Pipelines > update-employee-microservice'. Below the navigation is the pipeline name 'update-employee-microservice' with a copy icon and gear icon. A row of buttons includes 'Edit', 'Stop execution', 'Create trigger', 'Clone pipeline', and 'Release change' (which is highlighted in orange). Below these buttons are tabs for 'Pipeline', 'Executions', 'Triggers', 'Settings', 'Tags', and 'Stage'. The 'Pipeline' tab is selected.

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

The screenshot shows the AWS CodePipeline console after a release change. The pipeline status is displayed with two green checkmarks and an info icon. Below, the 'Source' stage (b5902e73-955a-406e-9d1c-fda519e13be7) is shown as completed with actions for AWS CodeCommit (18 minutes ago) and Amazon ECR (18 minutes ago). The 'Deploy' stage (b5902e73-955a-406e-9d1c-fda519e13be7) is also completed with an action for Amazon ECS (Blue/Green) (8 minutes ago). The overall status is 'All actions succeeded.'

The screenshot shows the AWS CodeDeploy console for deployment 'd-FXF75J7CG'. The 'Deployment status' section shows five steps: Step 1: Deploying replacement task set (Completed, Succeeded), Step 2: Test traffic route setup (Completed, Succeeded), Step 3: Rerouting production traffic to replacement task set (Completed, Succeeded), Step 4: Wait (Completed, Succeeded), and Step 5: Terminate original task set (Completed, Succeeded). Each step has a progress bar at 100% completion.

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) :

Help Me Event Management System

Register Event Organizer

User Name* : zhengyelan98

Email* :

Phone Number (optional):

Organization Name* :

Amazon Elastic Container Service > Clusters > microservices-serverlesscluster > Services

microservices-serverlesscluster

Last updated: November 25, 2025, 10:31 (UTC+8:00)

Cluster overview			
ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:us-east-1:634943294831:cluster/microservices-serverlesscluster	Active	Default	-

Services				Tasks	
Draining	Active	Pending	Running		
-	2	-	2		

Services (2) info

Service name	ARN	Status	Scheduling strategy	Launch type	Task definition	Deployments and tasks	Last deployment
customer-microservice	arn:aws:ecs:us-east-1:634943294831:task-definition/customer-microservice	Active	REPLICAS	FARGATE	customer-microservice	1/1 Tasks running	Succeeded View
employee-microservice	arn:aws:ecs:us-east-1:634943294831:task-definition/employee-microservice	Active	REPLICAS	FARGATE	employee-microservice	1/1 Tasks running	Succeeded View

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

```
voclabs:~/environment $ cd ~/environment/microservices/customer  
voclabs:~/environment/microservices/customer (dev) $ docker rm -f customer_1  
customer 1
```

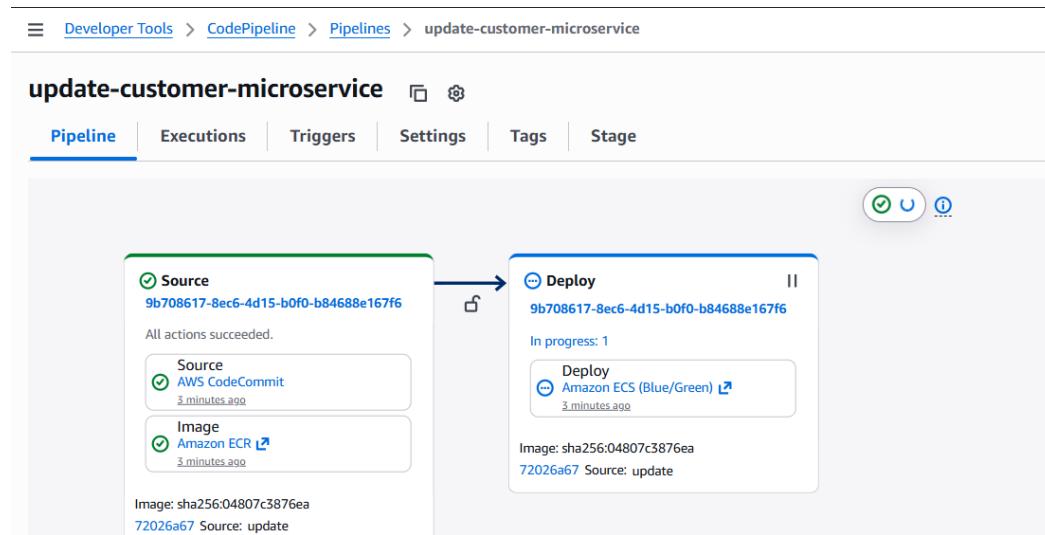
```
voclabs:~/environment/microservices/customer (dev) $ docker build --tag customer .  
[+] Building 15.7s (10/10) FINISHED  
          docker:default  
=> [internal] load build definition from Dockerfile  
              0.0s  
=> => transferring dockerfile: 176B  
              0.0s
```

```
vocabs:~/environment/microservices/customer (dev) $ echo $account_id  
634943294831
```

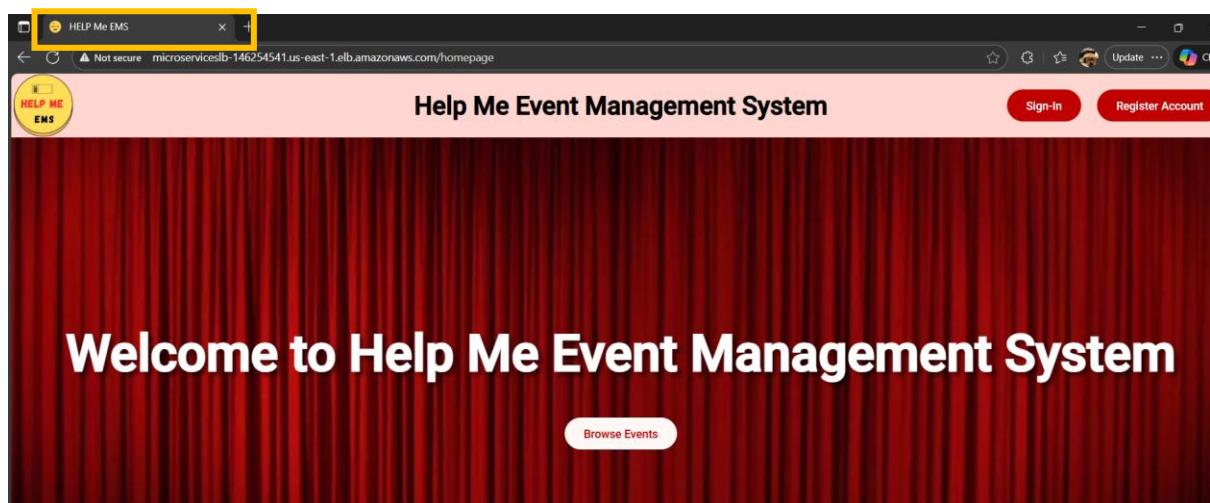
```
vocabs:~/environment/microservices/customer (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vocabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
634943294831.dkr.ecr.us-east-1.amazonaws.com/customer   latest    abc55412e5e2  6 minutes ago  219MB
customer            latest    abc55412e5e2  6 minutes ago  219MB
634943294831.dkr.ecr.us-east-1.amazonaws.com/employee   latest    3293f2b67fa  18 hours ago  220MB
employee            latest    3293f2b67fa  18 hours ago  220MB
634943294831.dkr.ecr.us-east-1.amazonaws.com/customer   <none>   7901f8e6fe  11 hours ago  220MB
vocabs:~/environment/microservices/customer (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
vocabs:~/environment/microservices/customer (dev) $
vocabs:~/environment/microservices/customer (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]
e34cc0d9a78: Pushed
d56f11425b64: Pushed
5f70fb1f18a086: Layer already exists
54fc9e9e9165: Layer already exists
249fecf5264: Layer already exists
d3b1ea8fffee6: Layer already exists
```

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 Sys... X +

Not secure microserviceslb-146254541.us-east-1.elb.amazonaws.com/admin/register-organizer

Help Me Event Management System

HELP ME EMS

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,
        "lastUpdated": "2025-11-25T23:02:00+00:00"
    }
}
```

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

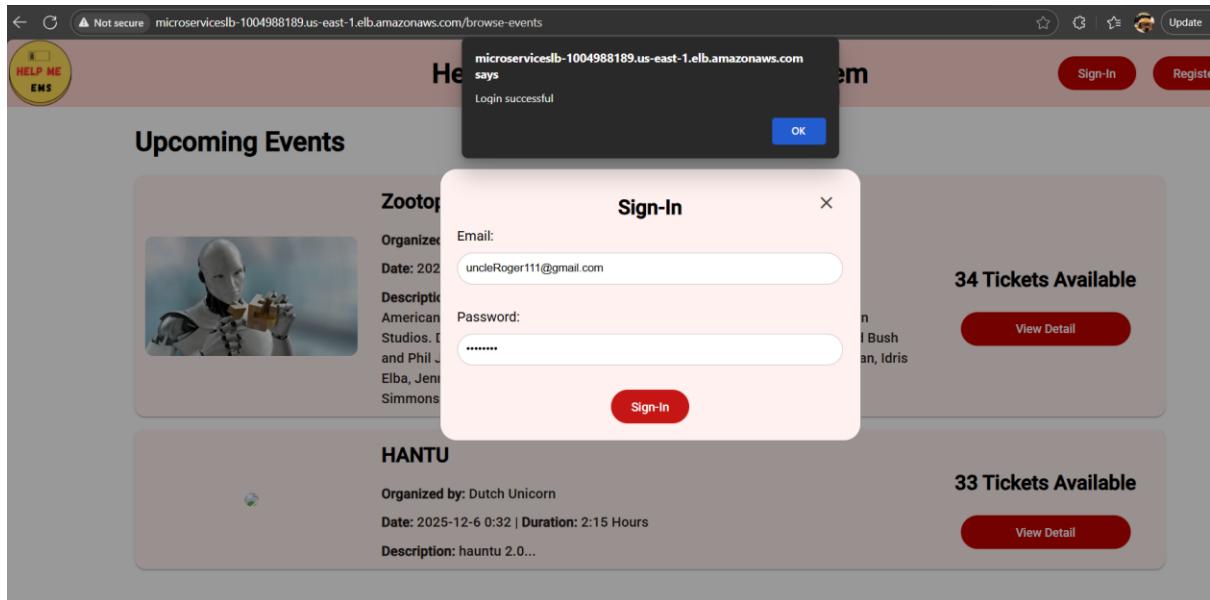
now.

The screenshot shows the AWS ECS Cluster Overview page for the 'microservices-serverlesscluster'. The 'Services' tab is active. There are two services listed:

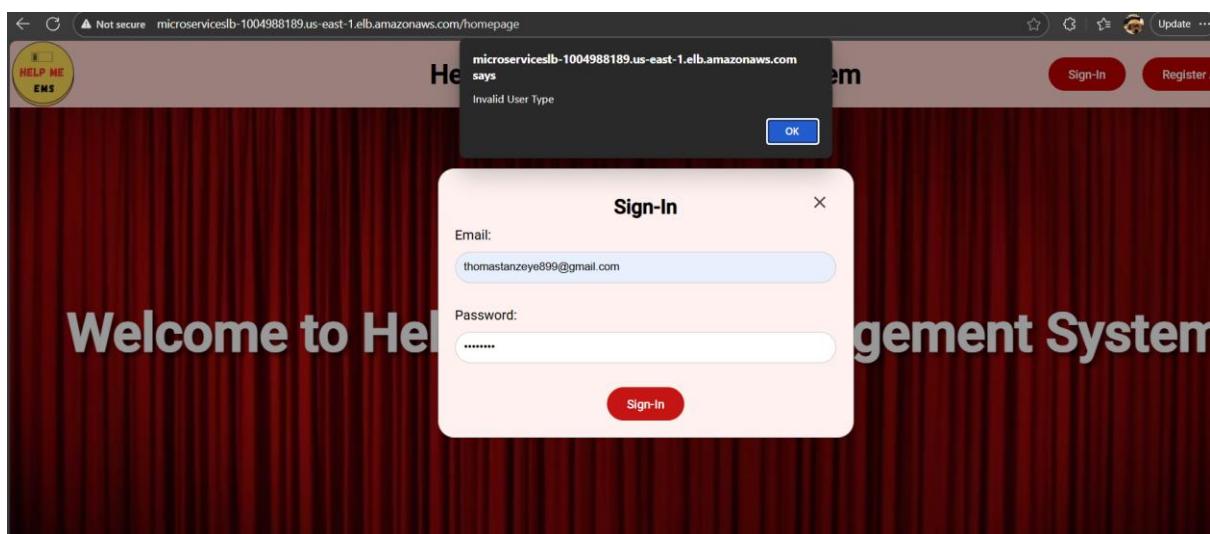
Service Name	ARN	Status	Task Definition	Deployments and Tasks	Last Deployment
customer-microservice	arn:aws:ecs:us-east-1:634943294831:task-definition/customer-microservice	Active	REPLICA	customer... 3/3 Tasks running	Succeeded
employee-microservice	arn:aws:ecs:us-east-1:634943294831:task-definition/employee-microservice	Active	REPLICA	employee... 1/1 Tasks running	Succeeded

Task 10: Validate the Function of Customer Microservice

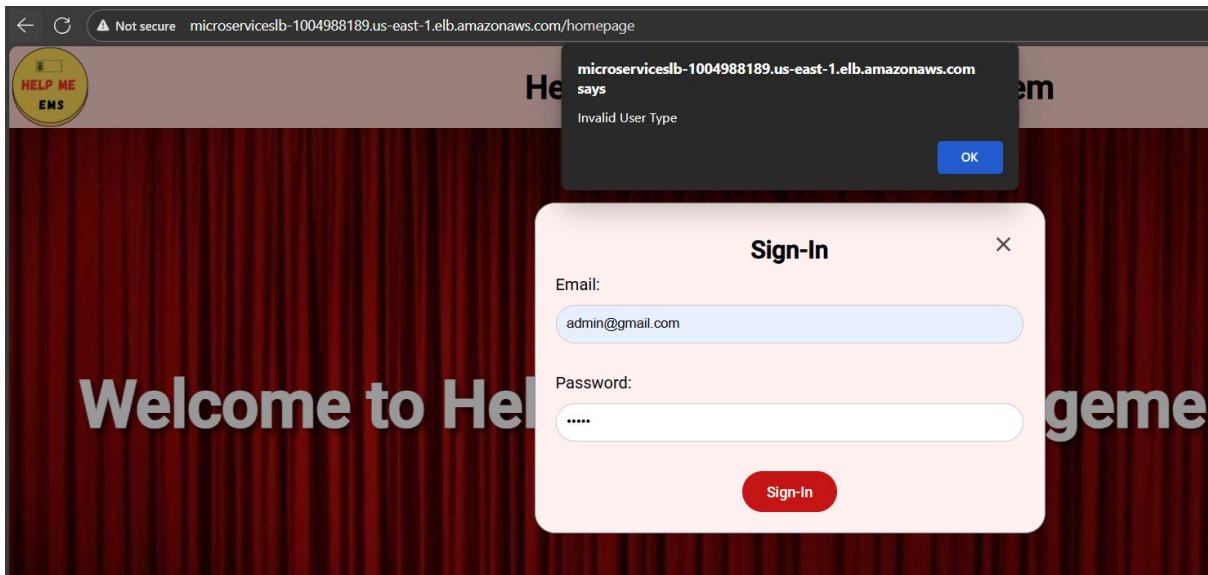
1. In Customer Microservice , login as “attendee” account, will be succeed. Then, they can continue browse event, book event, checkout, and add/remove event to waitlist.



2. In Customer Microservice , login as “event organiser” account, will be failed.



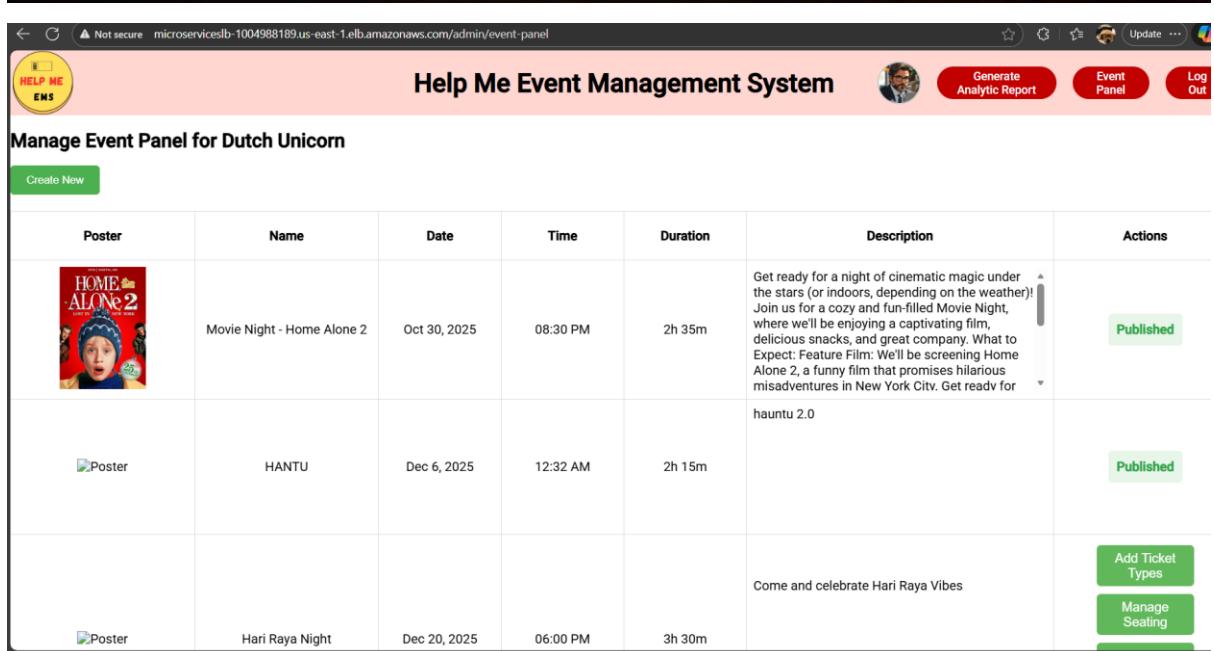
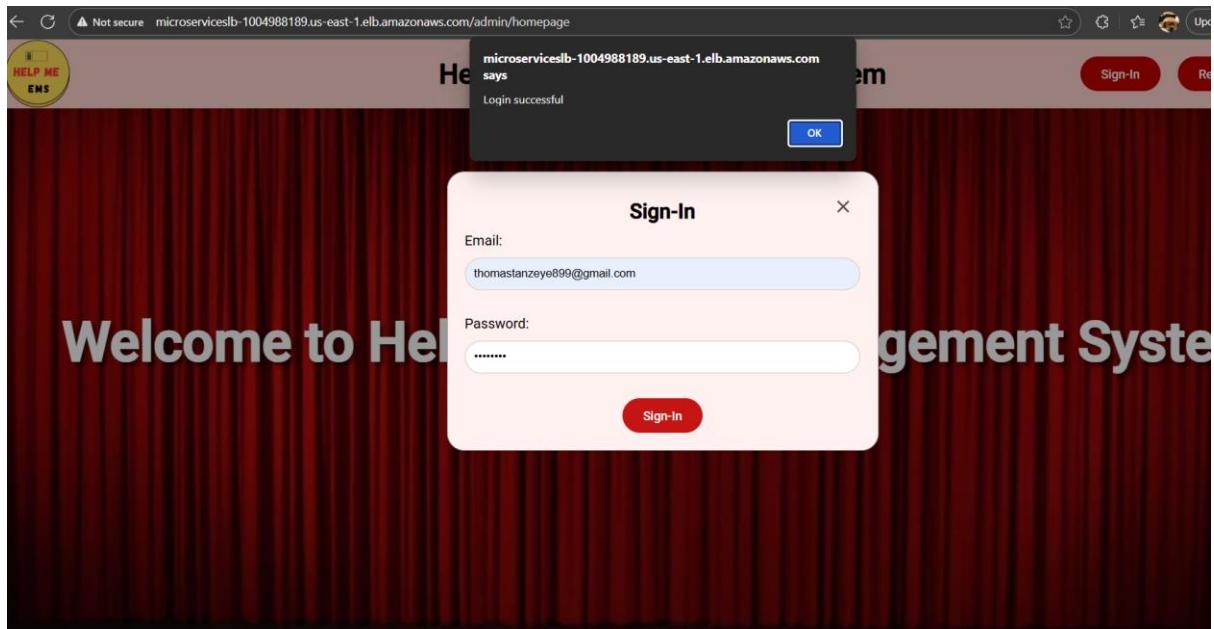
3. In Customer Microservice , login as “**admin**” account, will be failed.



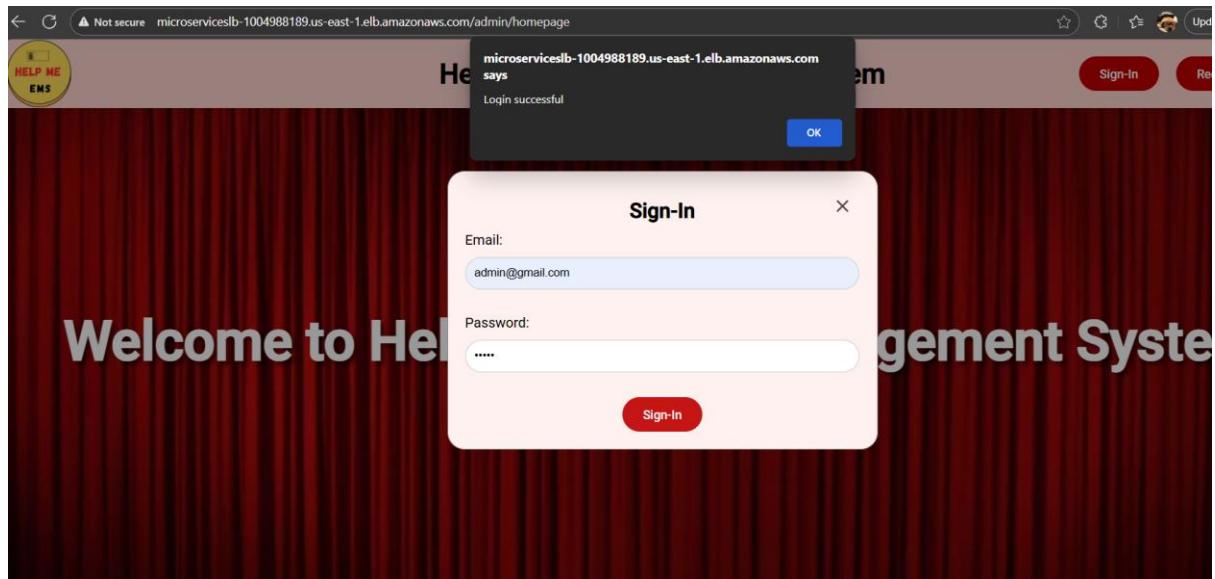
Task 11: Validate the Function of Employee Microservice

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.

A screenshot of a web page titled "Help Me Event Management System". On the left, there's a sidebar with "Register Event Organizer" and "Generate Report" buttons. The main content area has a title "Register Event Organizer". It contains four input fields: "User Name*" (with a placeholder), "Email*" (with a placeholder), "Phone Number (optional)" (with a placeholder), and "Organization Name*" (with a placeholder). Each field is represented by a long, thin, light-blue input bar.

Task 12: Limit Access to the Employee Microservice

- 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.

The screenshot shows the AWS CloudFront Rule configuration interface. The URL in the address bar is `microservicesLB > HTTP:80 listener > Rule (priority 1) > Edit rule`. The page displays two conditions:

- Source IP**: Set to `202.184.107.0/24`. A note indicates: "IPv4 or IPv6 CIDR notation. For example: `192.0.2.0/24` or `2001:db8::/32`. If your client is behind a proxy, enter the IP address of the proxy. No wildcards." A "Remove" button is available.
- Path**: Set to `/admin/*`. A note indicates: "Case sensitive." A "Remove" button is available.

Below the conditions, there are sections for "Match pattern type" (Value matching selected) and "Path condition value" (`/admin/*`).

Conditions (2 values) Info

Define 1-5 condition values. Additional conditions can't be added once the limit is reached.

▶ Path (value) = `/admin/*`

AND

▶ Source IP = `202.184.107.0/24`

[Add condition ▾](#)

You can add up to 3 more condition values for this rule.

Transforms - optional (0) Info

Modify incoming request host headers and URLs before routing.

[Add transform ▾](#)

You can add up to 2 more transforms.

Actions Info

Requests matching all rule conditions route according to the rule actions.

Routing action

Forward to target groups

Redirect to URL

Return fixed

Forward to target group Info

Choose a target group and specify routing weight or [create target group](#).

Target group

employee-tg-one

Target type: IP, IPv4 | Target stickiness: Off

HTTP ▾



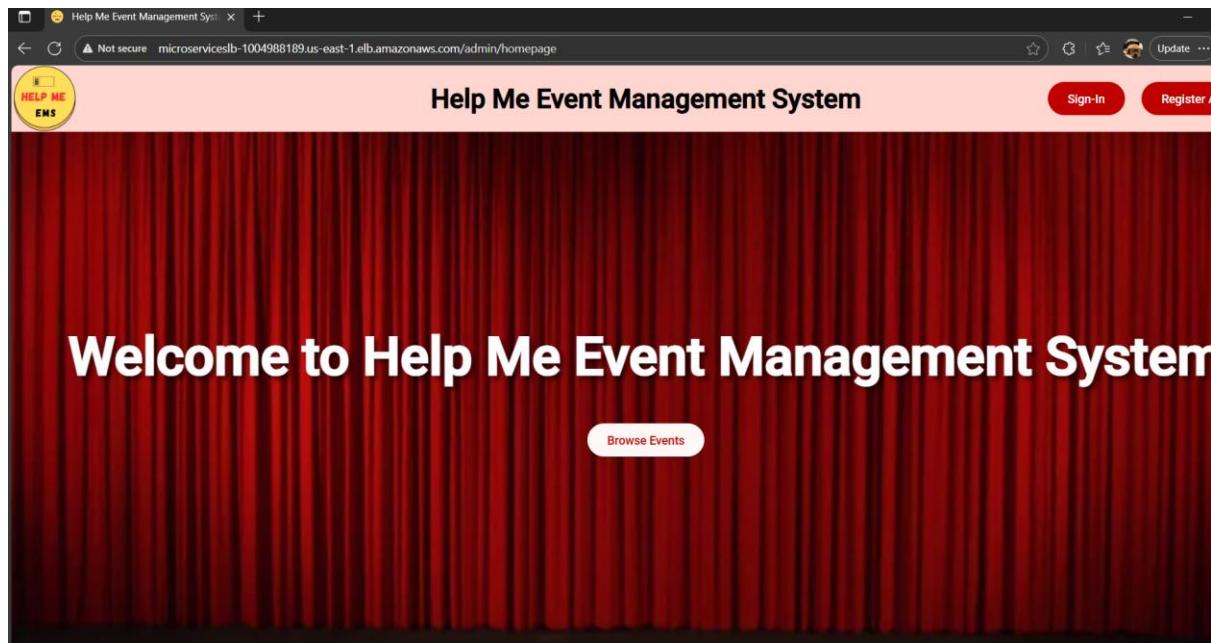
Weight

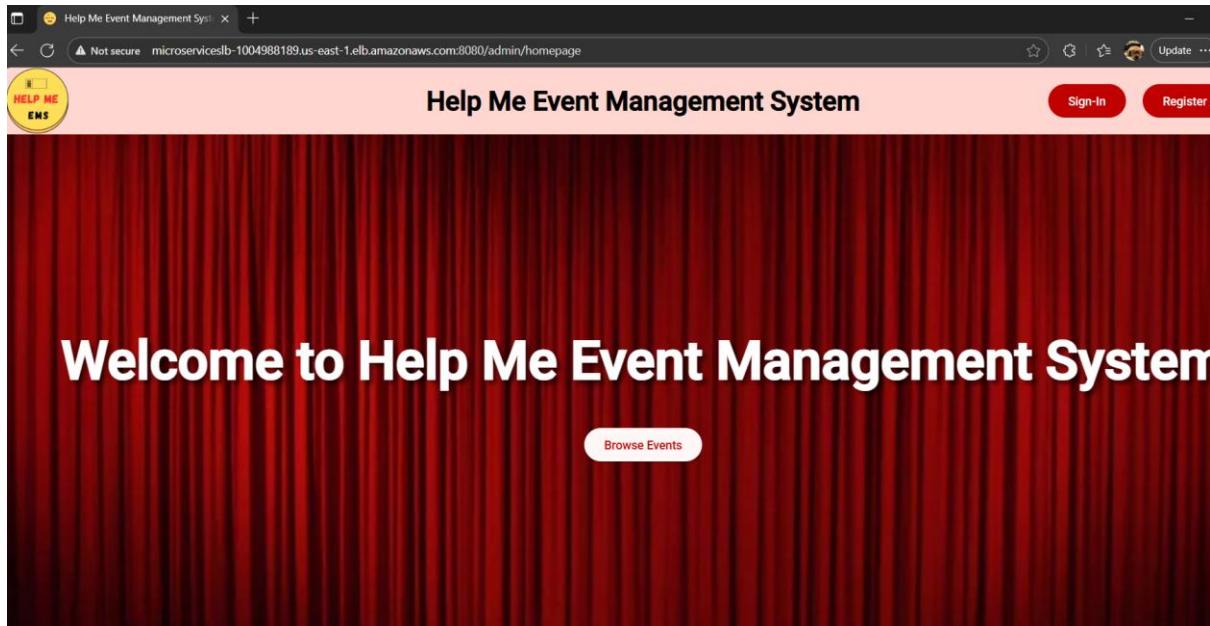
1

Percent

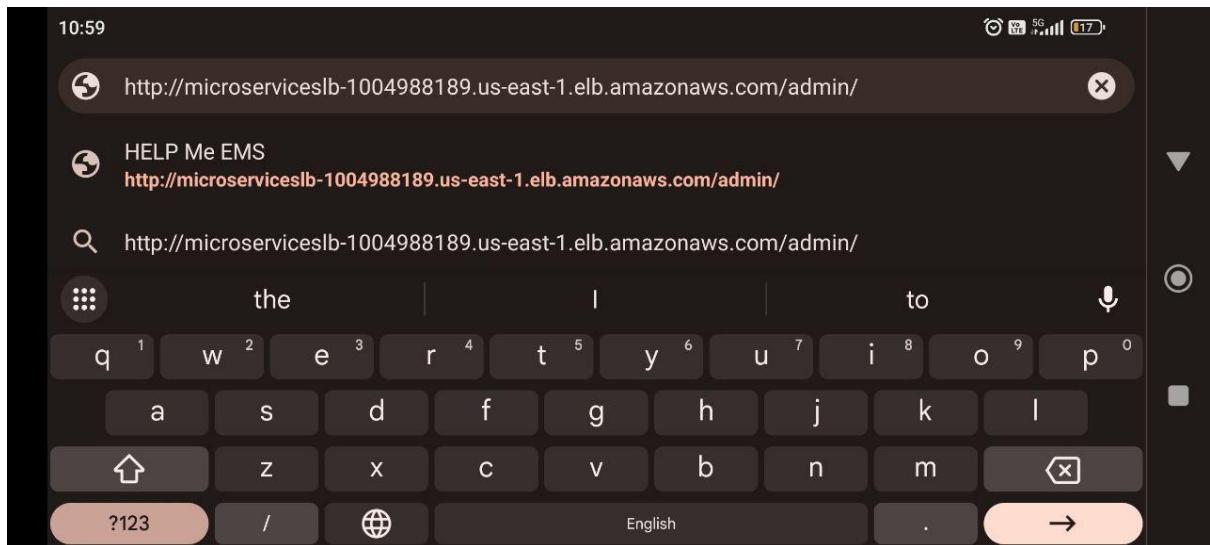
100%

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.

