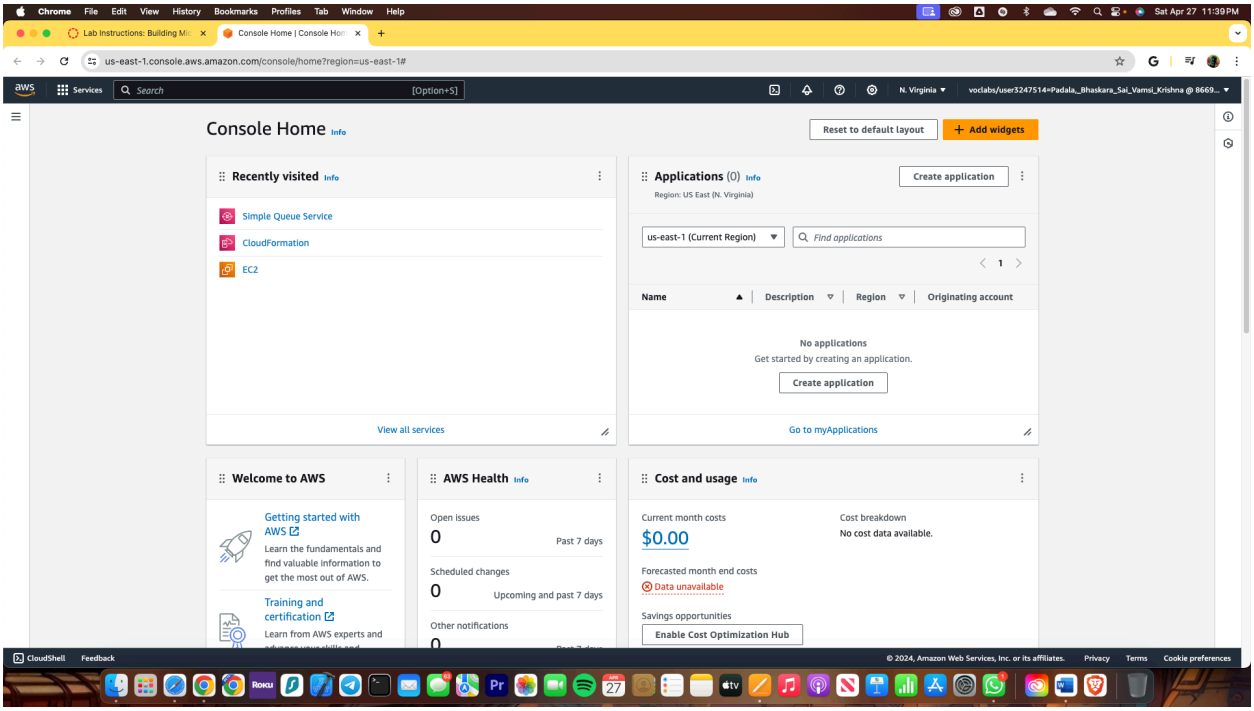
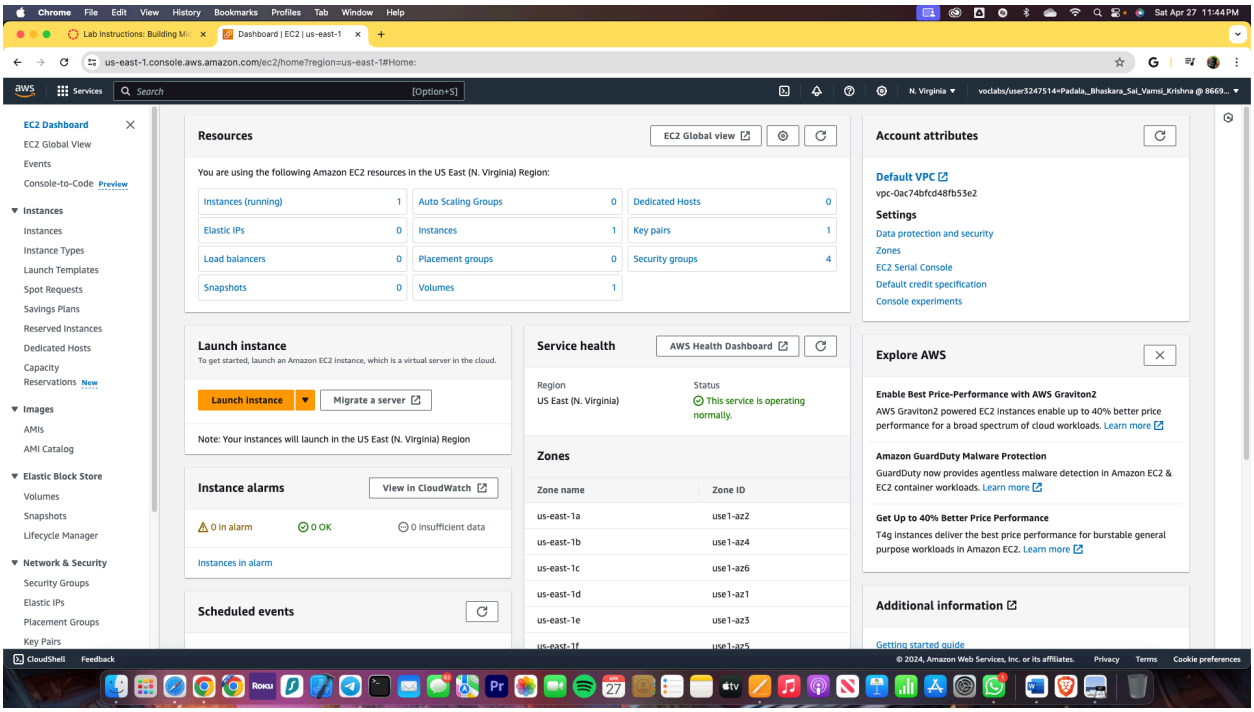
CS 524 - Final_Project

CWID : 20025924

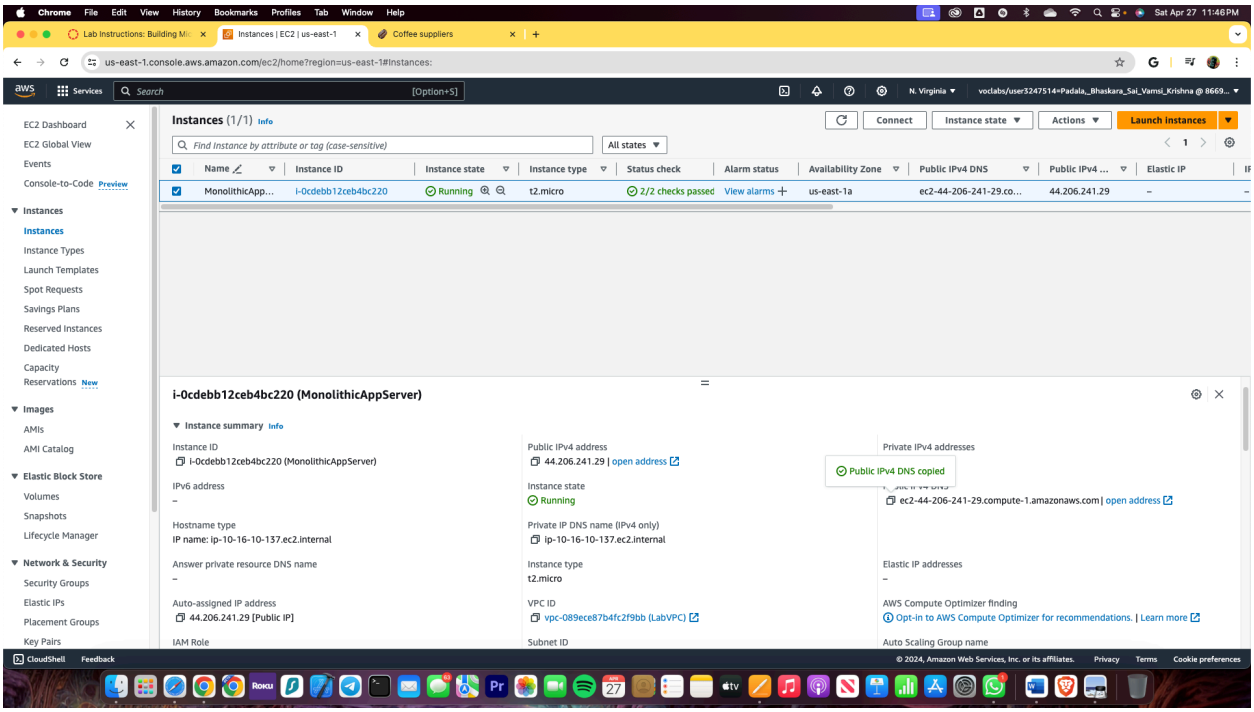
Firstly, open the AWS academy and open the LAB and then open AWS console as shown in the figure below.



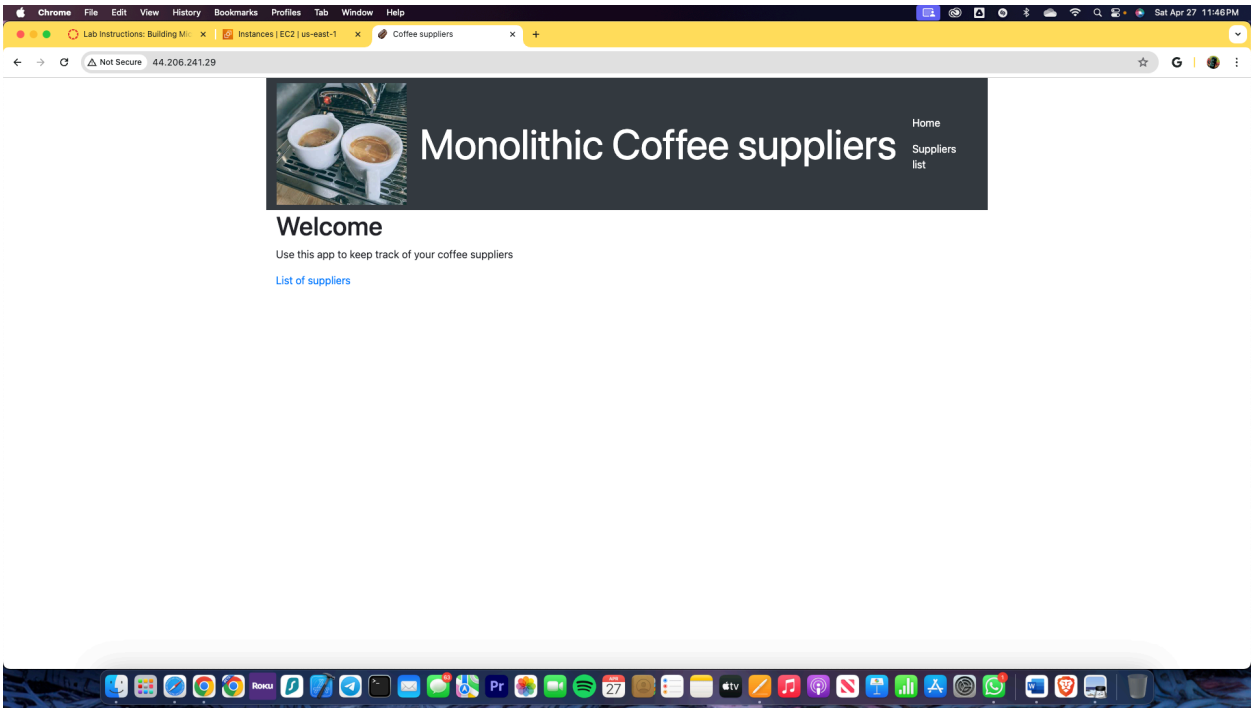
Now Navigate to EC2 console as given in the lab.



Now, we will copy the Public Ipv4 address and now we will paste it in another tab as shown

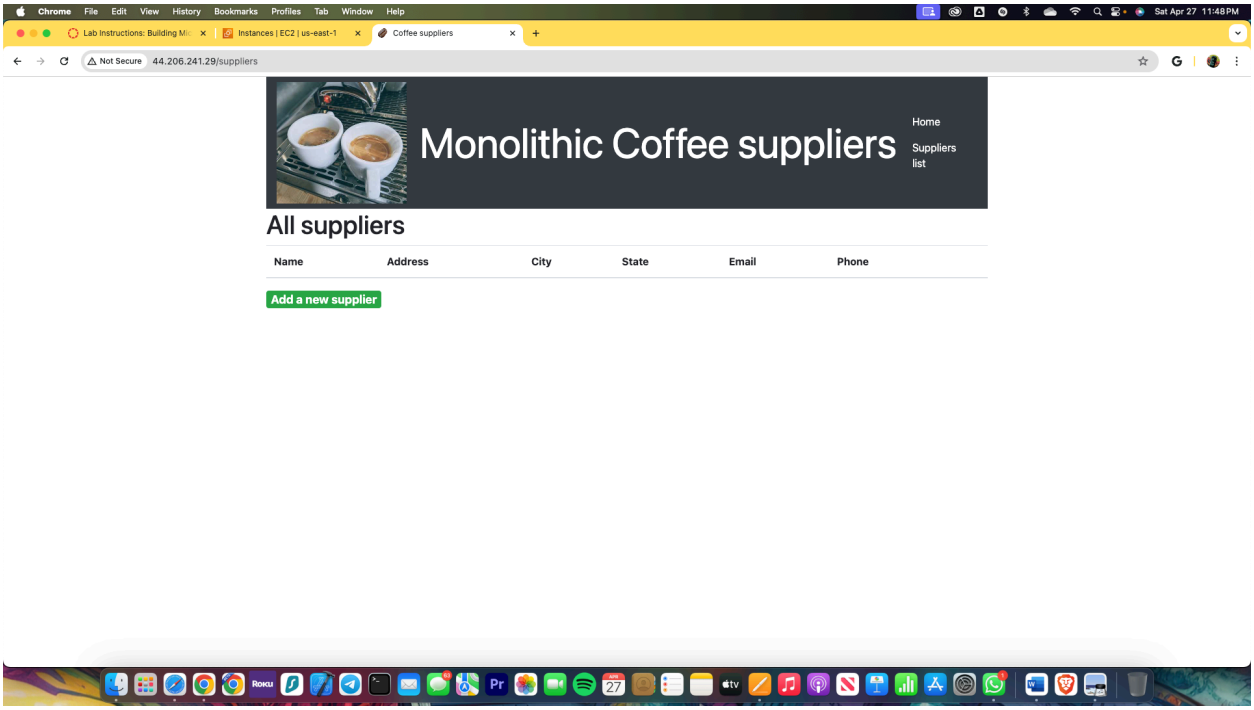


below.

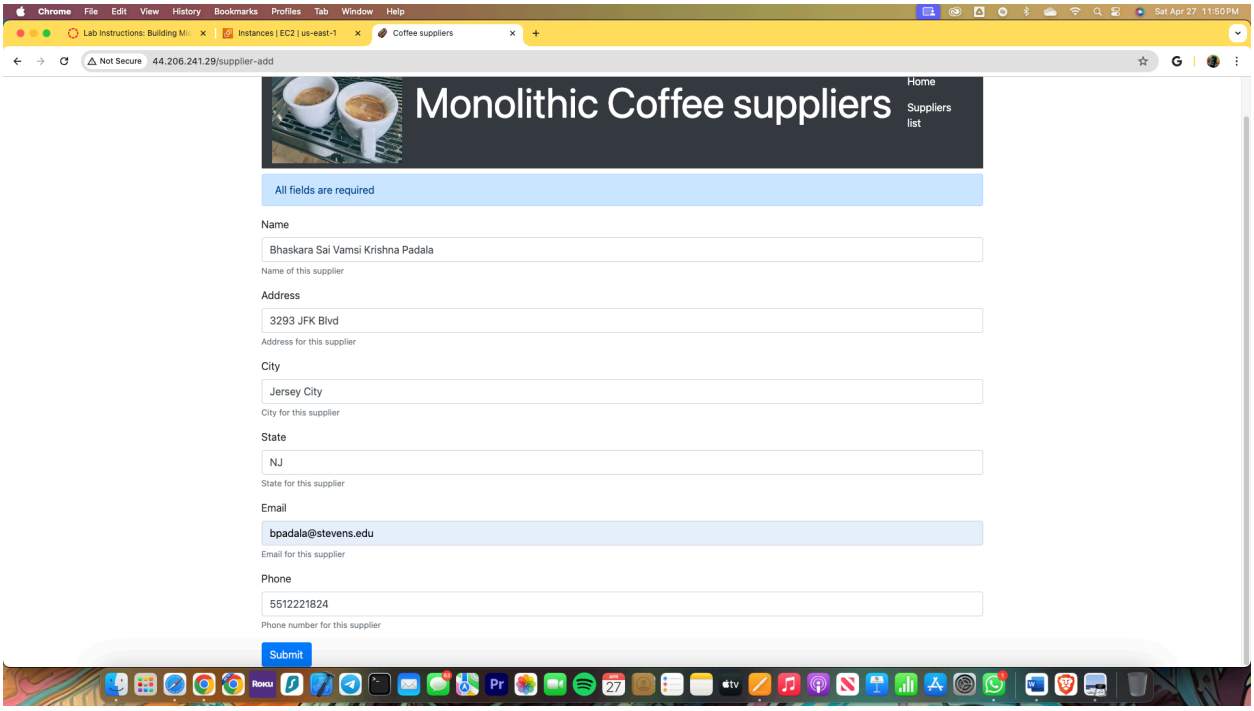


Task 2.2: Test the monolithic web application

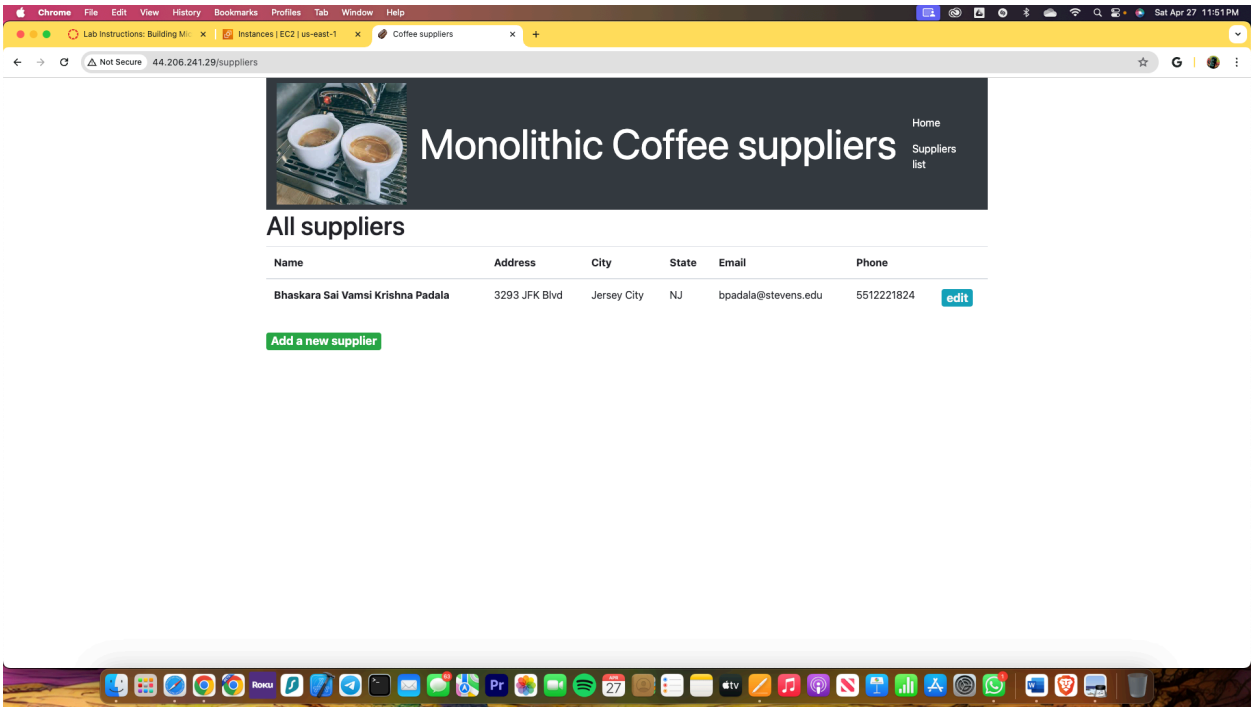
Here click on the list of Suppliers and now we will get the following screen as shown in the figure below.



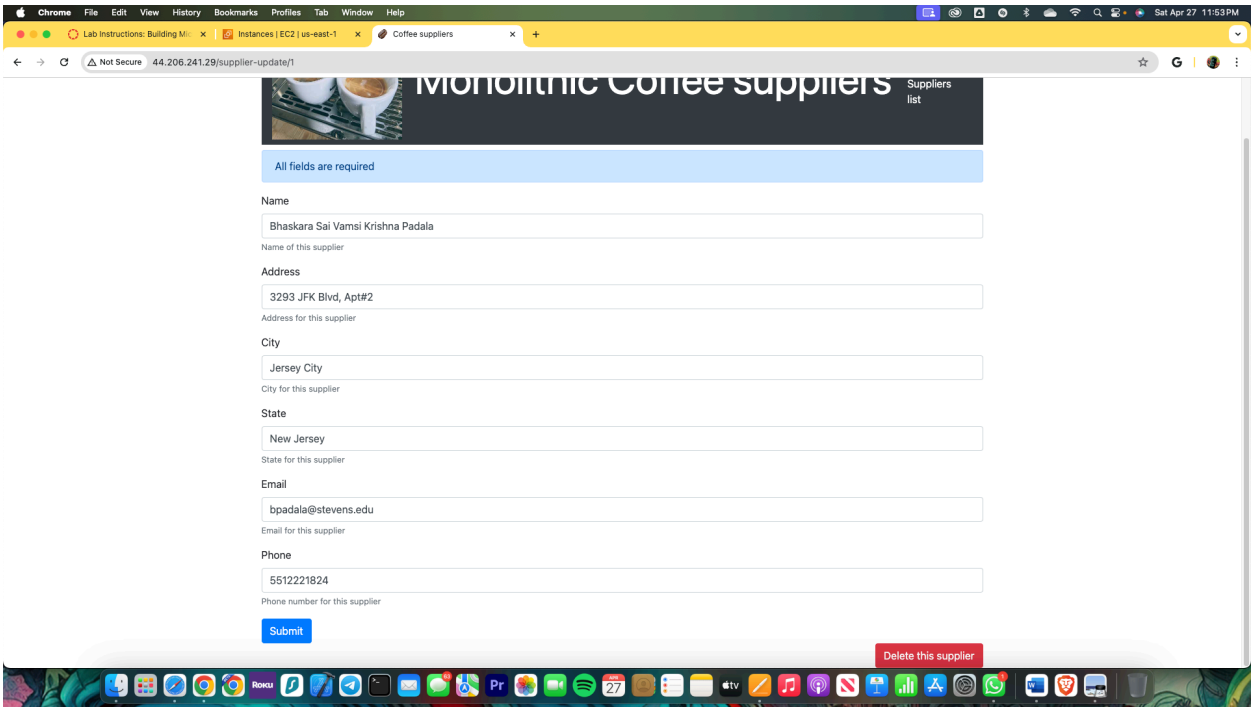
Now fill in the fields as shown below and add the supplier as shown.



Now click on the edit entry button so that we can edit the supplier in the page after we get after clicking on it.



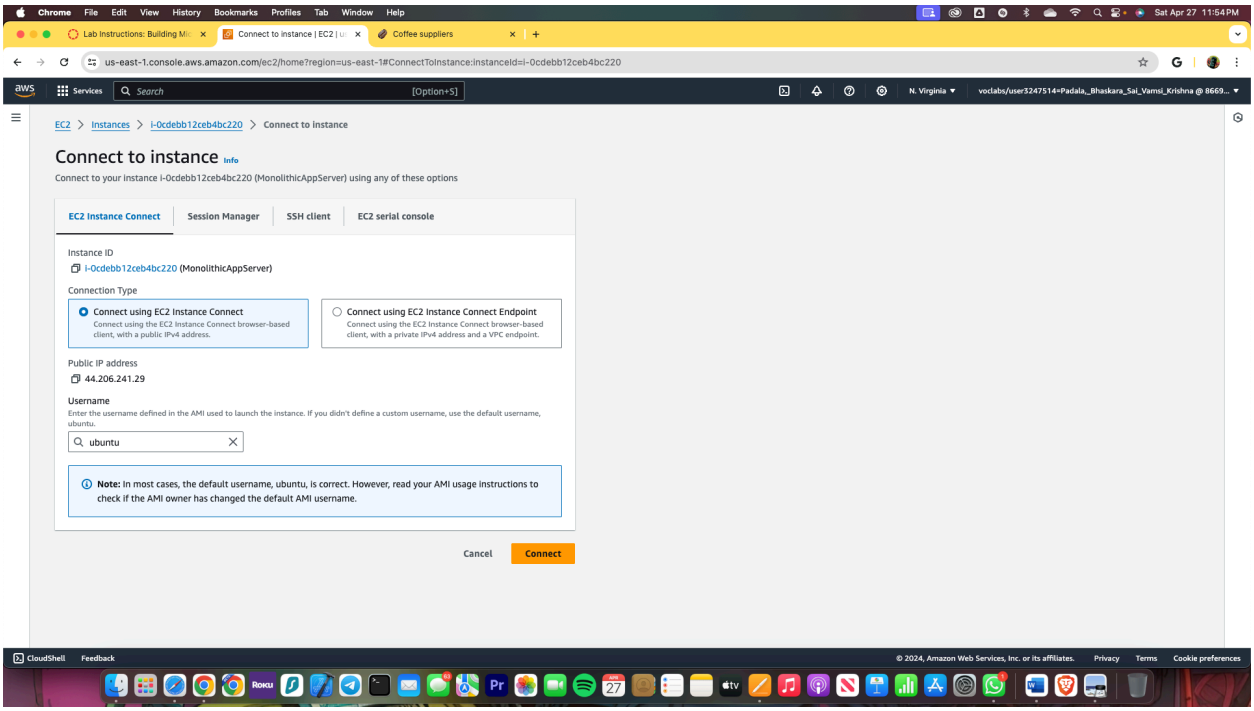
Now I edited the supplier details and we will see the difference as shown the next figure below.



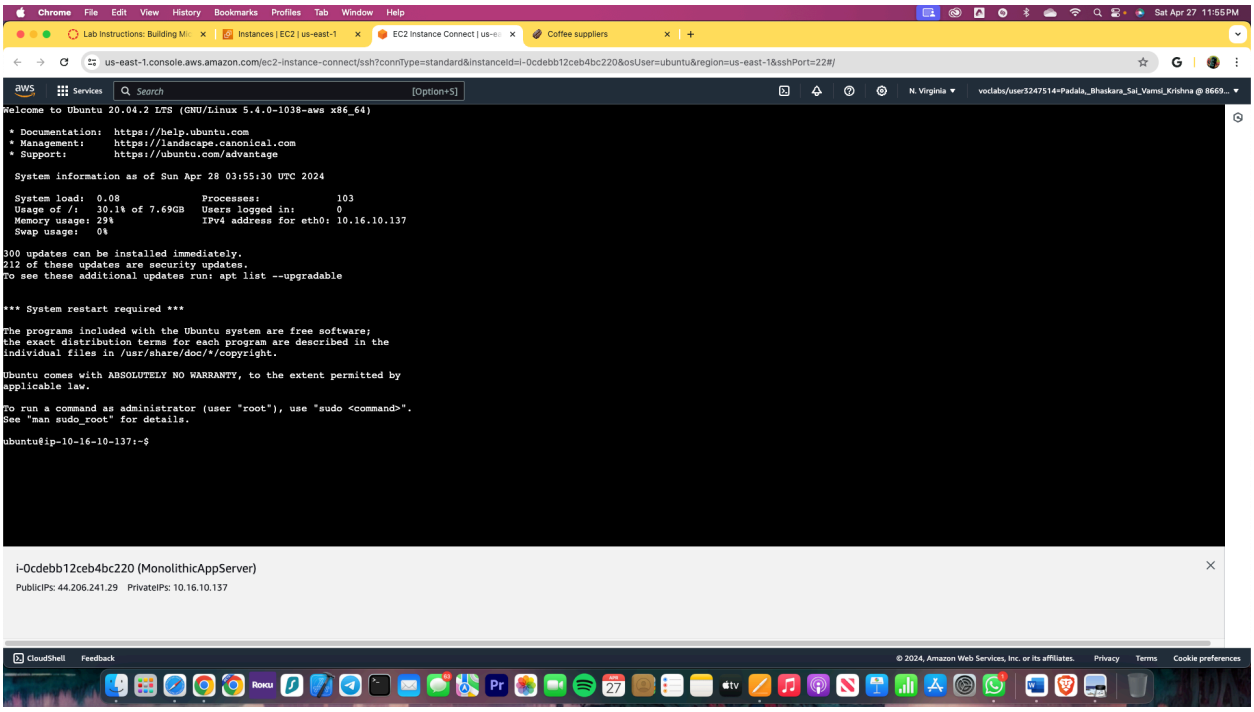
Now let's move on the TASK 2.3

Task 2.3: Analyze how the monolithic application runs

Now connect to the Monolithic App instance and we will proceed as shown In the figure below.

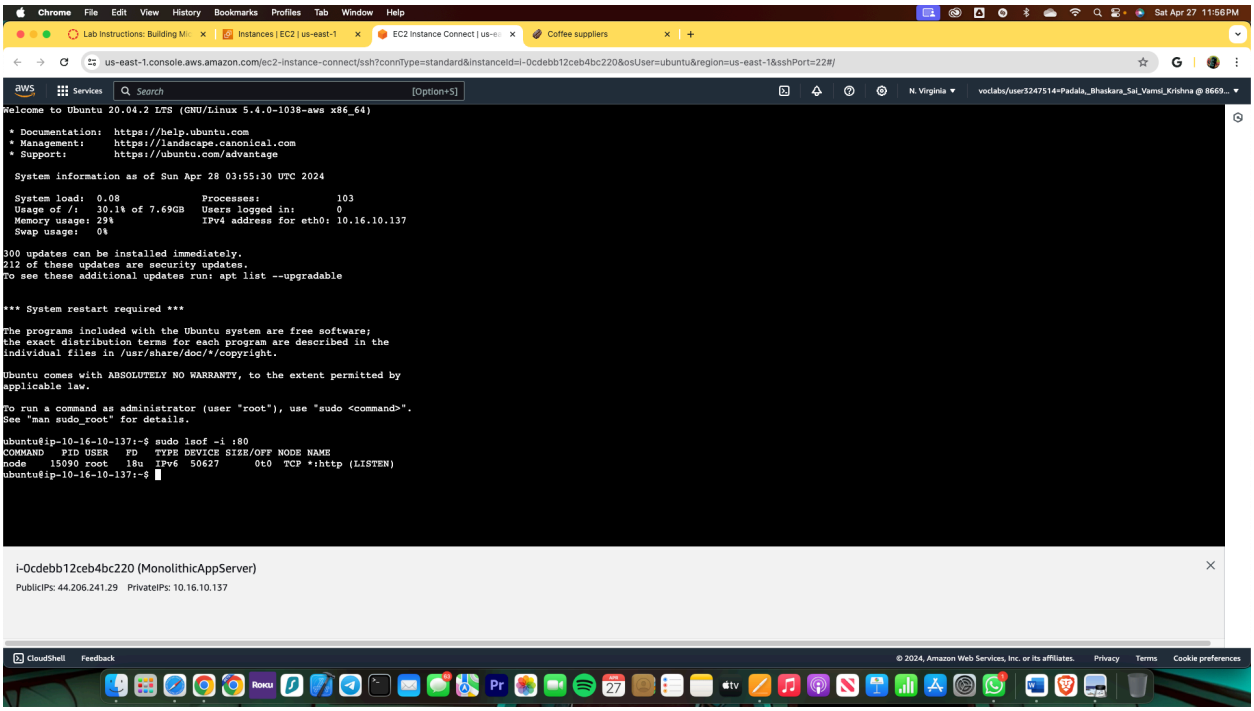


After connecting to the MonolithicAppInstance we will get the following console screen as shown.



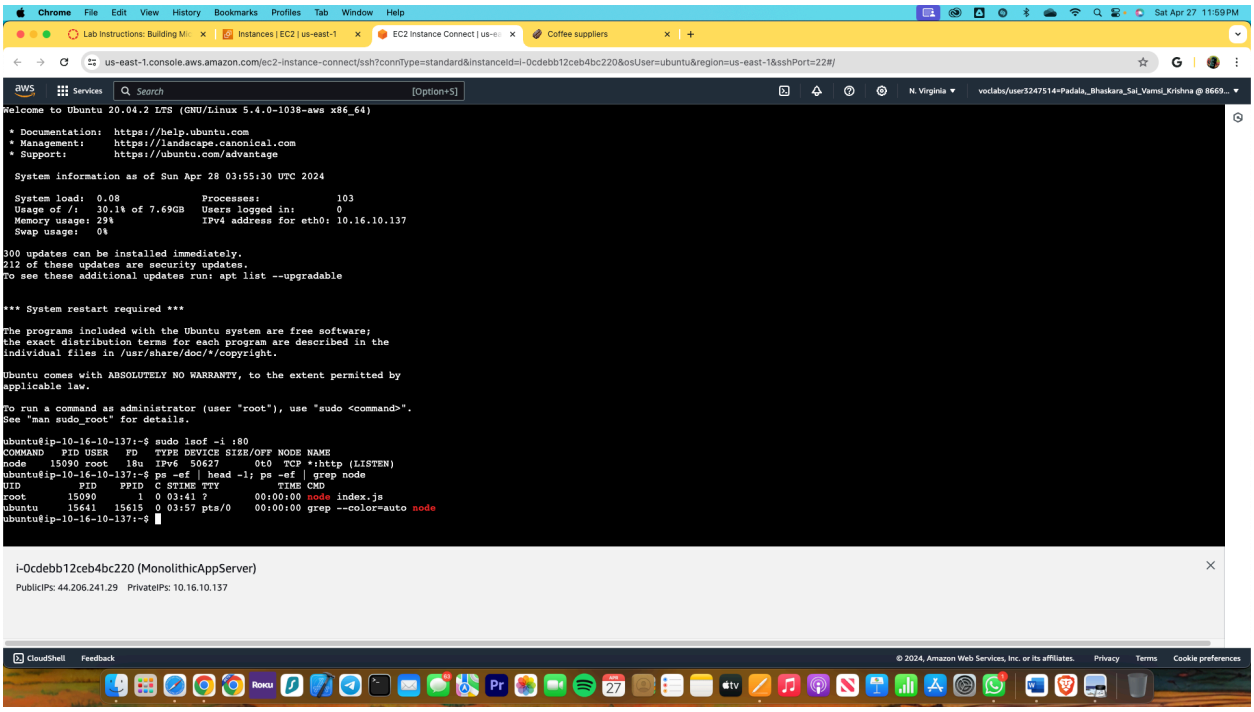
After running the following command, we will get the following output on the screen as shown.

```
sudo lsof -i :80
```



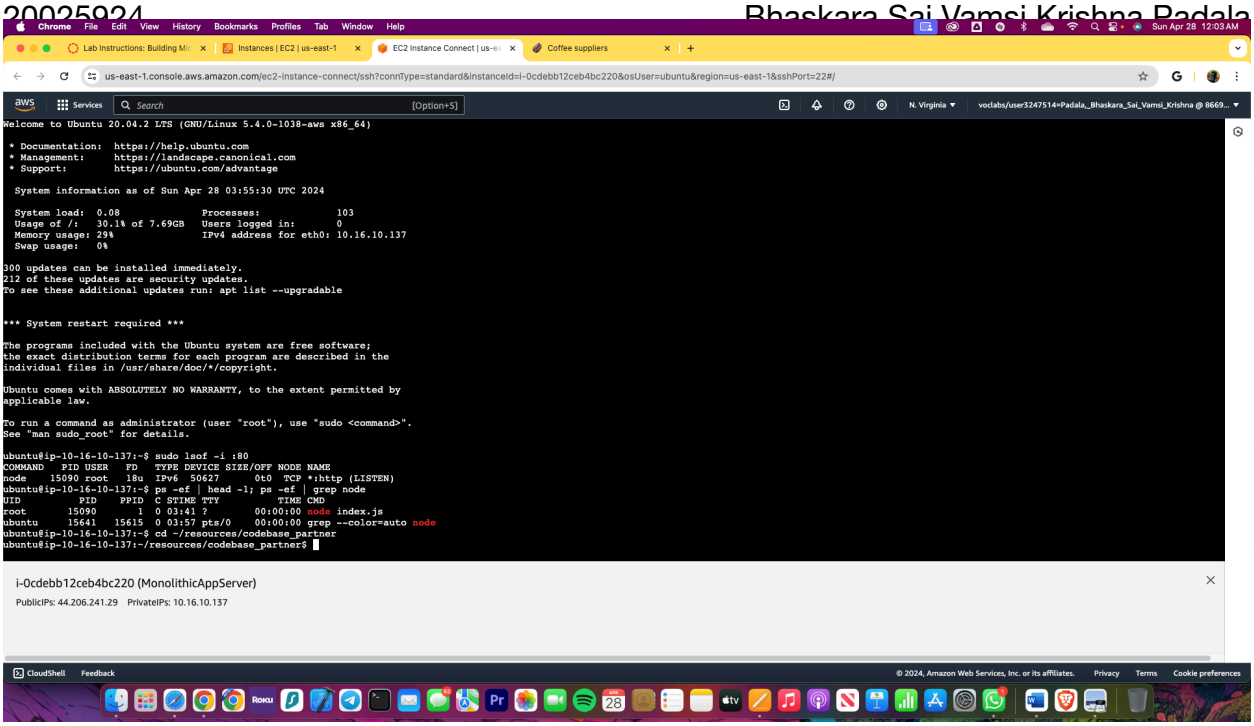
Now let's run the second command which is given below and after running the command we get the following output.

ps -ef | head -1; ps -ef | grep node

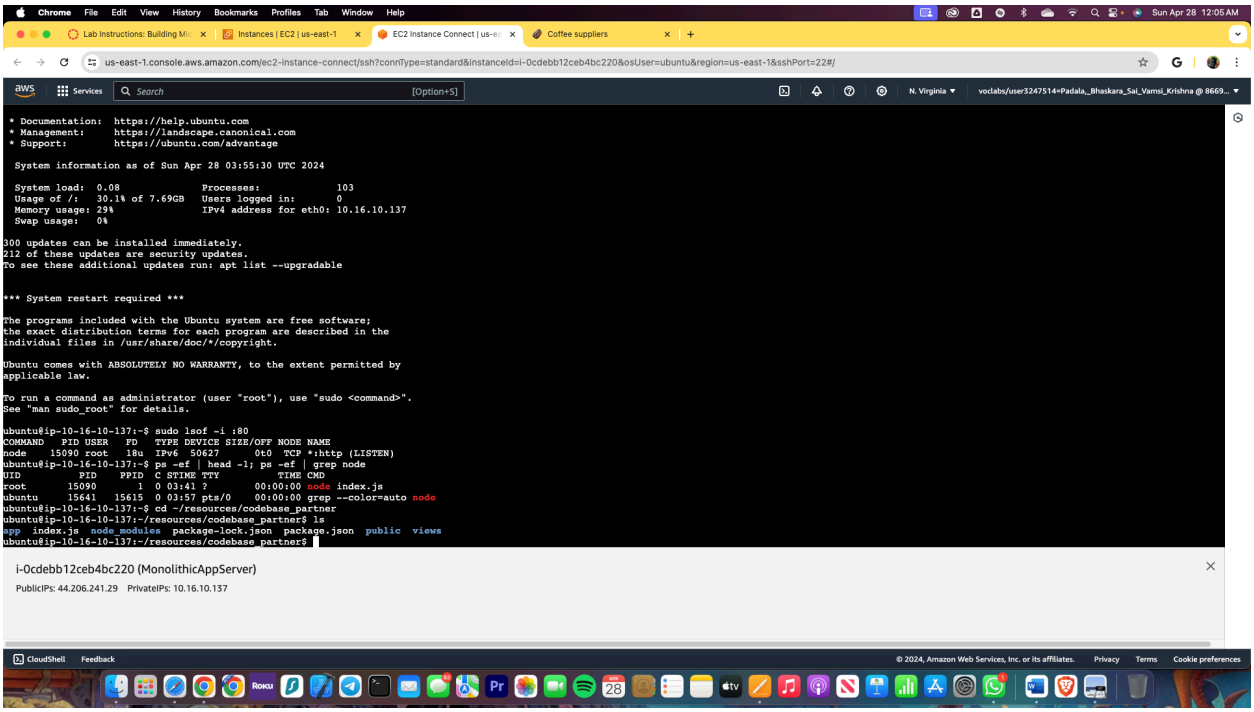


Now let's run the cd command which is shown below and we will get the following output.

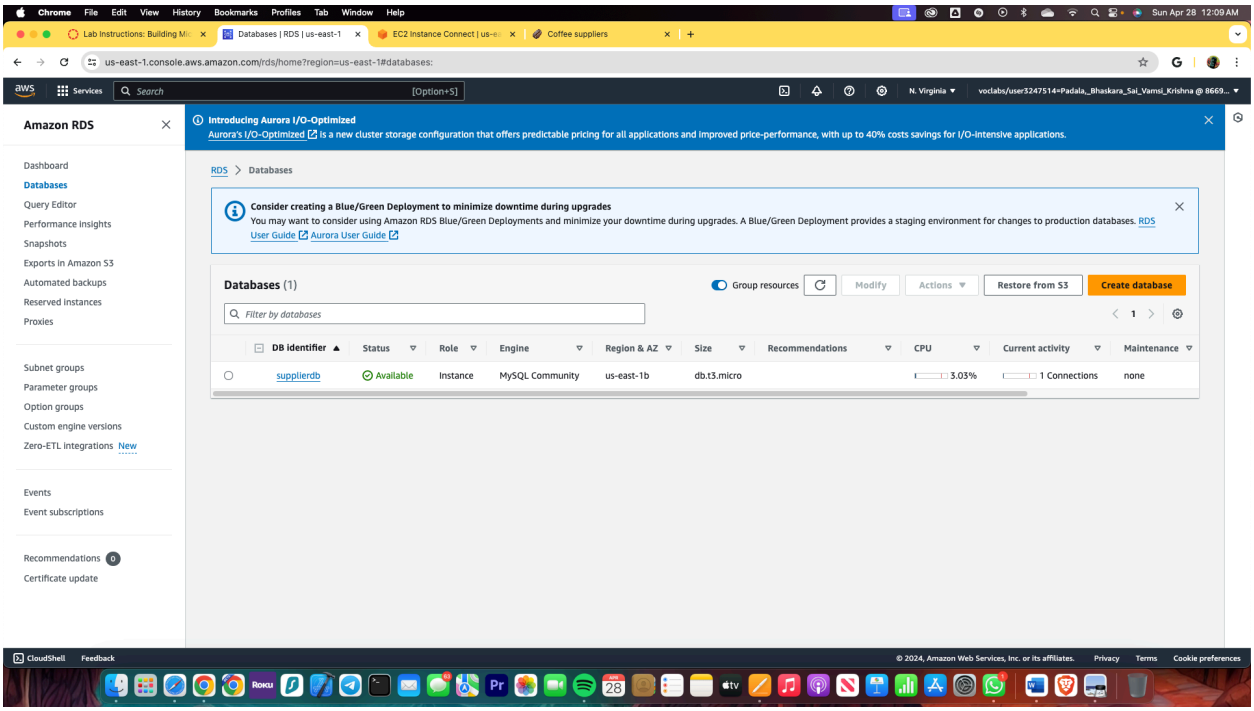
```
cd ~/resources/codebase_partner
```



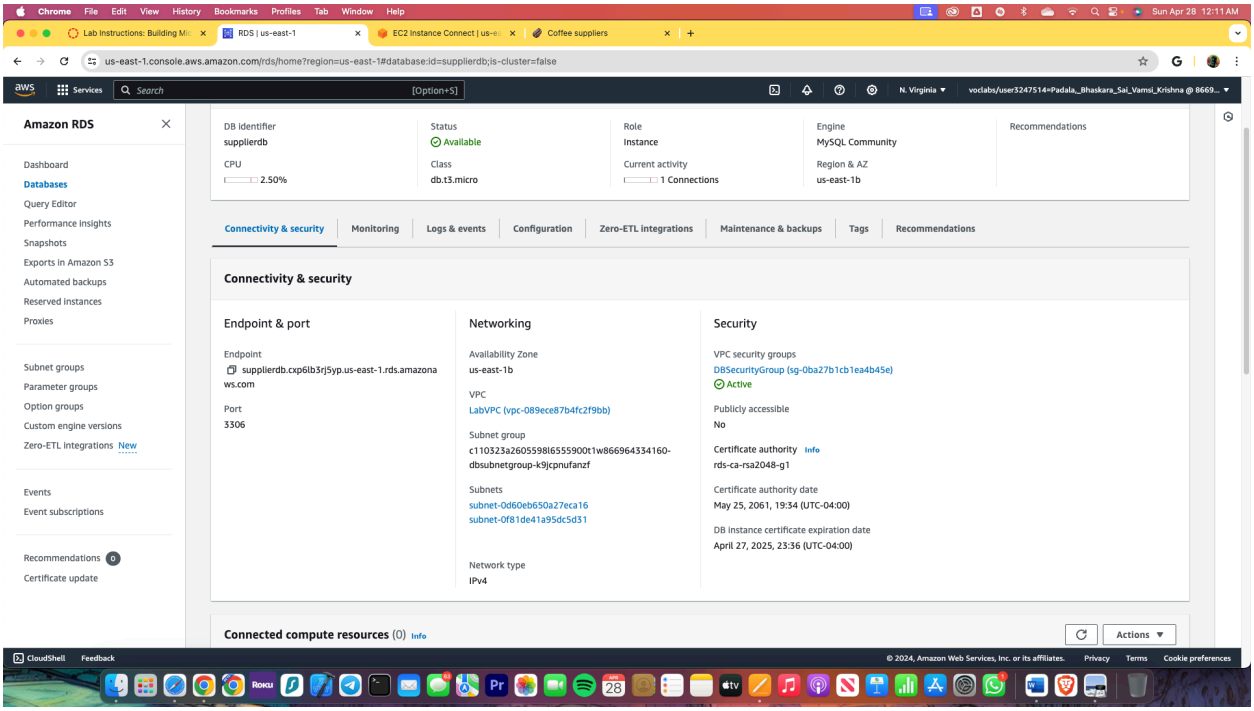
Now let's run the ls command and we will get the following output.



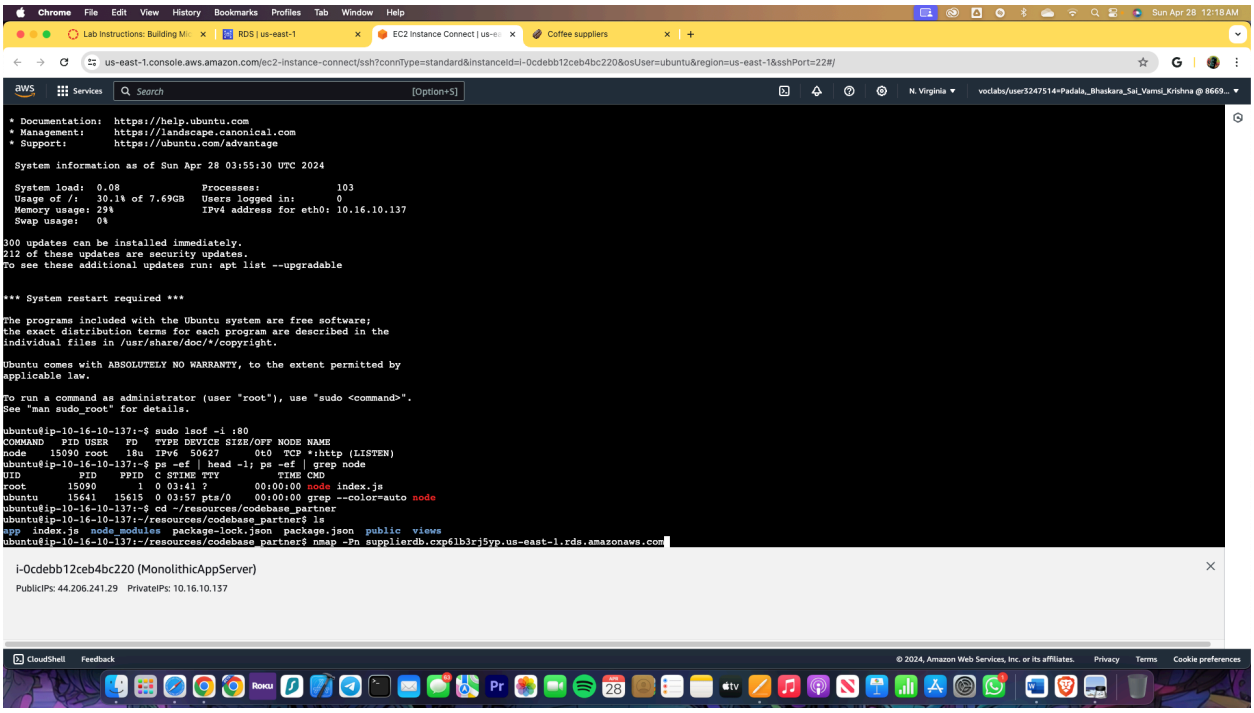
Now we will connect the RDS by searching for it in the resources and we will get the following screen.



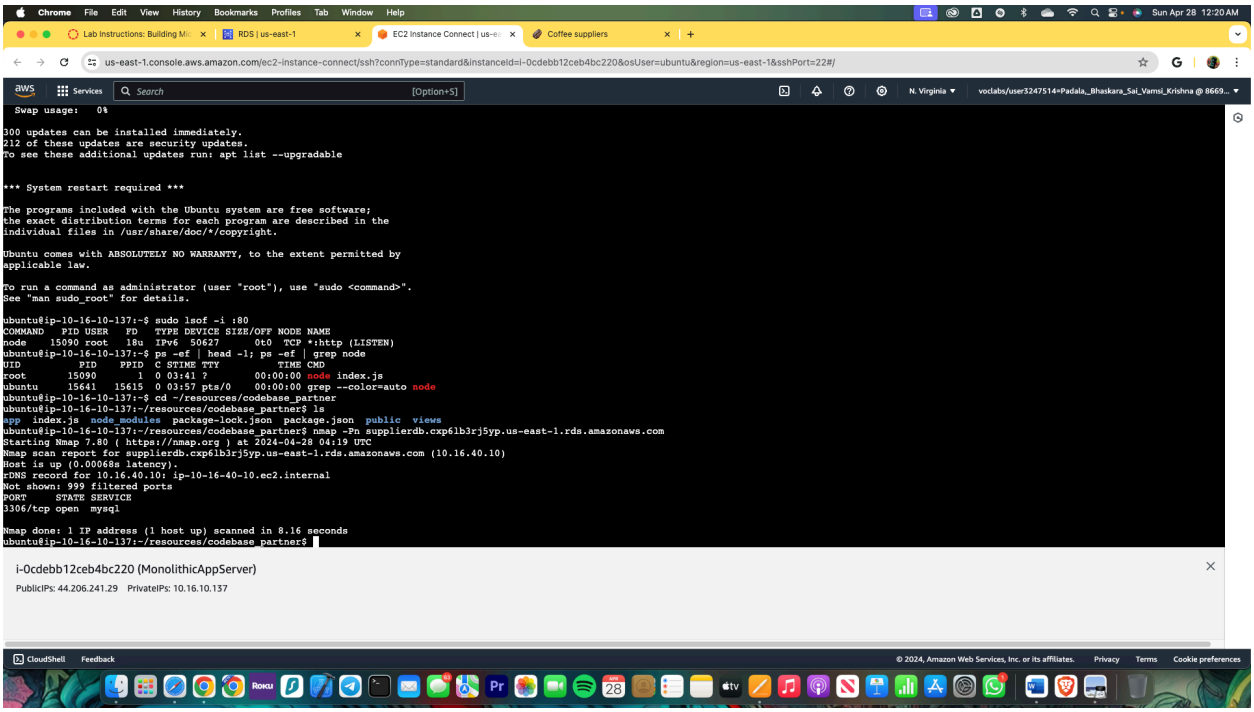
Now we will find and copy the Endpoint port number as shown in the below screen.



Now we will paste the following commands side by side in the command prompt and we will then run the console as shown below.
nmap -Pn supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com

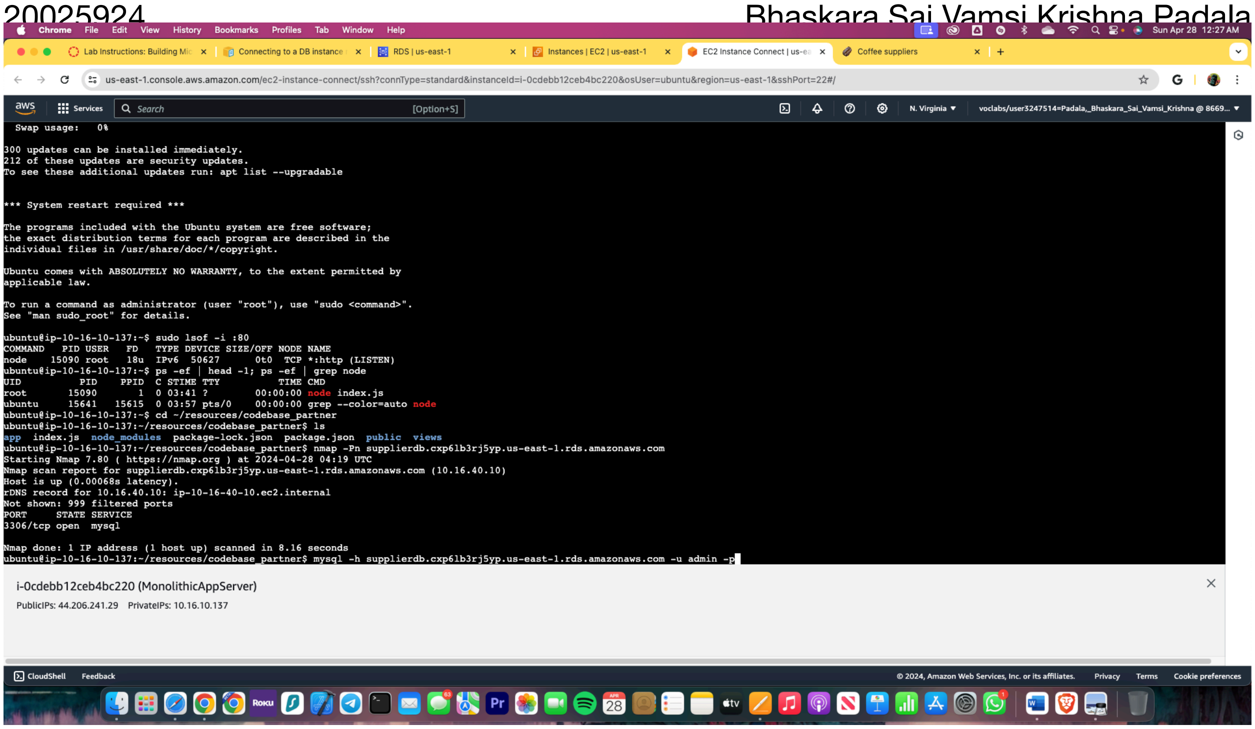


Then we will get the following screen after running the command in the command prompt.

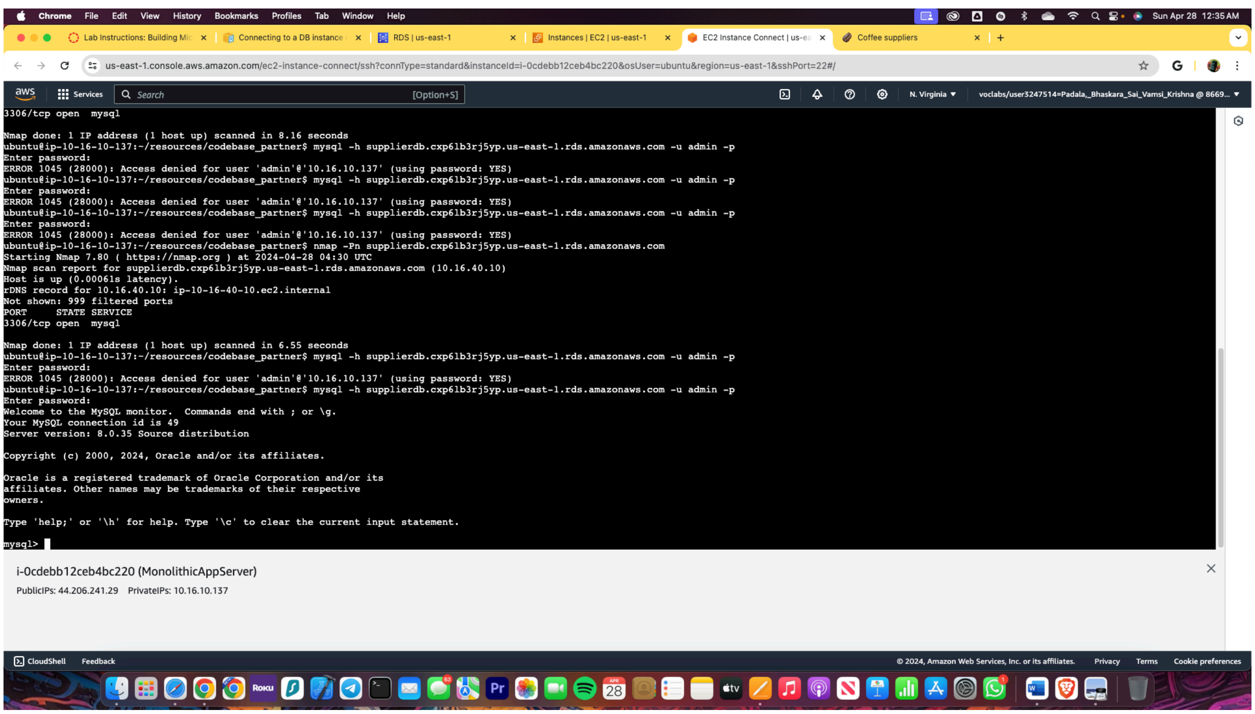


Now we will follow the instructions to connect the MySQL and to do that we will type the command the following prompt as shown below.

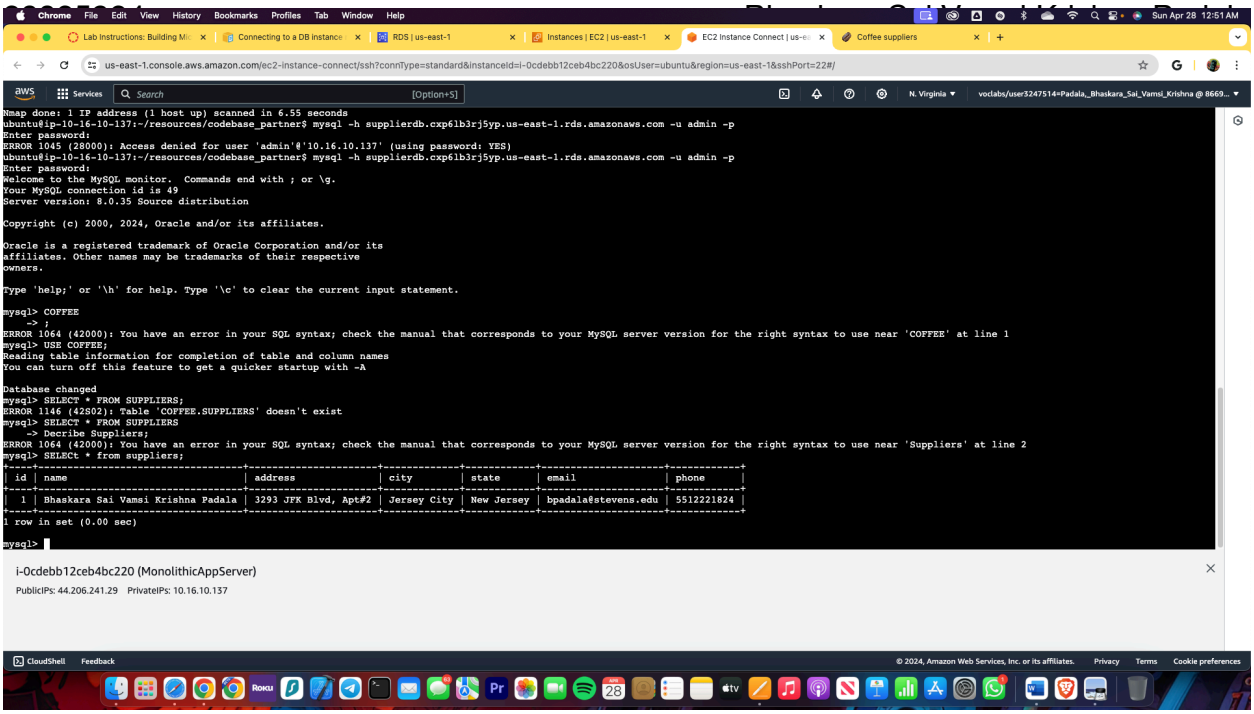
-h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p



Now enter password as lab-password and we will get the following screen.

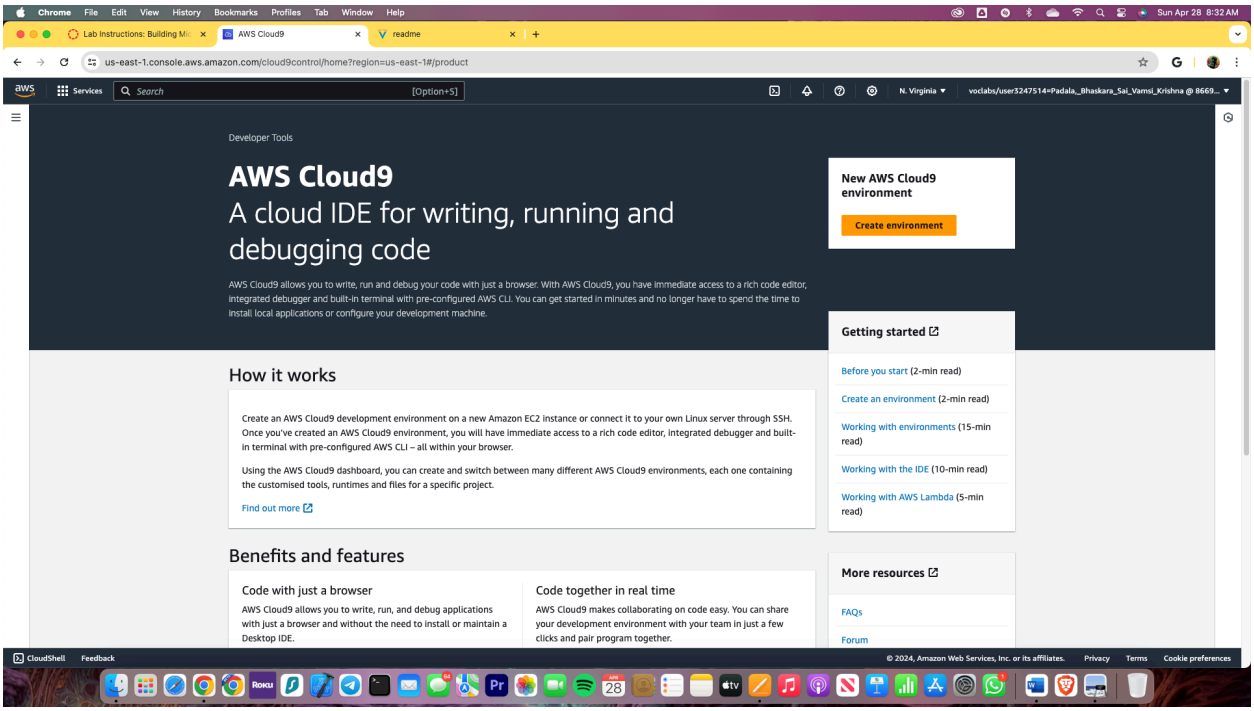


Now after connecting to MYSQL, run the following command which is
SELECT * FROM SUPPLIERS;



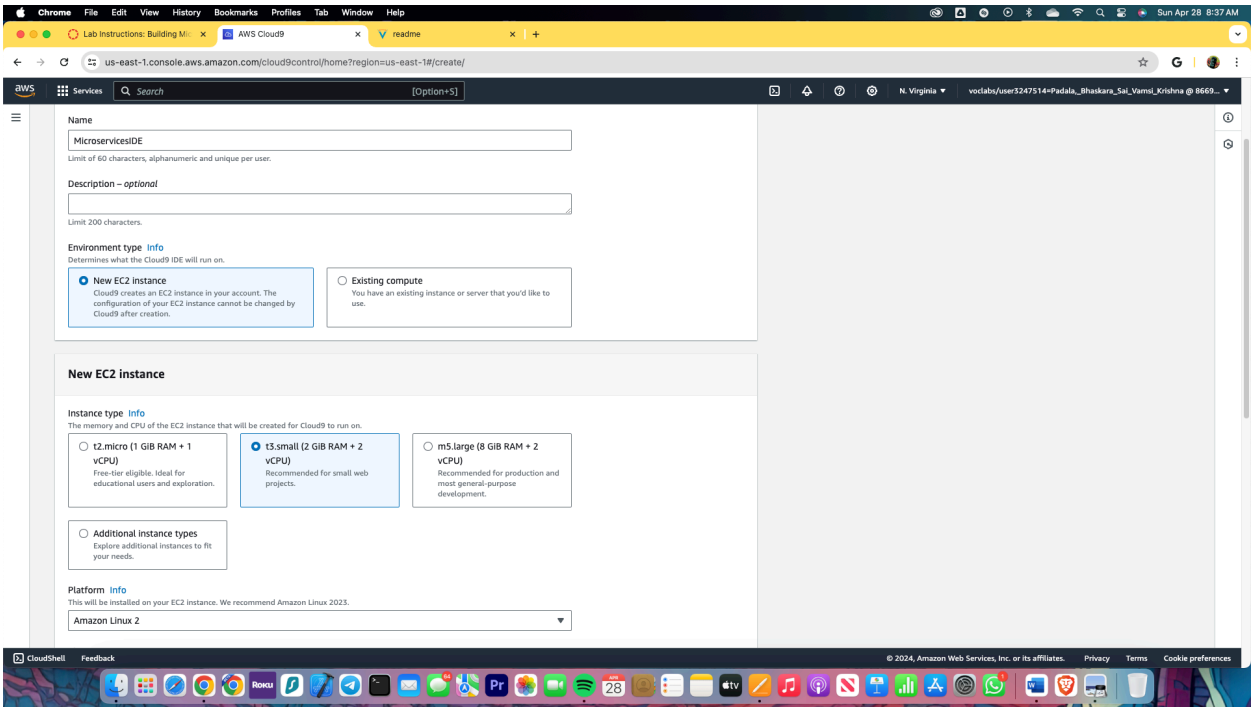
Now we will Exit the MySQL client and then we will close the EC2 Instance Connect tab. Also we will close the coffee suppliers web application tab.

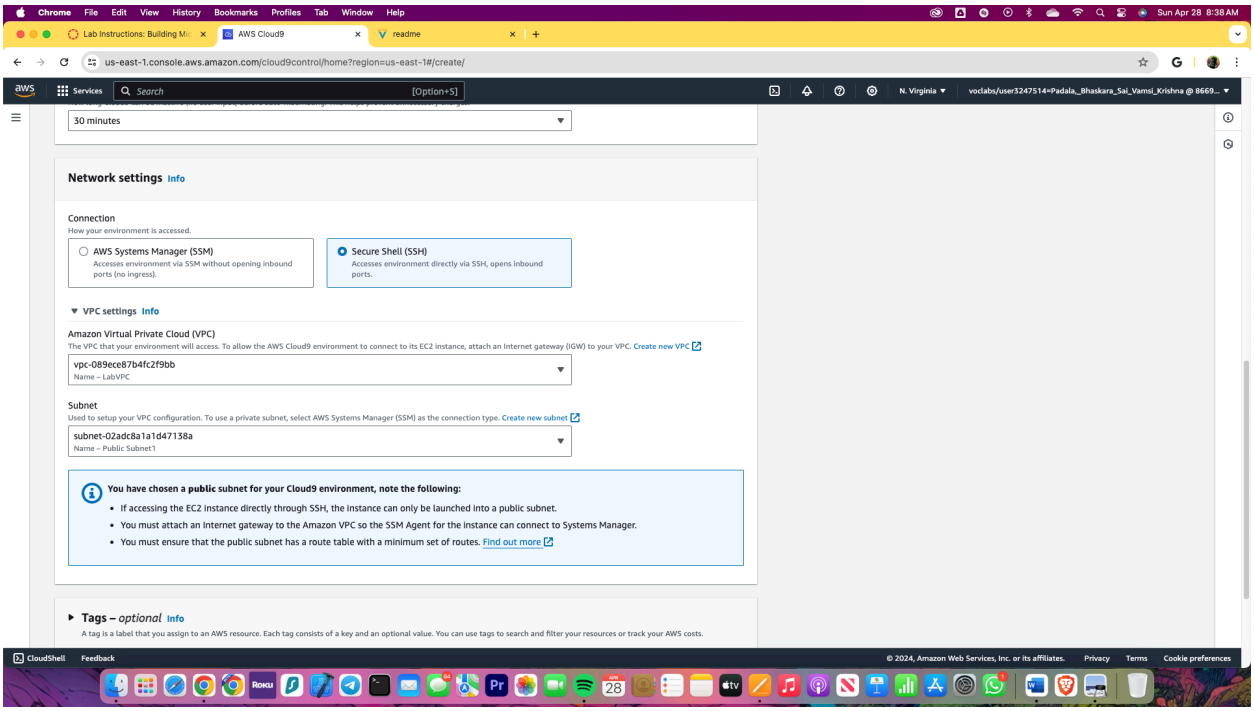
Phase 3: Creating a development environment and checking code into a Git repository



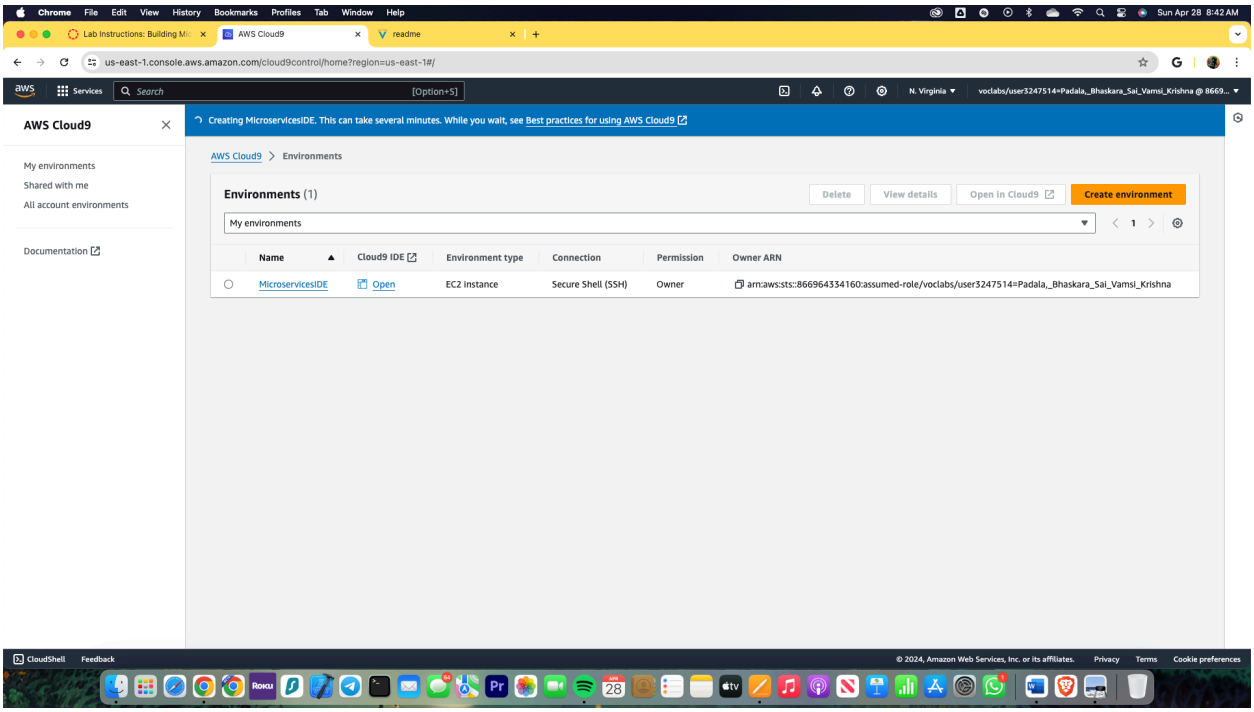
Let's search for Cloud9 and we will get the following screen as shown. Create an AWS Cloud9 instance that is named MicroservicesIDE and then open the IDE.

It should run as a new EC2 instance of size t3.small and run Amazon Linux 2. The instance should support SSH connections and run in the LabVPC in Public Subnet1.

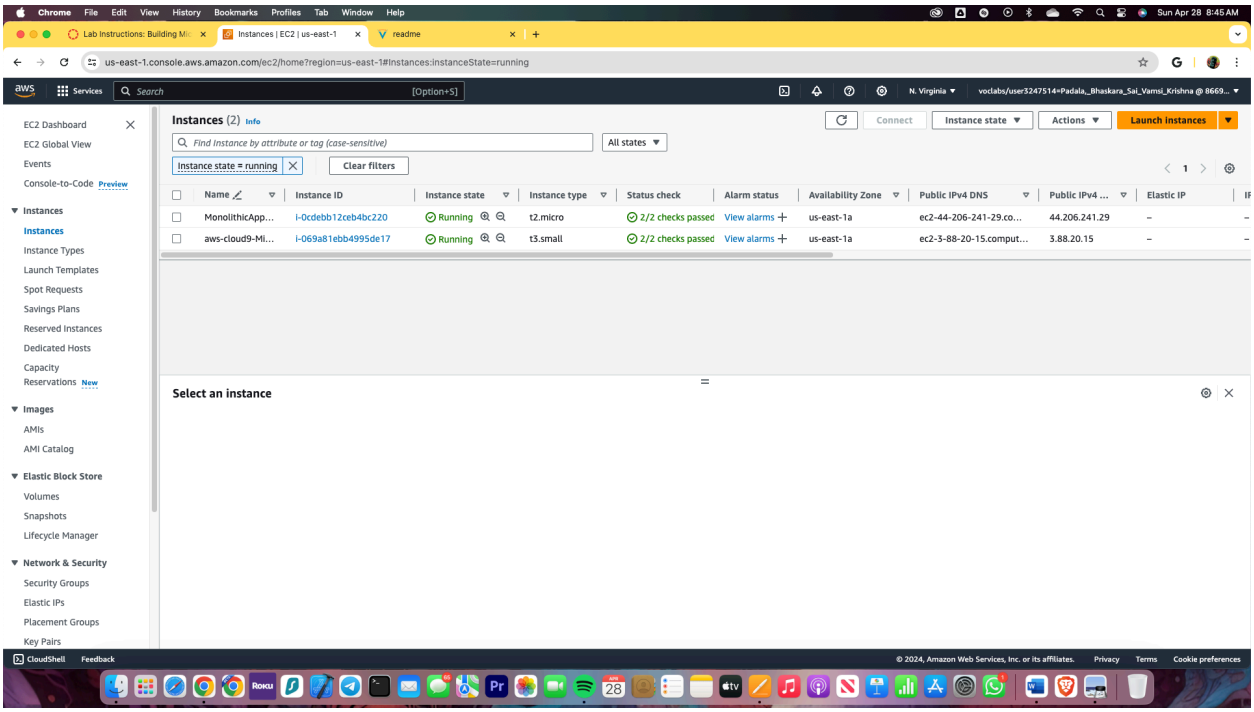




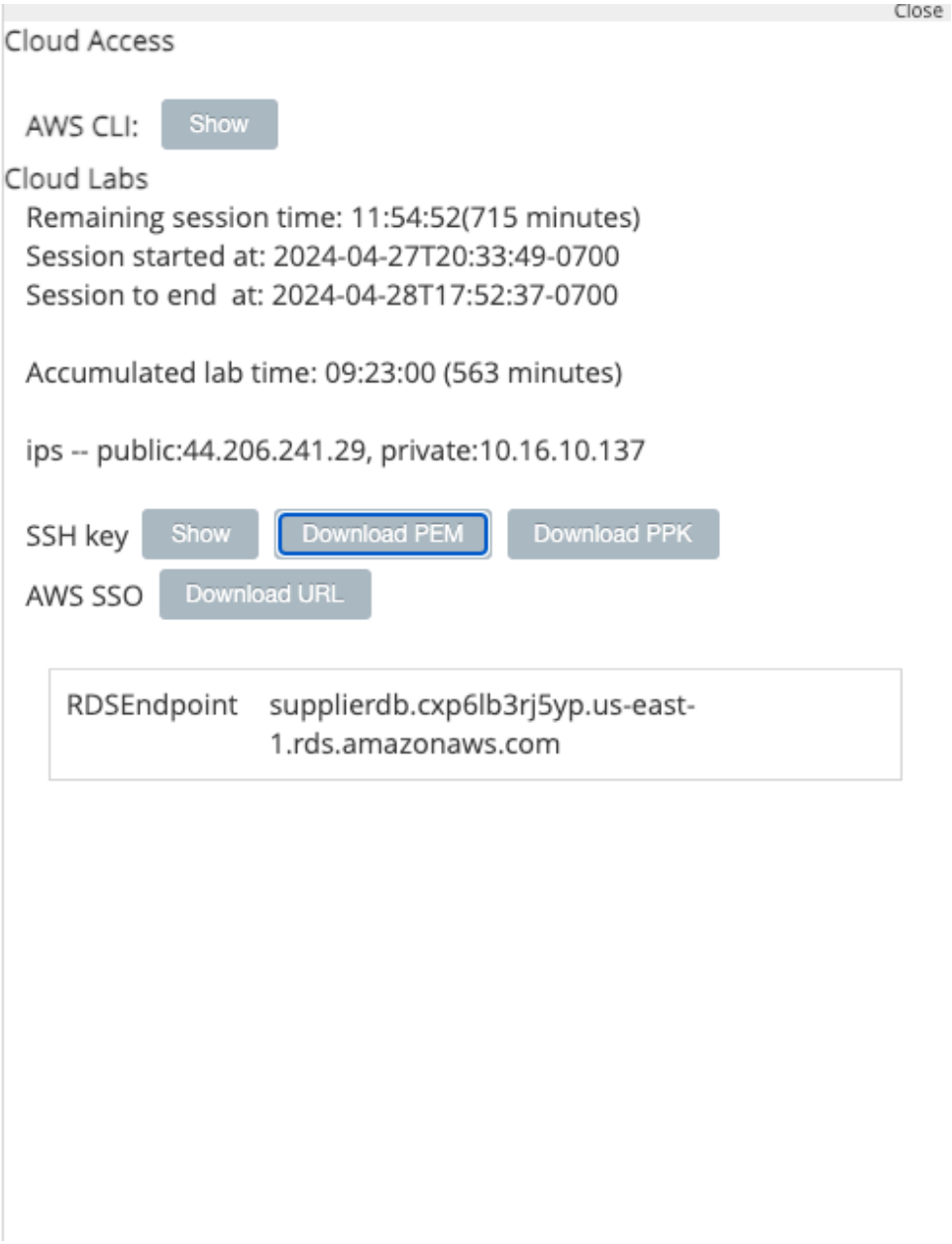
Now after creating and editing the configurati0ons we get the following screen.



Now go to the instances page and we will get two instances up and running.

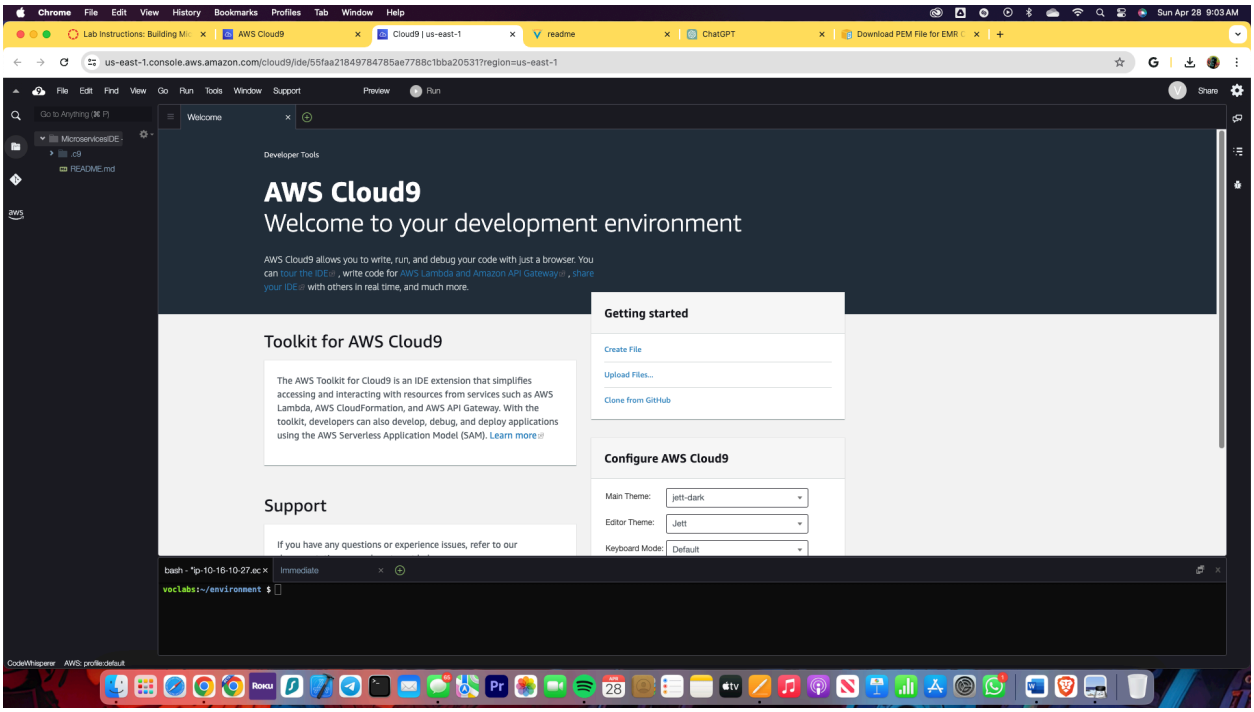


Now we downloaded the .pem file from the instructions page. From the AWS Details panel on this lab instructions page, we downloaded the labsuser.pem file to our local computer.

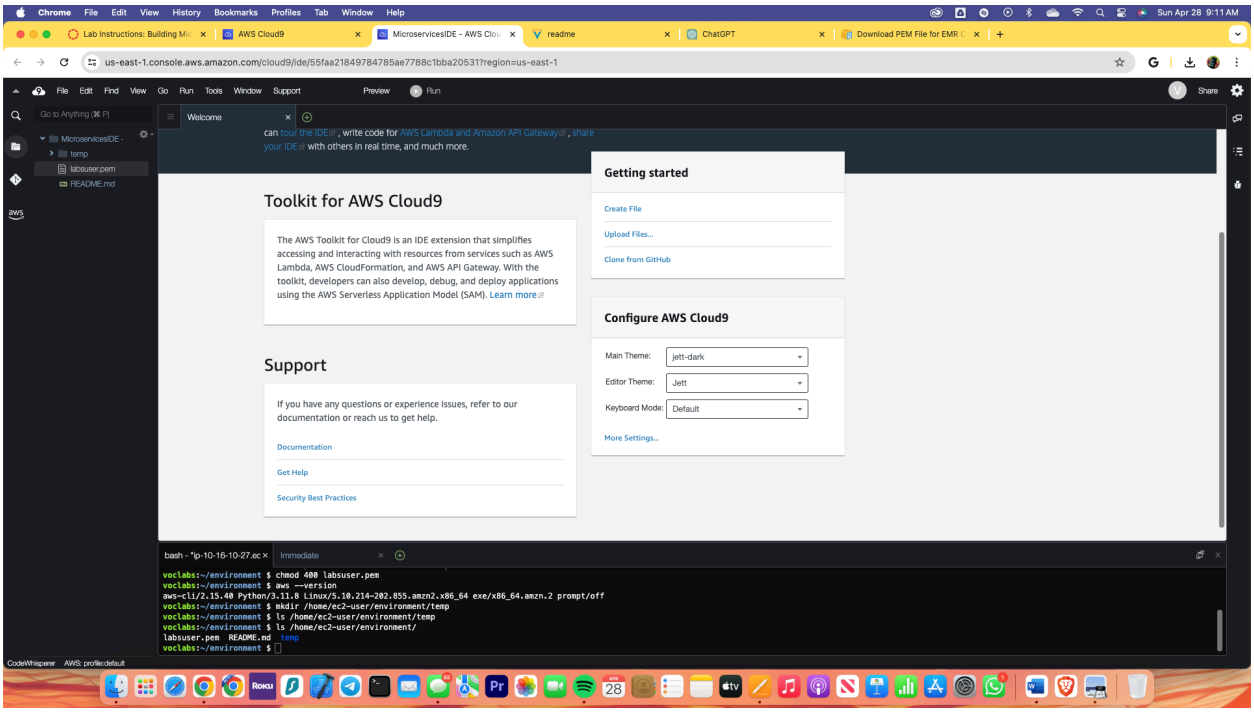


Now we will, Navigate to the AWS Cloud9 console and open the Cloud9 IDE environment that you want to use. And then we will Upload the .pem file:

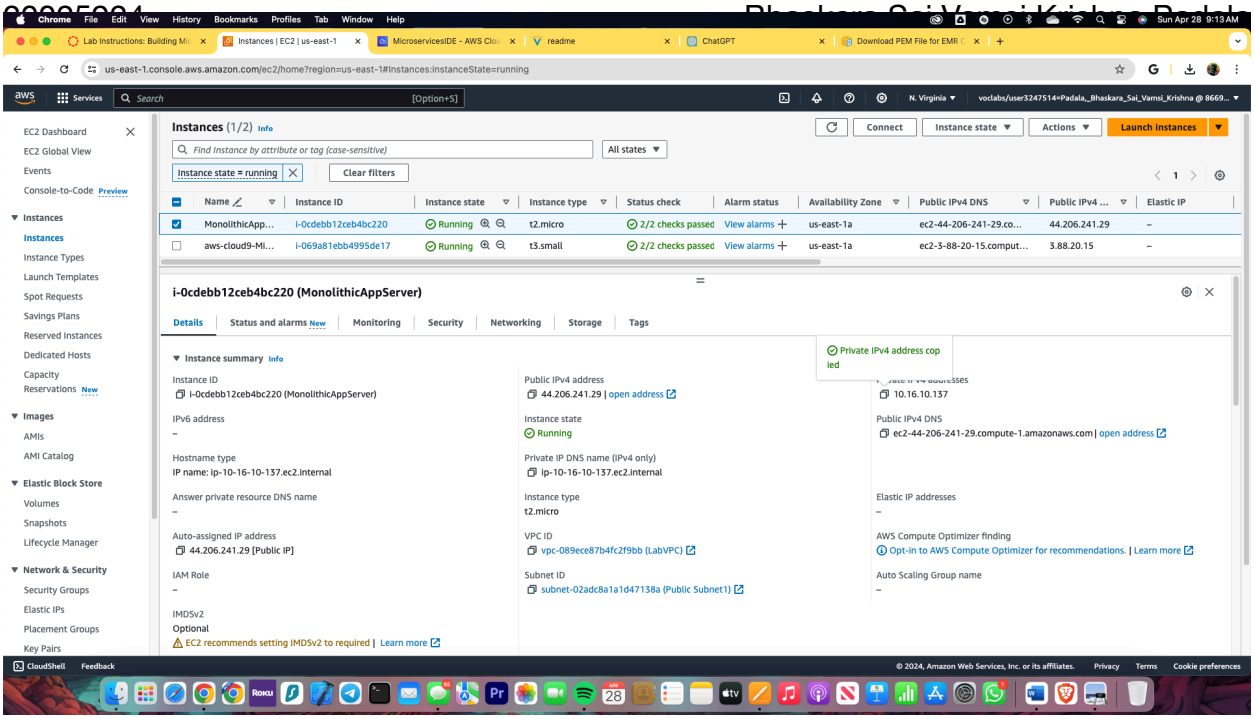
In the Cloud9 IDE, locate the file explorer panel, usually on the left side of the IDE. Click on the folder where you want to upload the .pem file or create a new folder if necessary. Drag and drop the .pem file from your local computer to the appropriate folder in the Cloud9 IDE.



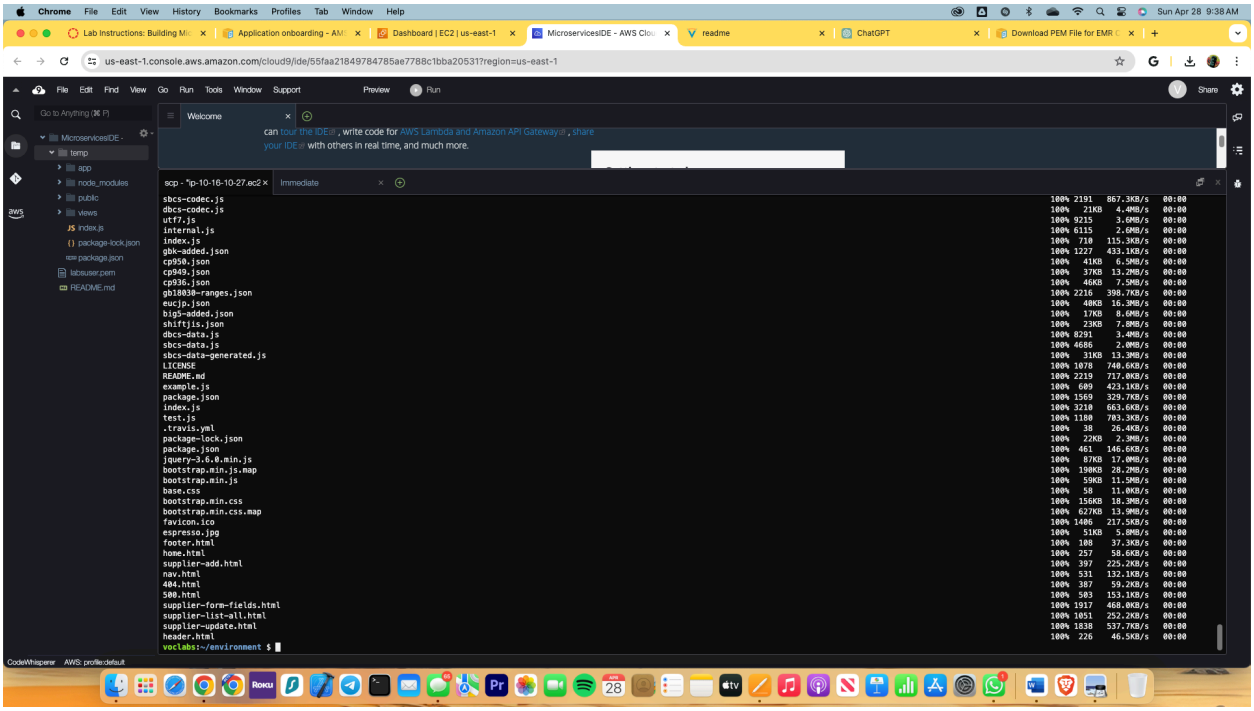
After uploading the labs user.pem file, we need to now perform the below commands in the command prompt as shown in the figure below.



Now copy the Private IPv4 address of Monolithic App Server. From the Amazon EC2 console, retrieve the private IPv4 address of the MonolithicAppServer instance



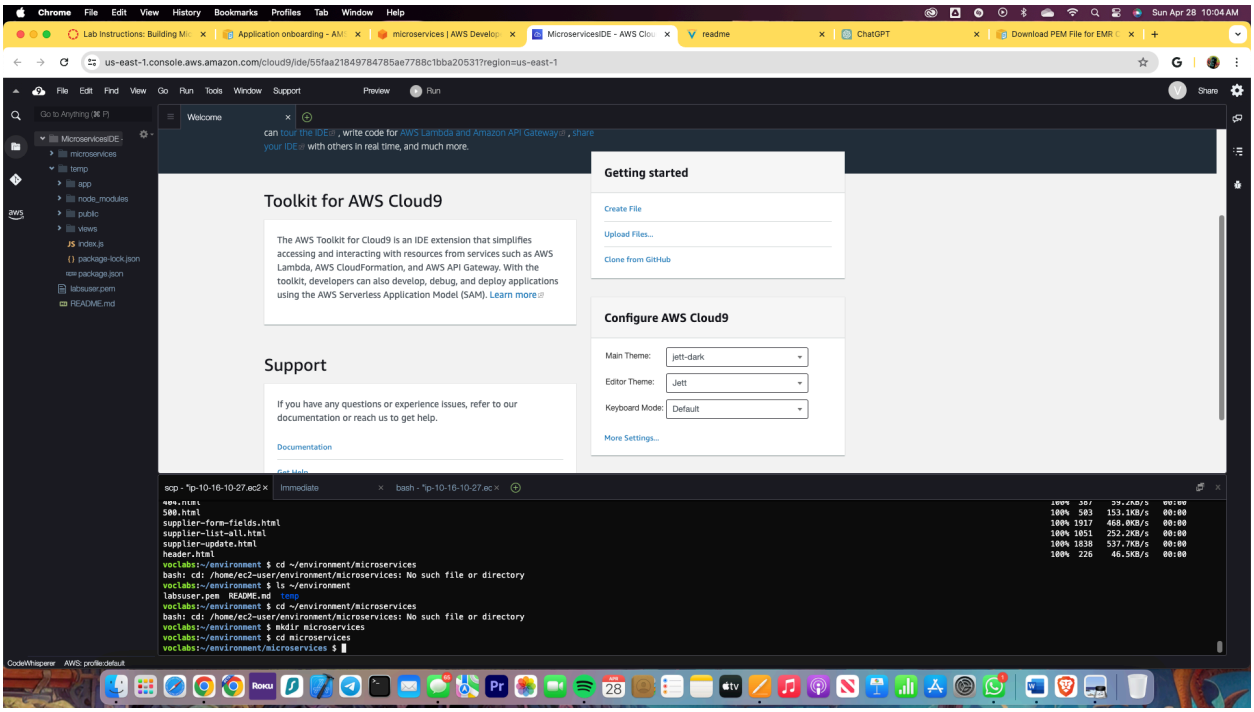
Now after copying and running the following command below, we get the screen showing that we downloaded all these files.
scp -r -i ~/environment/labsuser.pem ubuntu@10.16.10.137:/home/ubuntu/resources/codebase_partner/* ~/environment/temp/



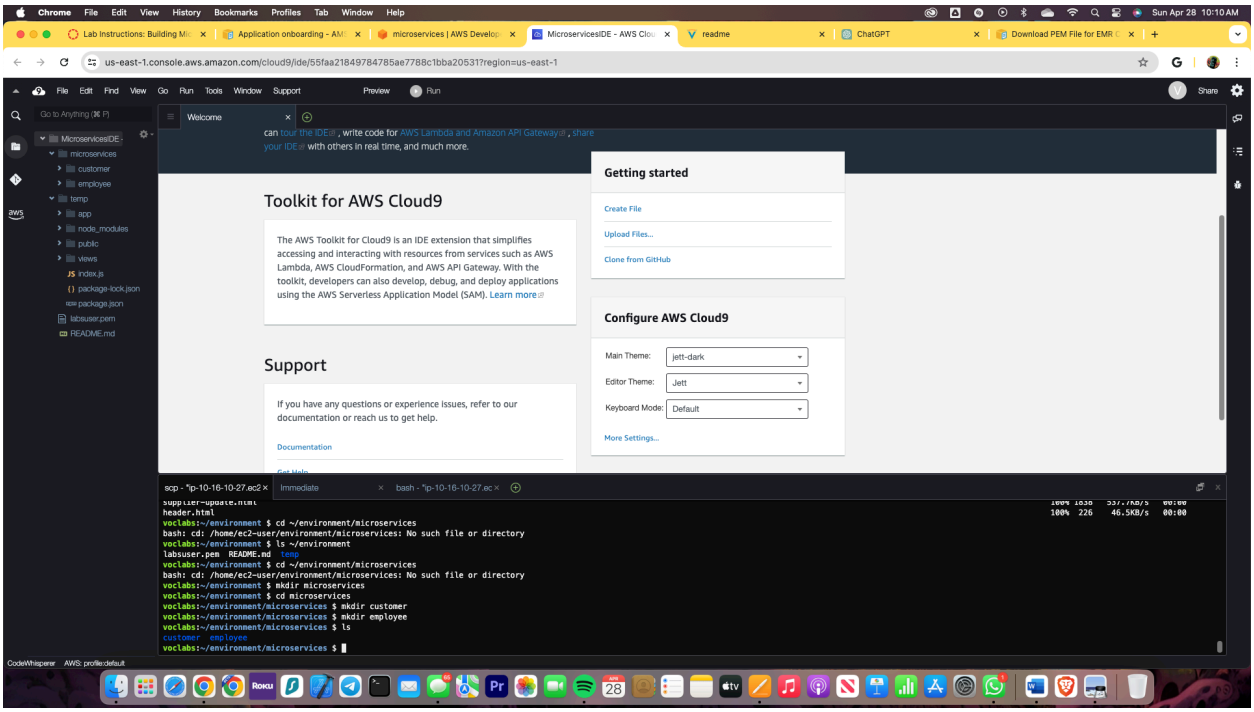
In the file browser of the IDE, we verified that the source files for the application have been copied to the temp directory on the AWS Cloud9 instance.

Task 3.3: Create working directories with starter code for the two microservices.

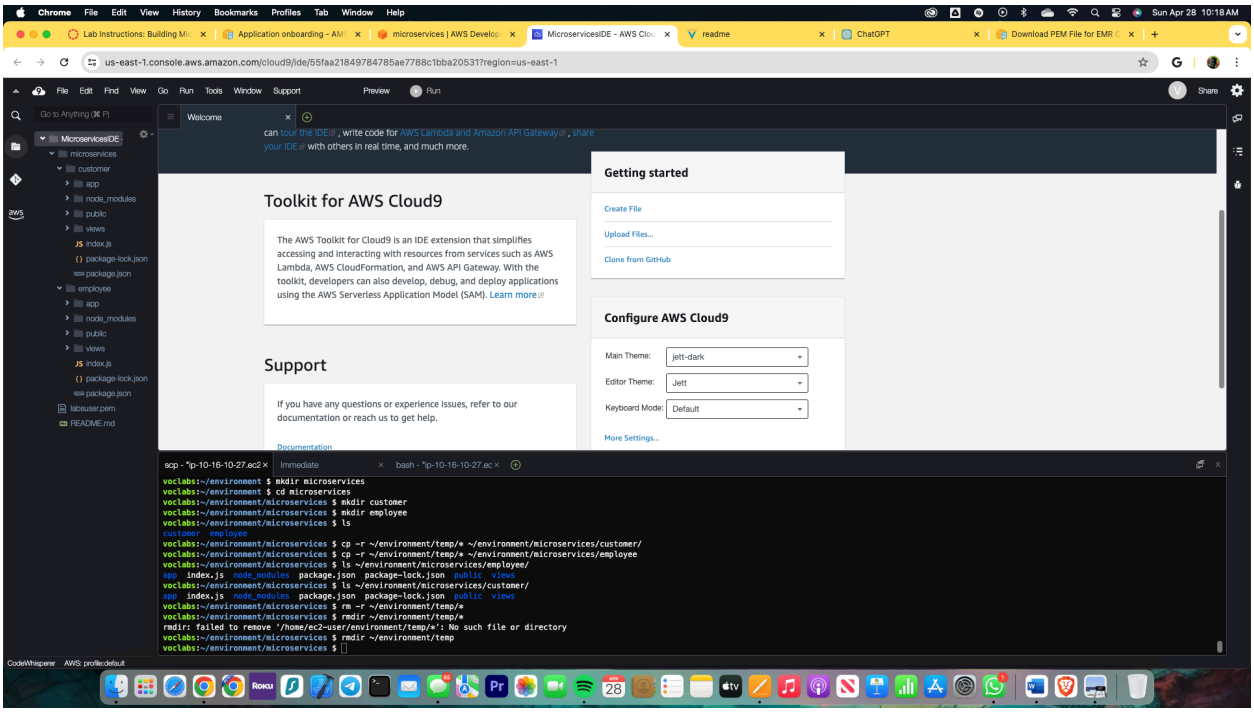
Now we created micro services directory in command prompt.



Now we've created the customer and employee directories in the micro services directory.

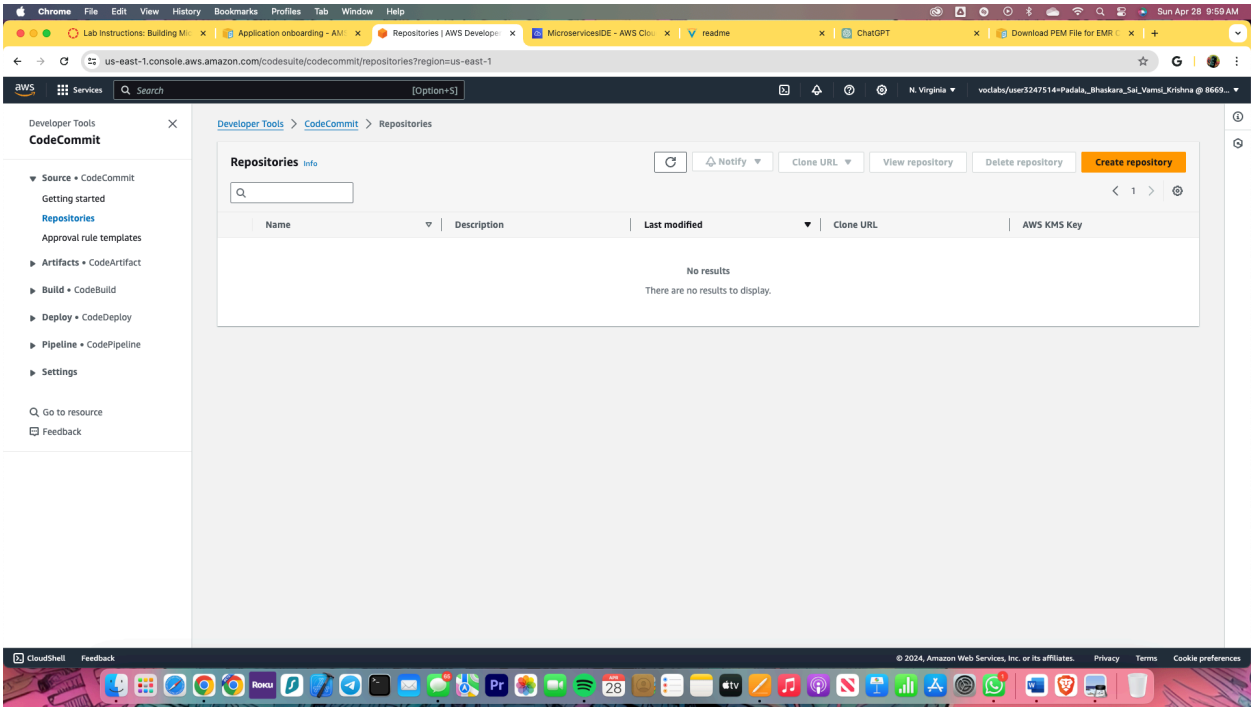


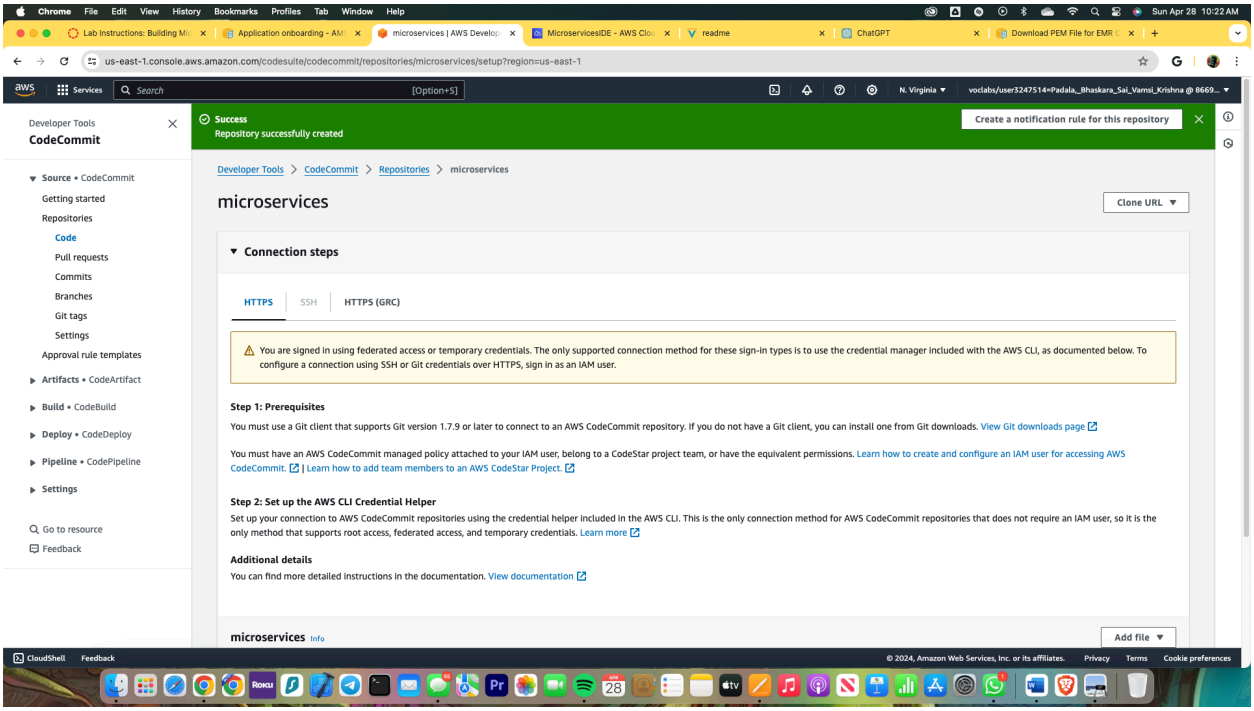
Now we will move the files so that they will be in the same format as given in the instructions . So that we will get the following output.

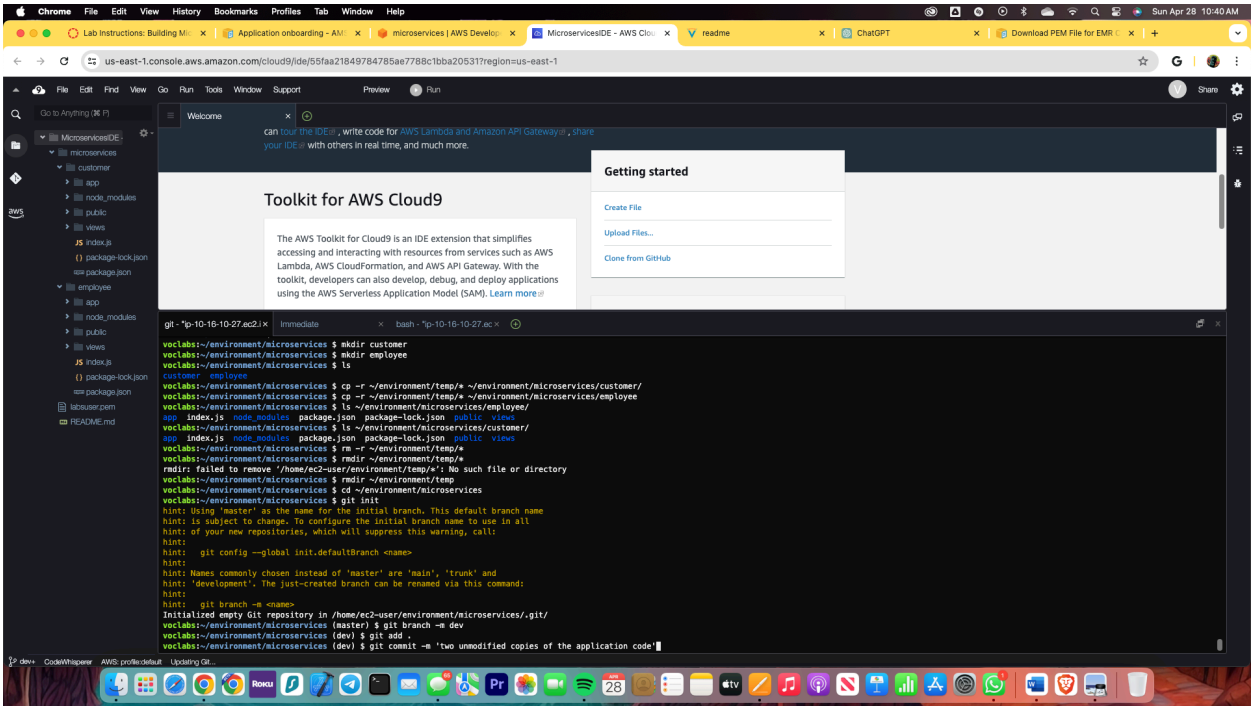


Task 3.4: Create a Git repository for the microservices code and push the code to CodeCommit

Create repository in CodeCommit.

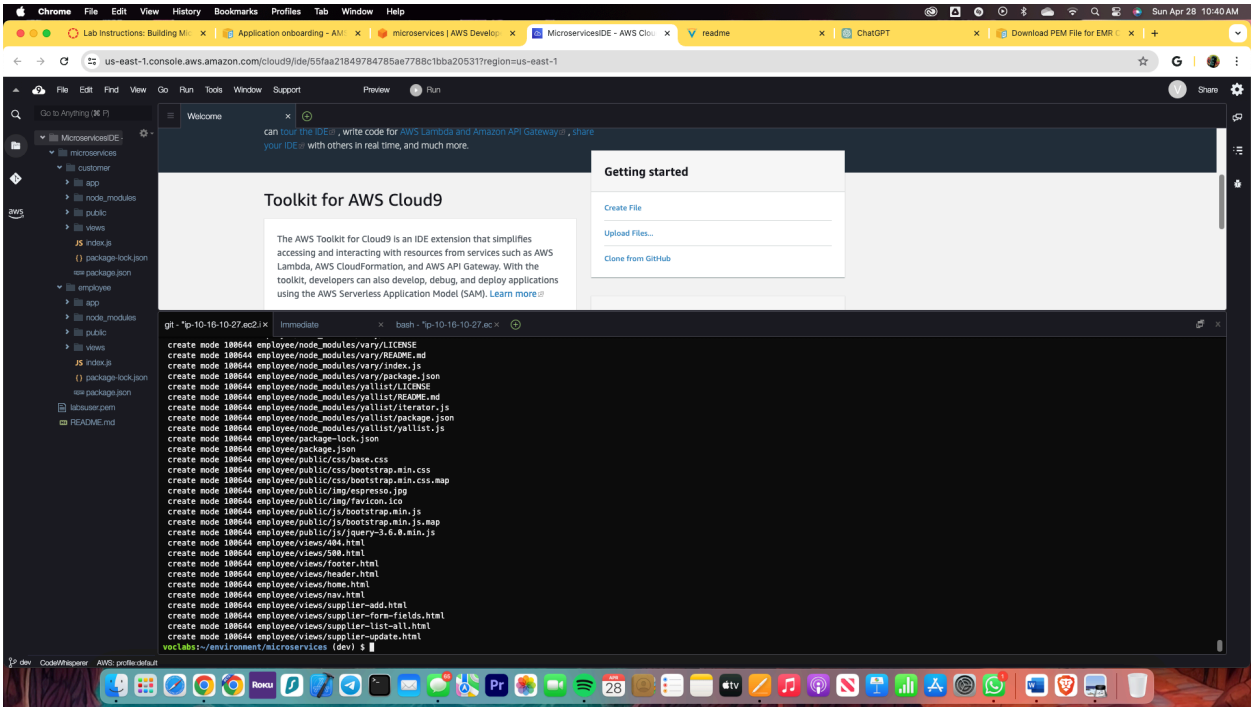






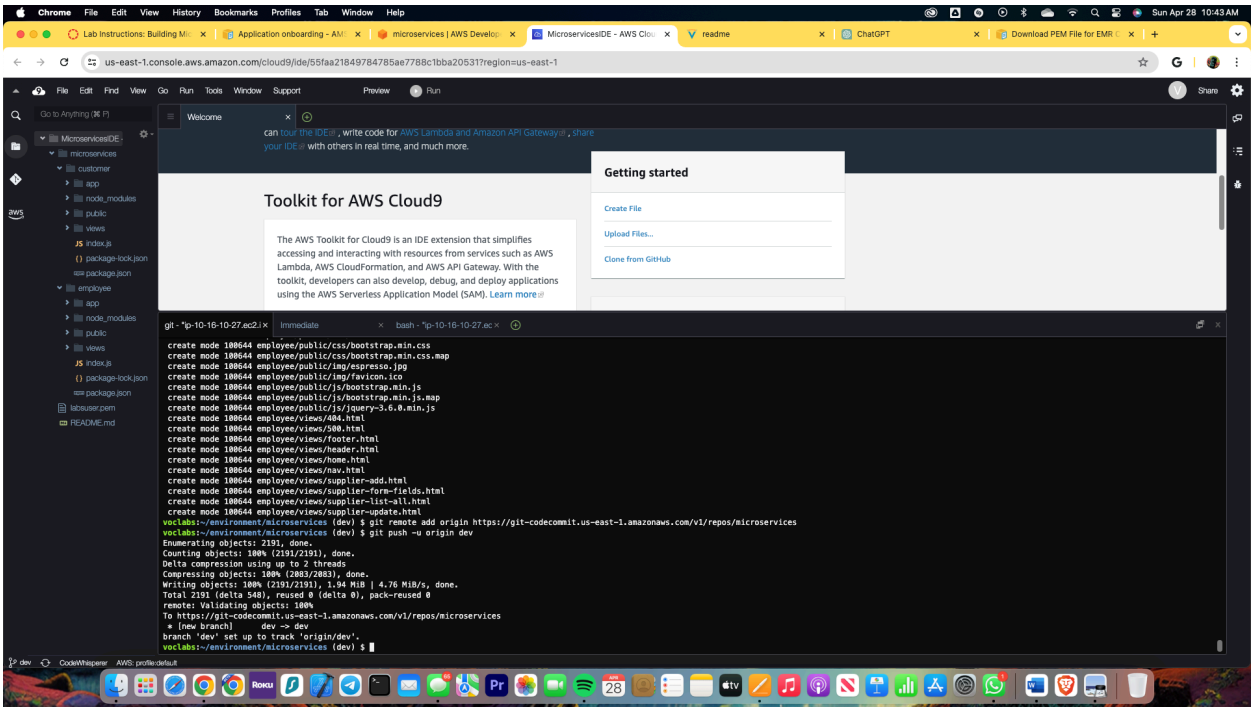
To check the unmodified application code into the microservices CodeCommit repository, we will run the following commands:

cd ~/environment/microservices
git init
git branch -m dev
git add .
git commit -m 'two unmodified copies of the application code'



Now we will run these following commands.

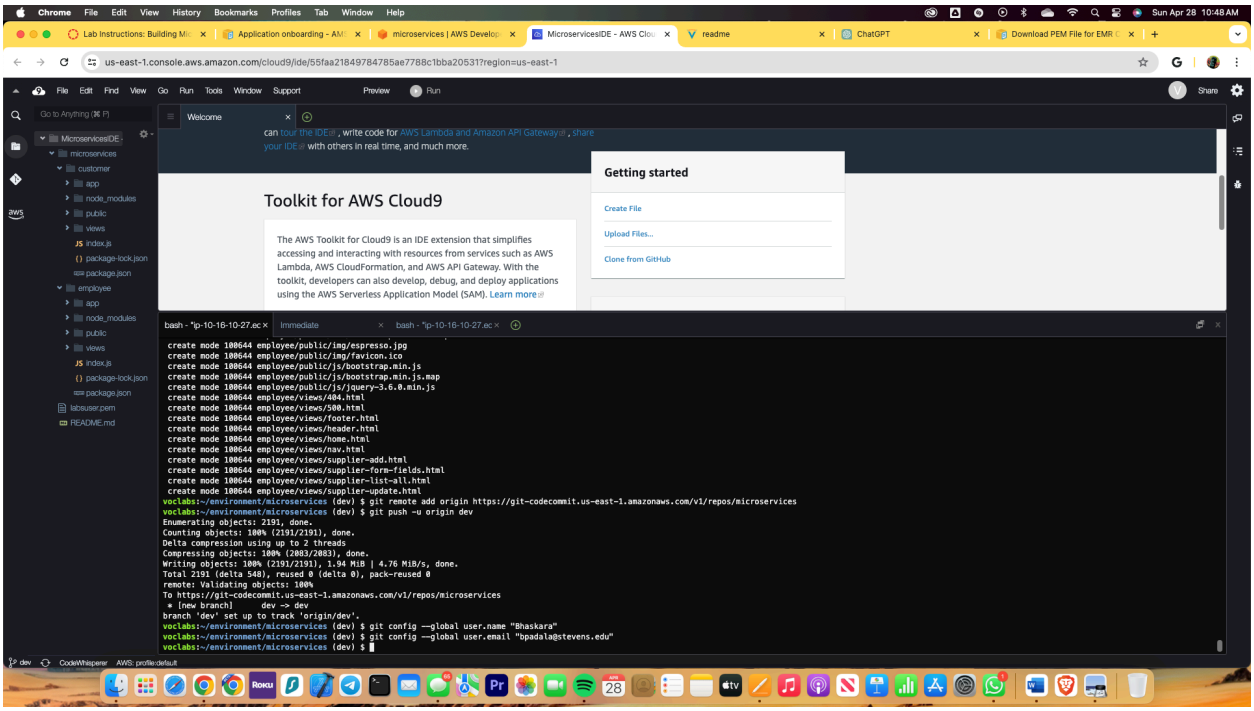
git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
git push -u origin dev

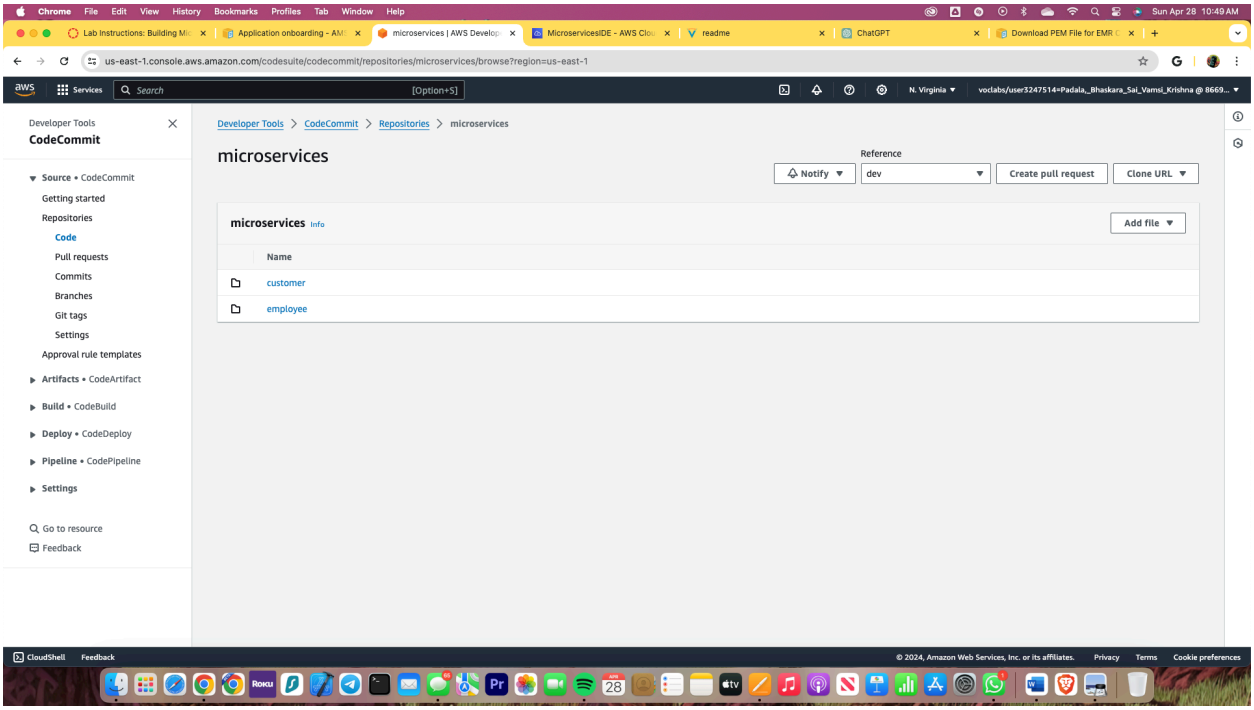


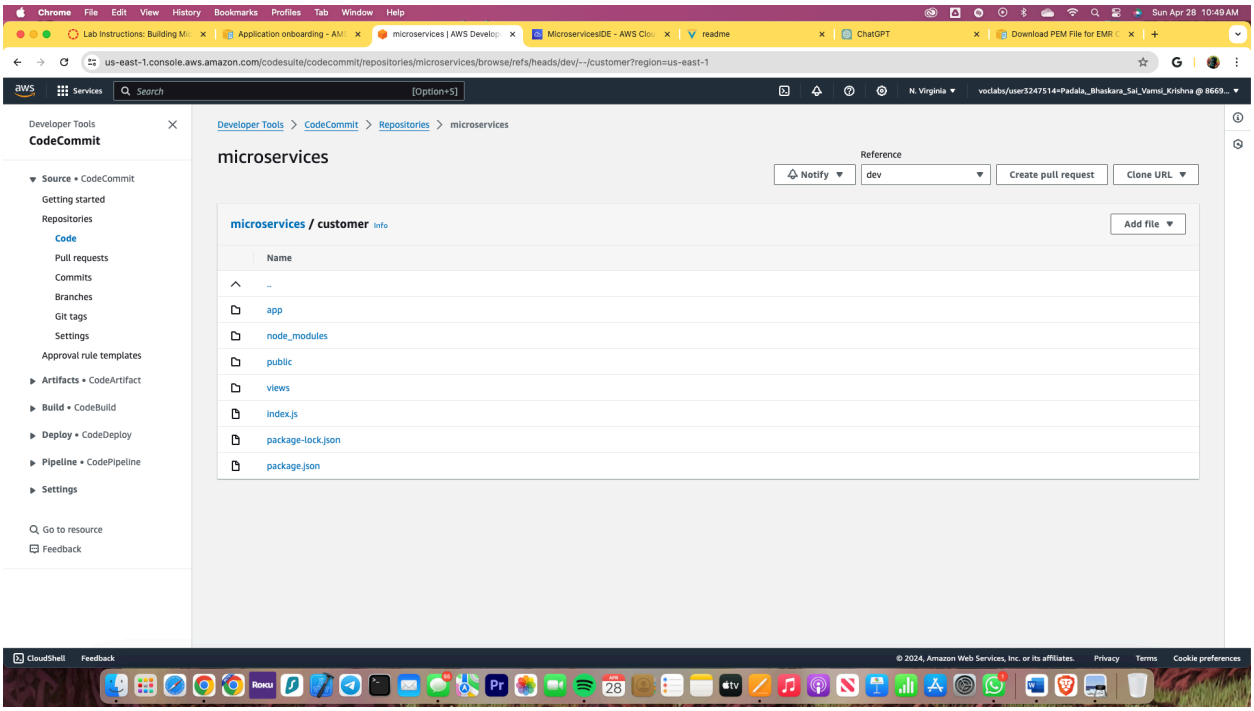
By running these commands, we first initialized the microservices directory to be a Git repository. Then, we created a branch in the repository named dev. We added all files from the microservices directory to the Git repository and committed them. Then, we defined the

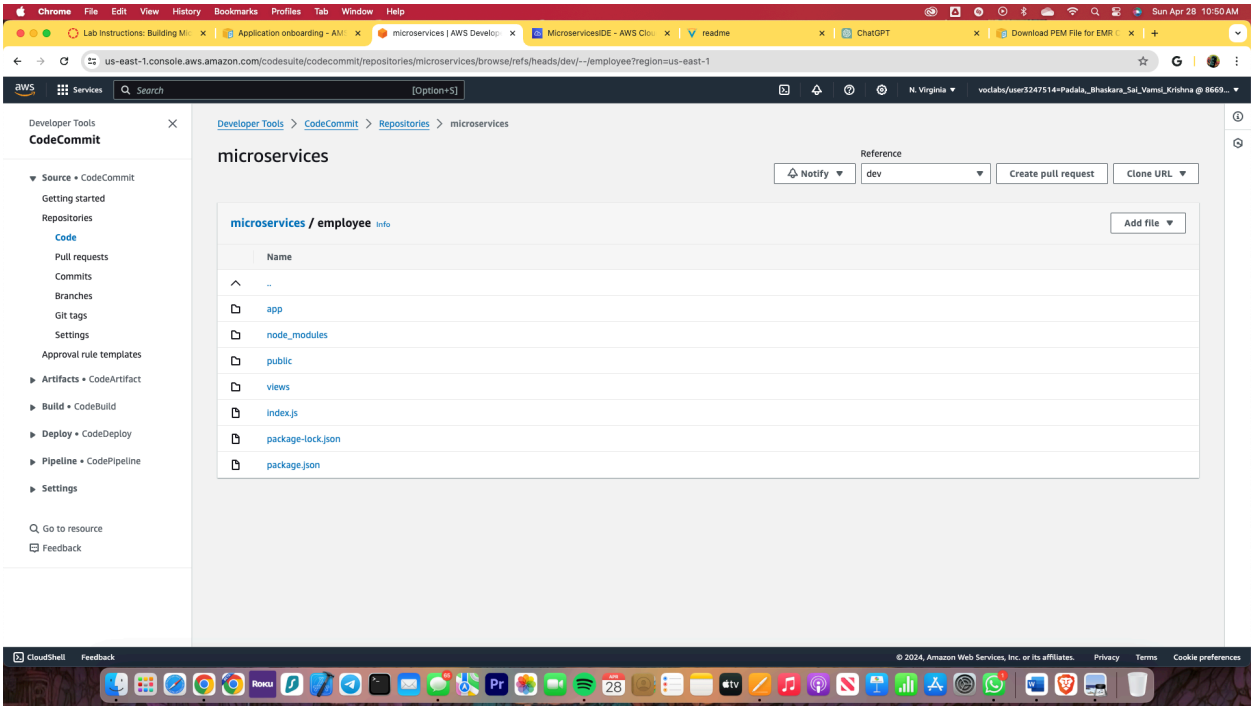
microservices repository that you created in CodeCommit as the remote origin of this Git repository area on our IDE. Finally, we pushed the changes that were committed in the dev branch to the remote origin.

Now we will configure our Git client to know our username and email address.





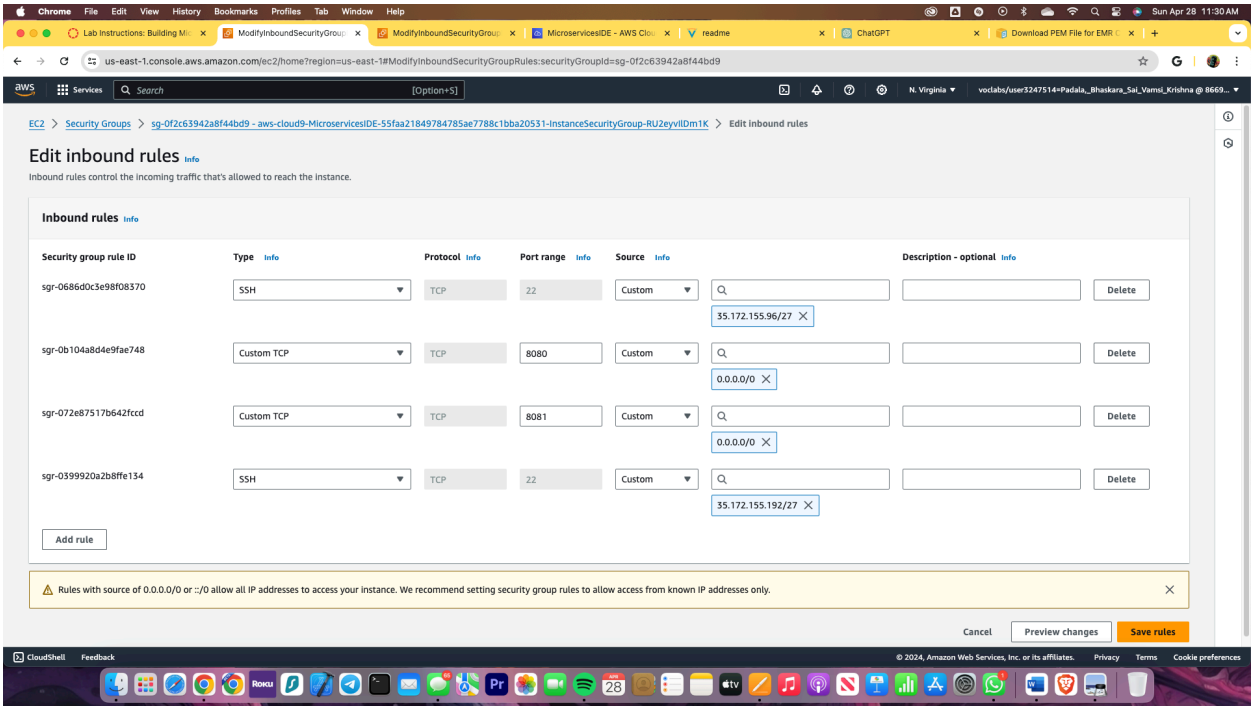




Hence, the code is pushed correctly and we did this task successfully.

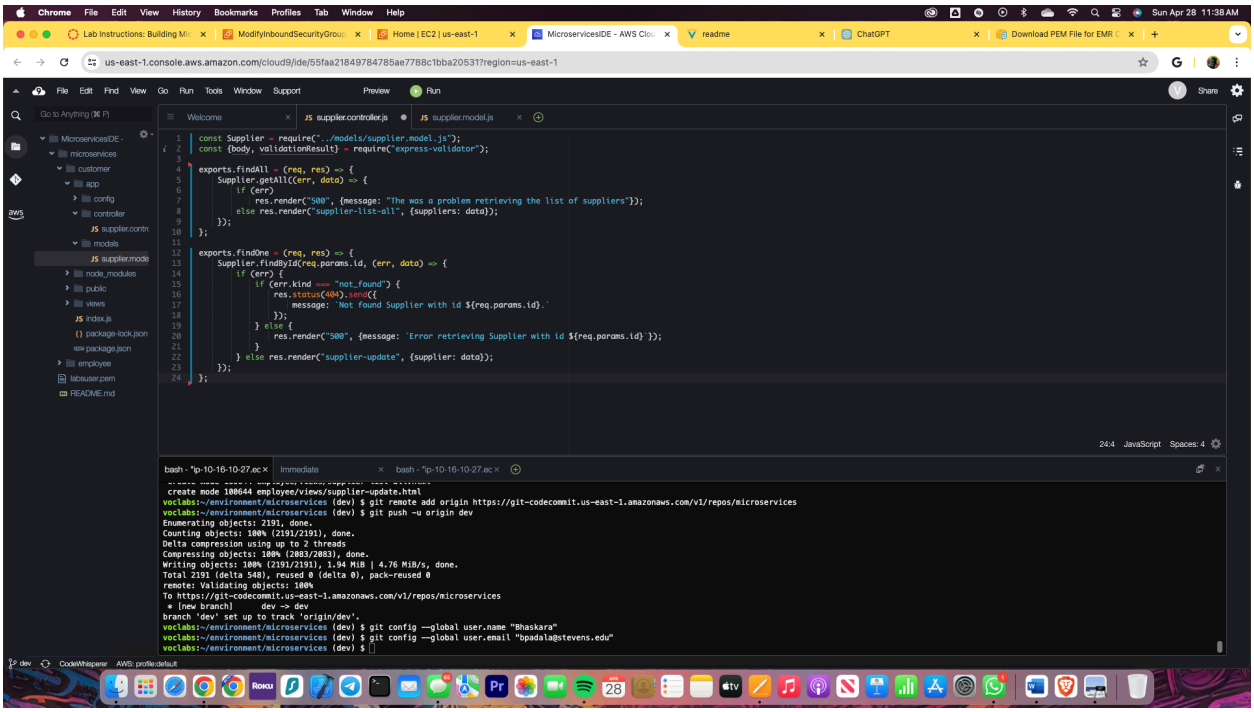
Task 4.1: Adjust the AWS Cloud9 instance security group settings

Now we will adjust the security group of the AWS Cloud9 EC2 instance to allow inbound network traffic on TCP ports 8080 and 8081.



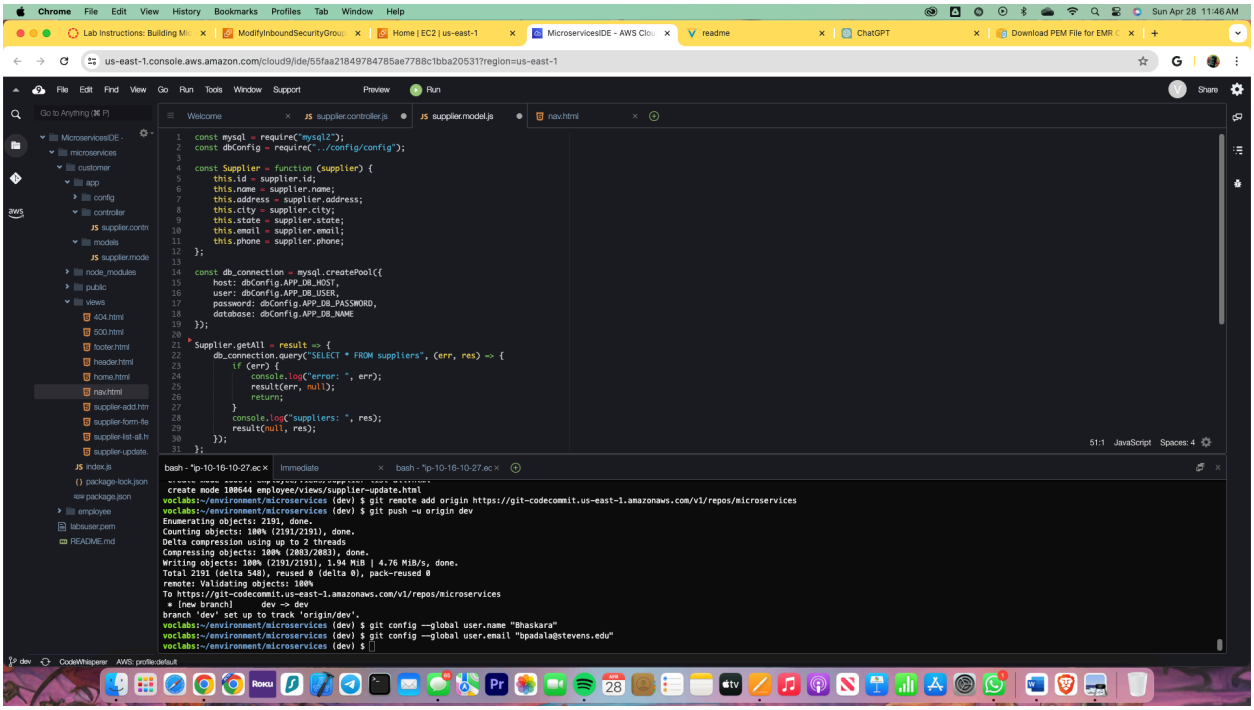
Task 4.2: Modify the source code of the customer microservice

In this task, we will edit the source code for the customer microservice. Recall that the source code you are starting with is an exact copy of the monolithic application. Therefore, it still has features in it that will be handled by the employee microservice and that you don't want as part of the customer microservice. Specifically, customers shouldn't have the ability to add, edit, or



delete suppliers; therefore, the changes that we make will remove that functionality from the source code. Ideally, the source code should contain only code that is needed.

In the AWS Cloud9 file panel, firstly, collapse the employee directory, if it is expanded, and then expand the customer directory. Now we will edit the customer/app/controller supplier.controller.js file so that the remaining functions provide only the read-only actions that you want customers to be able to perform.



In the above screen, we Edited the customer/app/models/supplier.model.js file. Then we Delete the unnecessary functions in it so that what remains are only read-only functions.

Here we KEEP the last line of the file: module.exports = Supplier;
Also The model has still contain two functions: Supplier.getAll and Supplier.findById.

Now we will proceed to the next step which is Later in the project, when you deploy the microservices behind an Application Load Balancer, you will want employees to be able to navigate from the main customer page to the area of the web application where they can add, edit, or delete supplier entries. To support this, edit the customer/views/nav.html file:

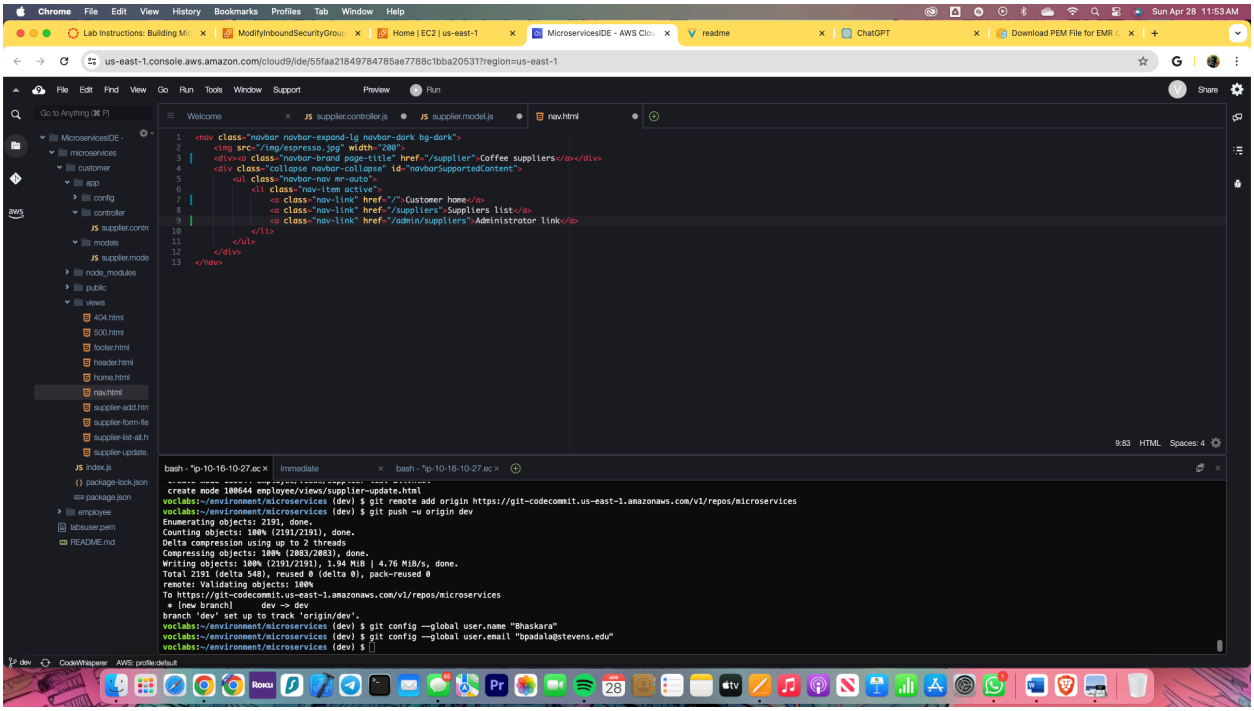
On line 3, we changed Monolithic Coffee suppliers to Coffee suppliers

On line 7, we changed the Home to Customer home

Add a new line after line 8 that contains the following HTML:

Don’t delete or overwrite any of the existing lines in the file.

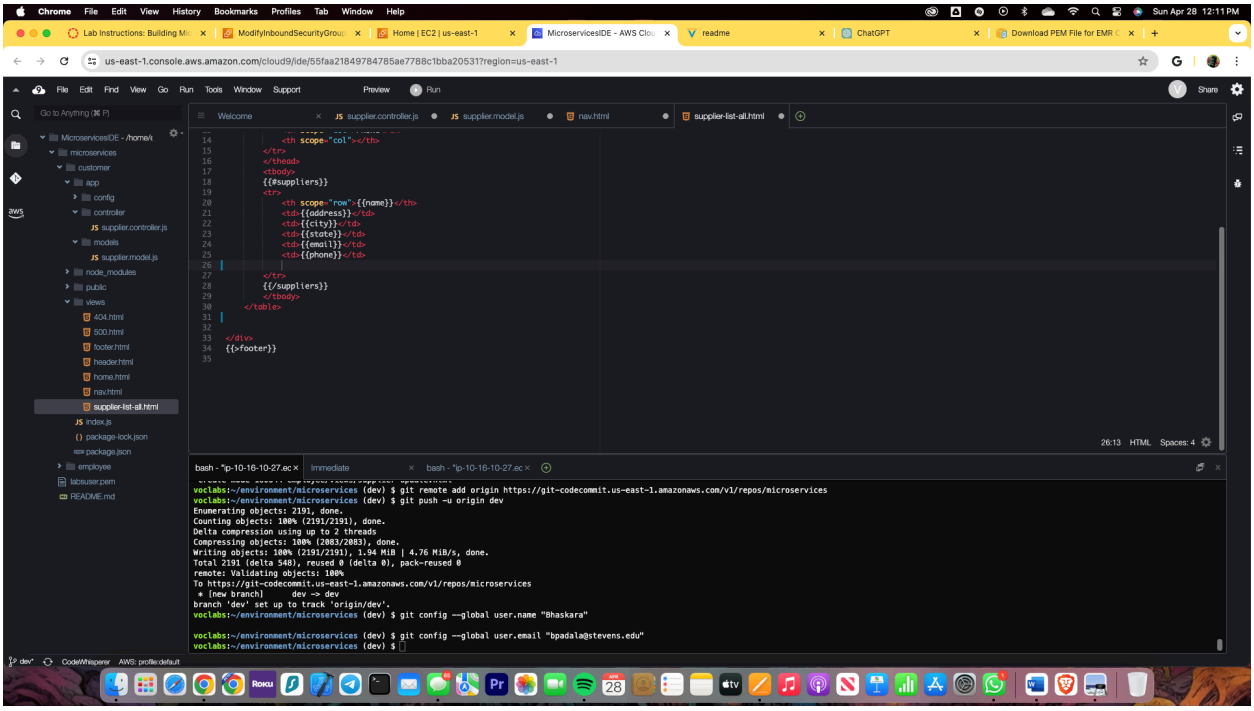
Administrator link



Now we edited the customer/views/supplier-list-all.html file:

Remove line 32, which contains Add a new supplier.

Remove lines 26 and 27, which contain badge badge-info and supplier-update.



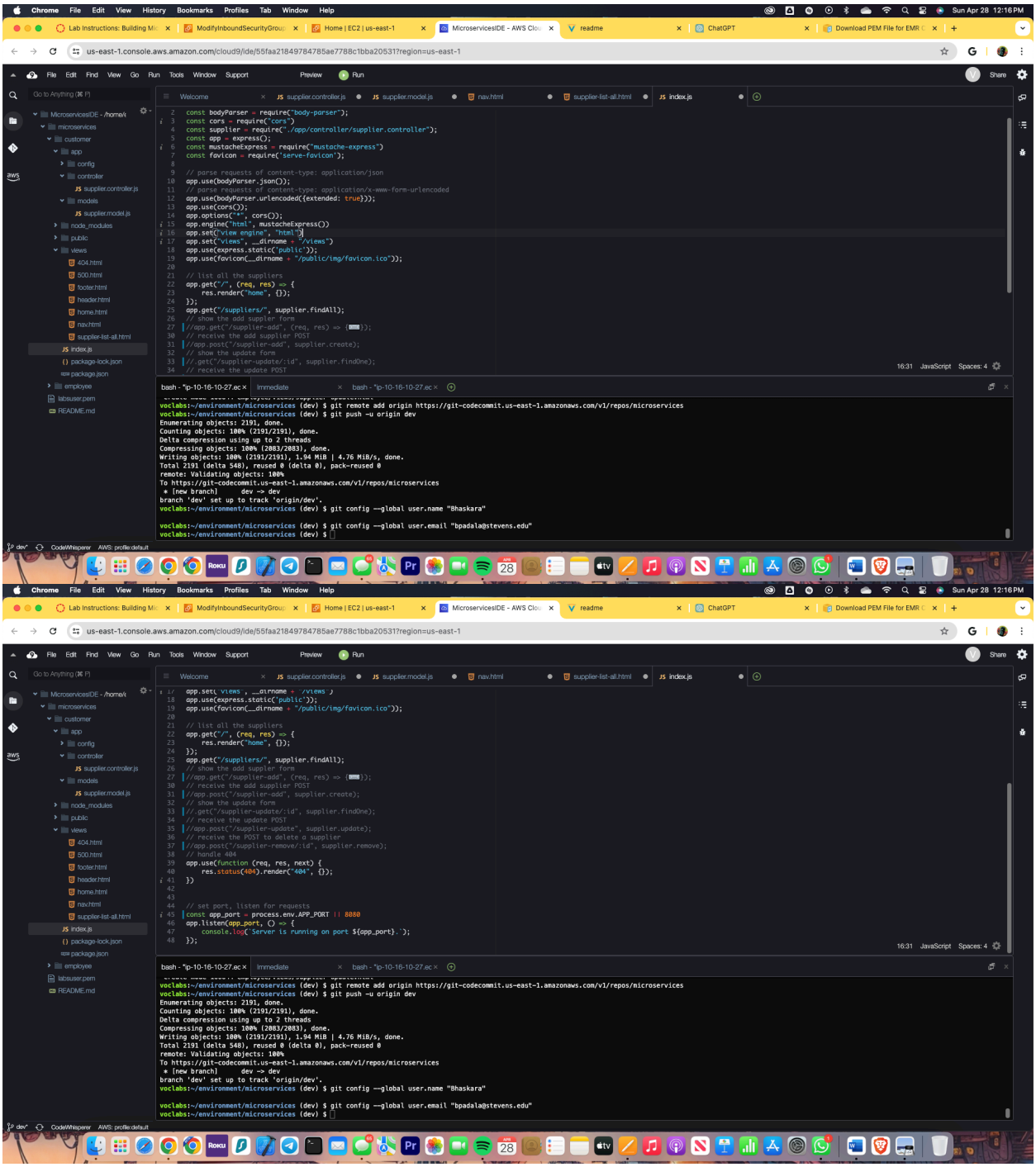
Because the customer microservice doesn't need to support read-write actions, we DELETED the following .html files from the customer/views directory:

supplier-add.html ; supplier-form-fields.html ; supplier-update.html

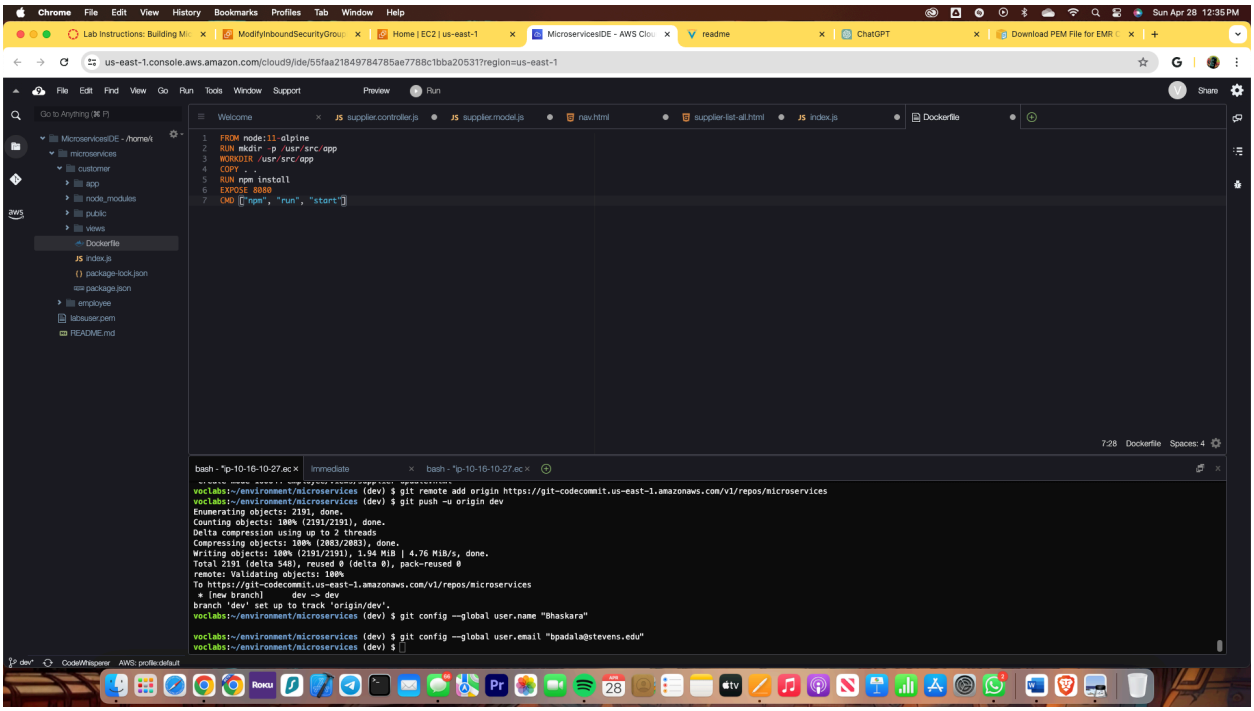
Now we Edited the customer/index.js file as needed to account for the fact that the node application will now run on Docker containers:

Here we Comment out lines 27 to 37 (ensure that each line starts with //).

On line 45, we changed the port number to 8080



Now within the customer directory, we created a new file named **Dockerfile** that contains the following code:

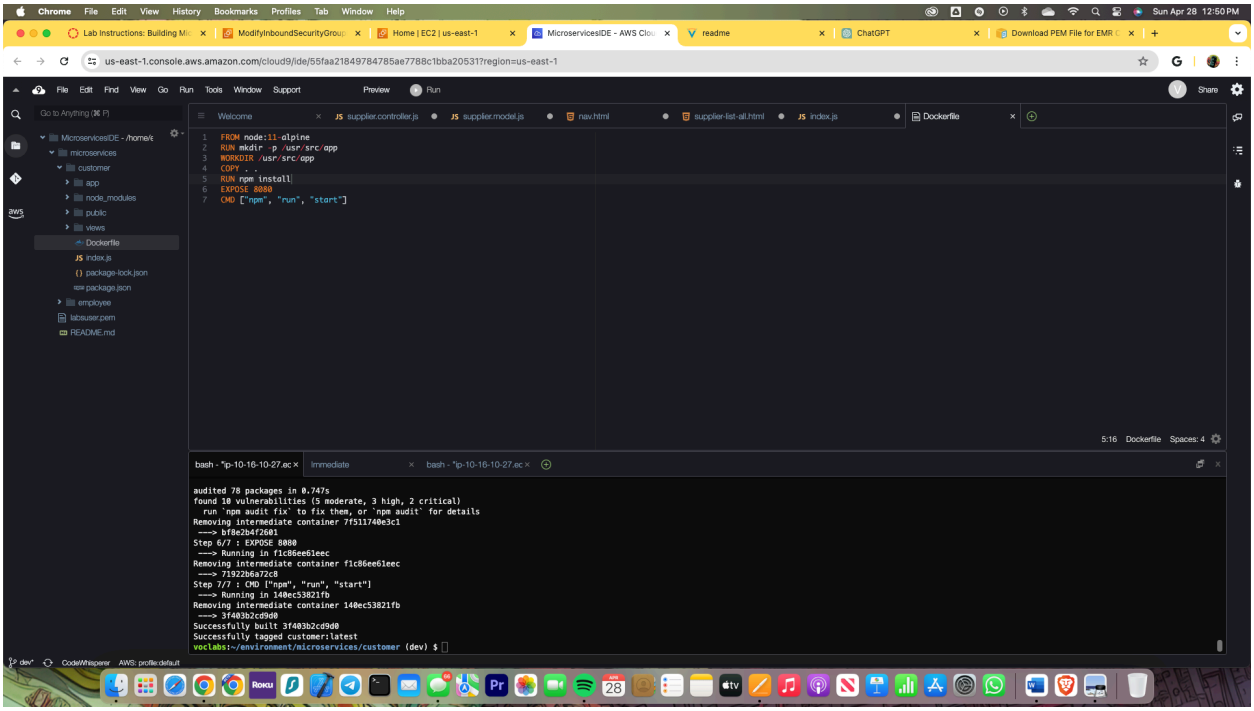


Build an image from the customer Dockerfile.

In the AWS Cloud9 terminal, change to the customer directory.

Run the following command:

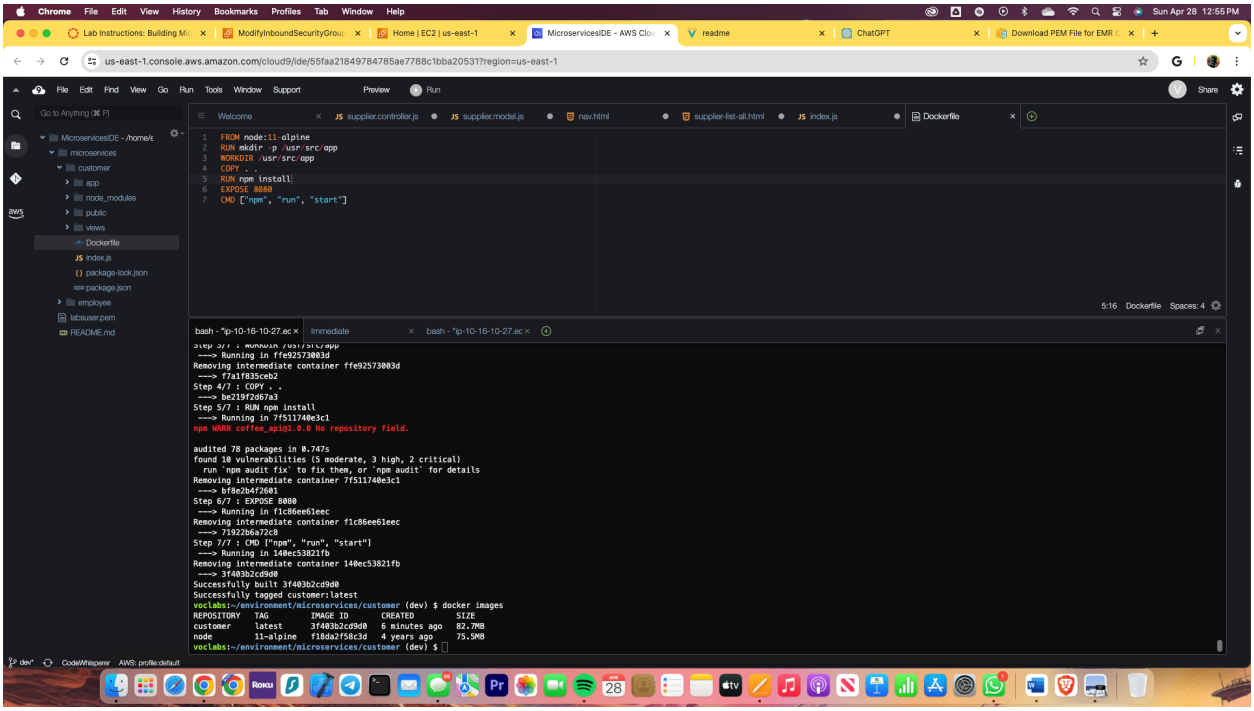
docker build --tag customer .



Verify that the customer-labeled Docker image was created.

Run a Docker command to list the Docker images that your Docker client is aware of.
Here To find the command that you need to run, see Use the Docker Command Line in the Docker documentation. Hence we get the output of the command should be similar to the following:

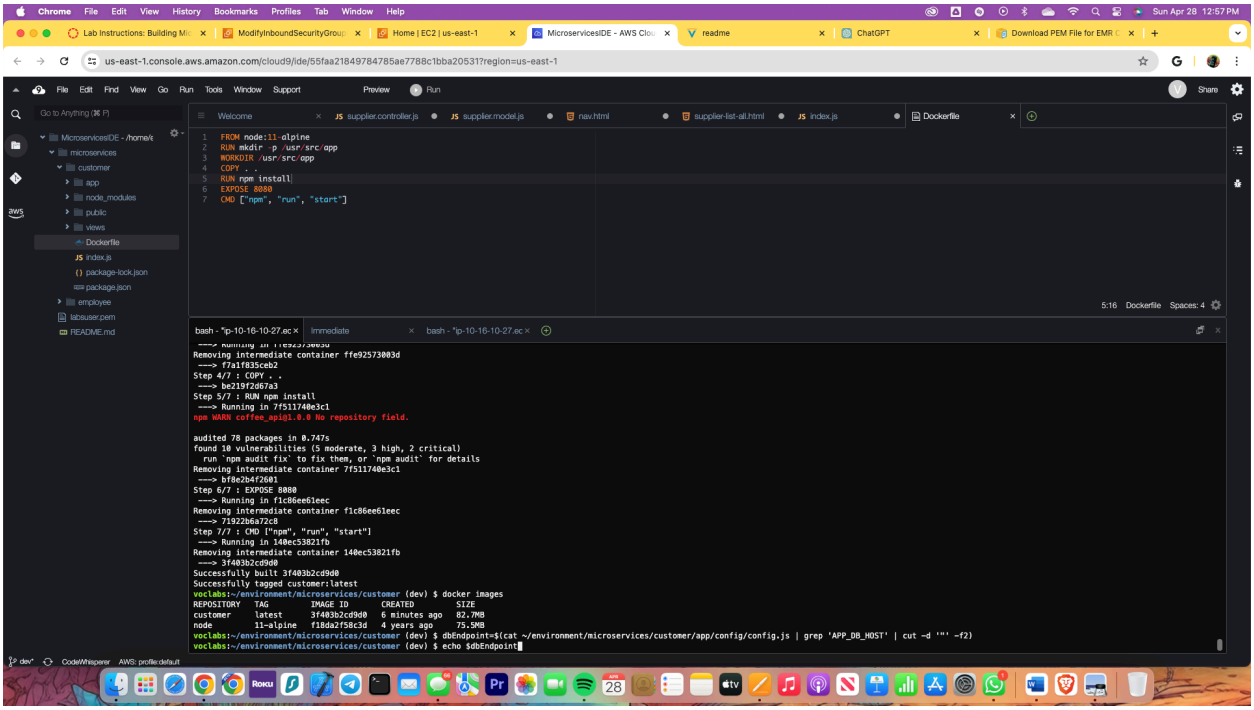
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
customer	latest	cdc593c9bf51	59 seconds ago	82.7MB
node	11-alpine	f18da2f58c3d	3 years ago	75.5MB

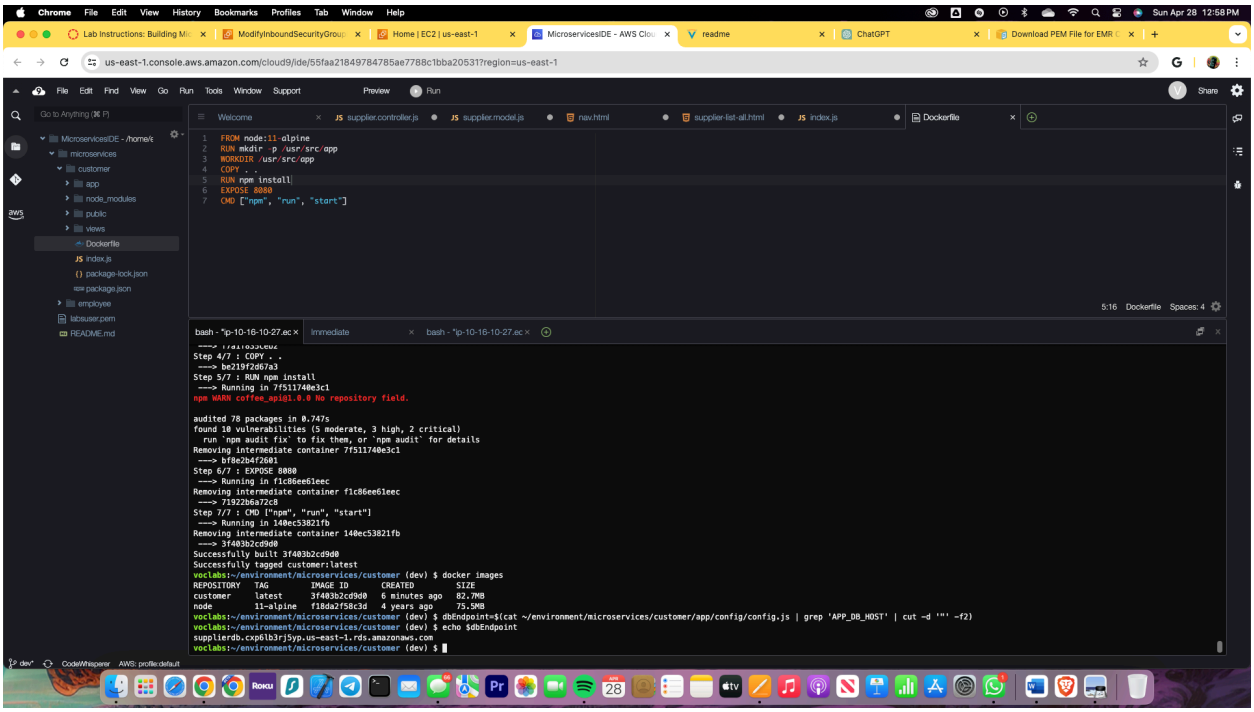


Now we Launch a Docker container that runs the customer microservice on port 8080. As part of the command, pass an environment variable to tell the node application the correct location of the database. To set a dbEndpoint variable in our terminal session, we will run the following commands:

```
dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint
```

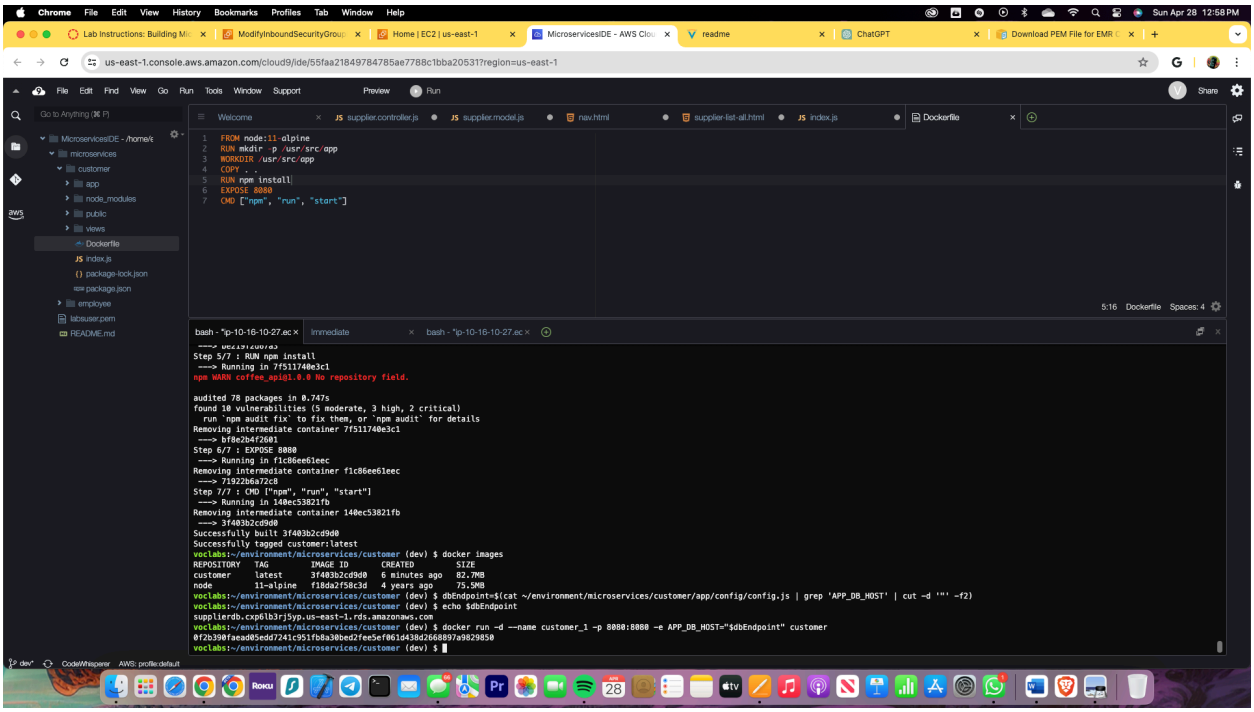
Here we will manually find the database endpoint in the Amazon RDS console and set it as an environment variable by running dbEndpoint="<actual-db-endpoint>" instead of using the cat command.



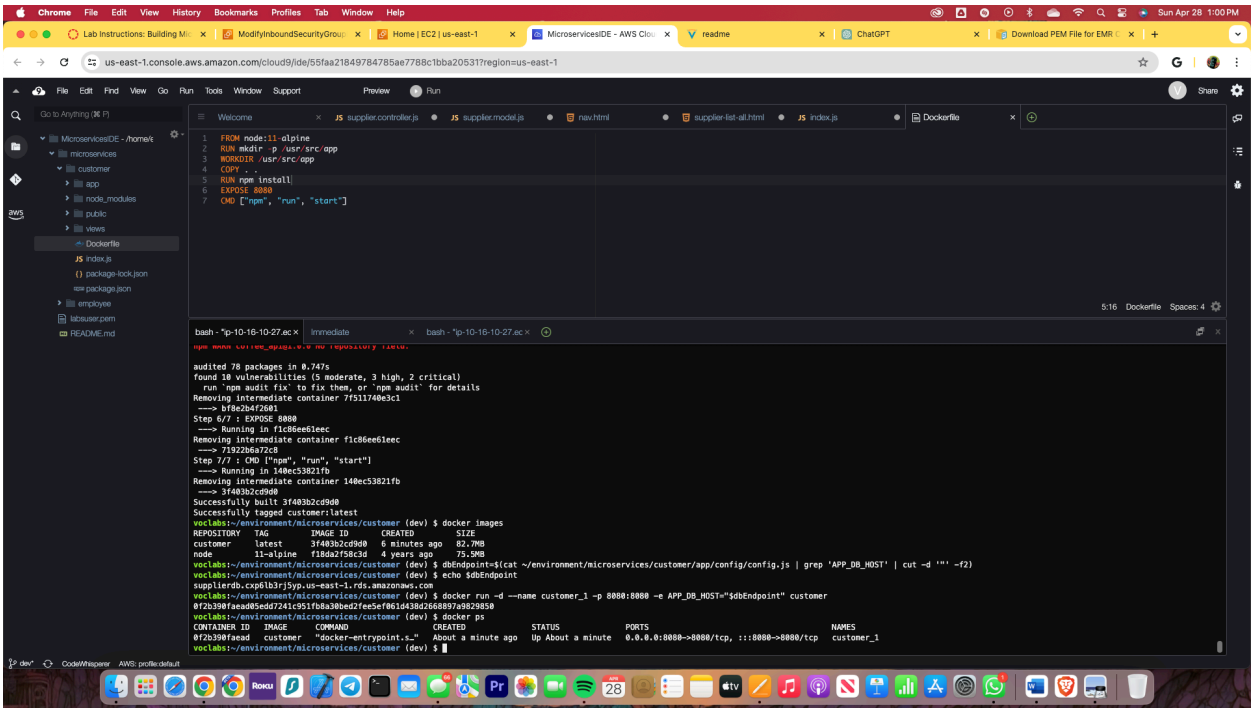


Here we will run the following command :

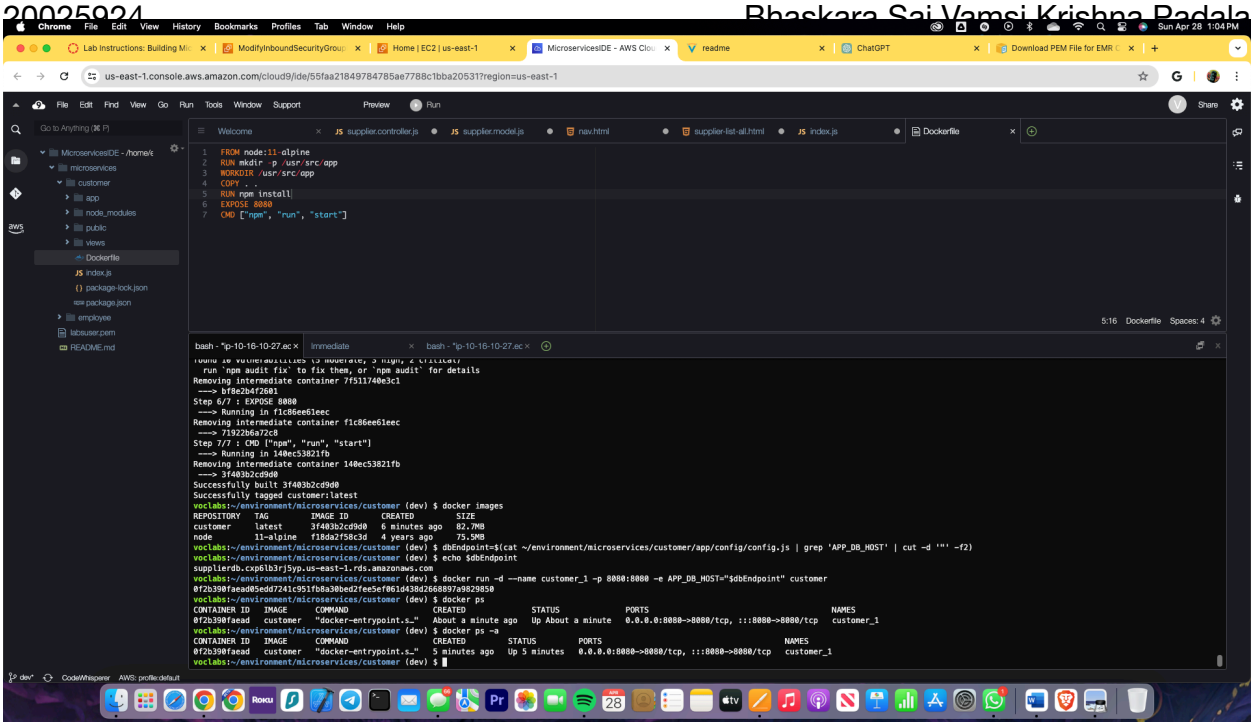
```
docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer
```

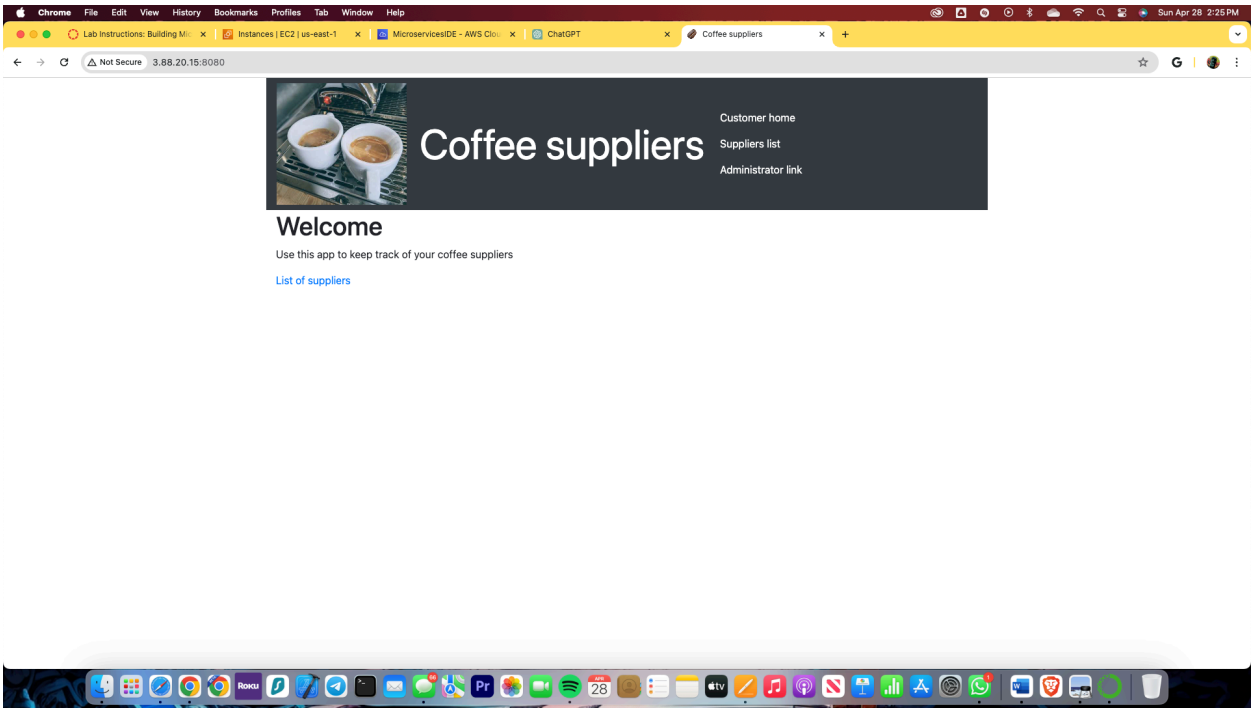


Now we will run the following command :
docker images



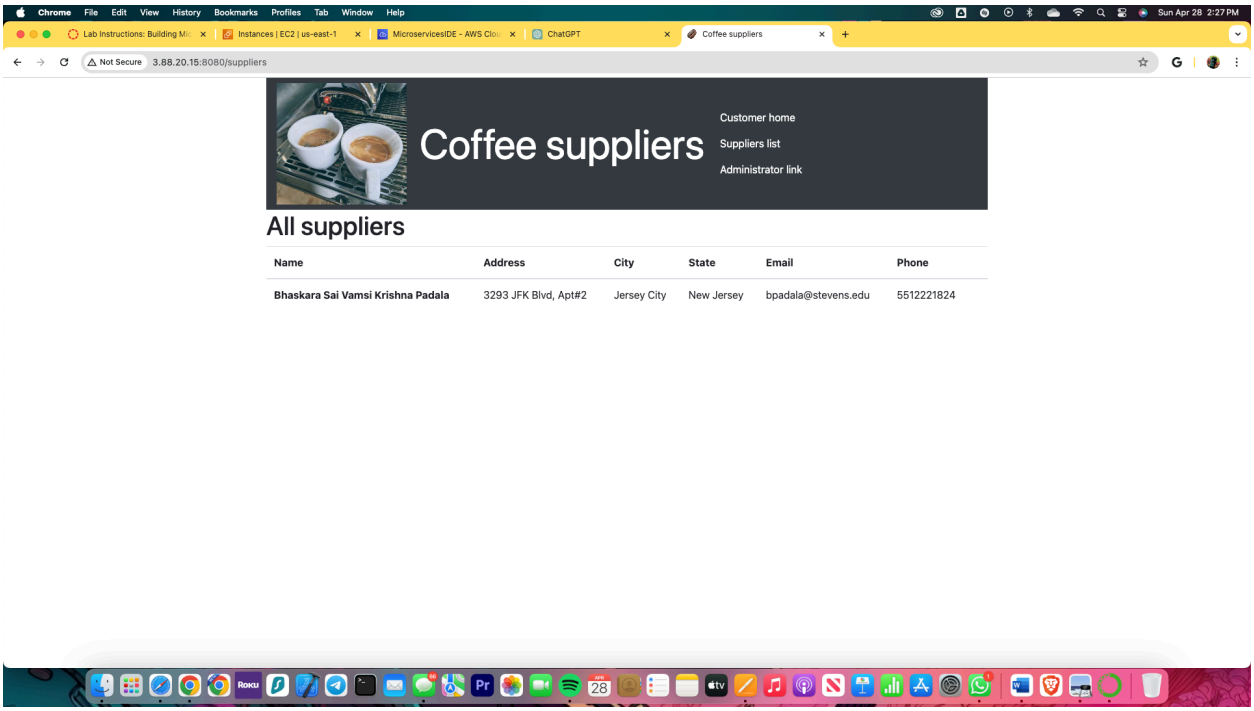
Now we ran the following code in the terminal and we got the following output.
voclabs:~/environment/microservices/customer (dev) $ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
0f2b390faead customer "docker-entrypoint.s…" 5 minutes ago Up 5 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1



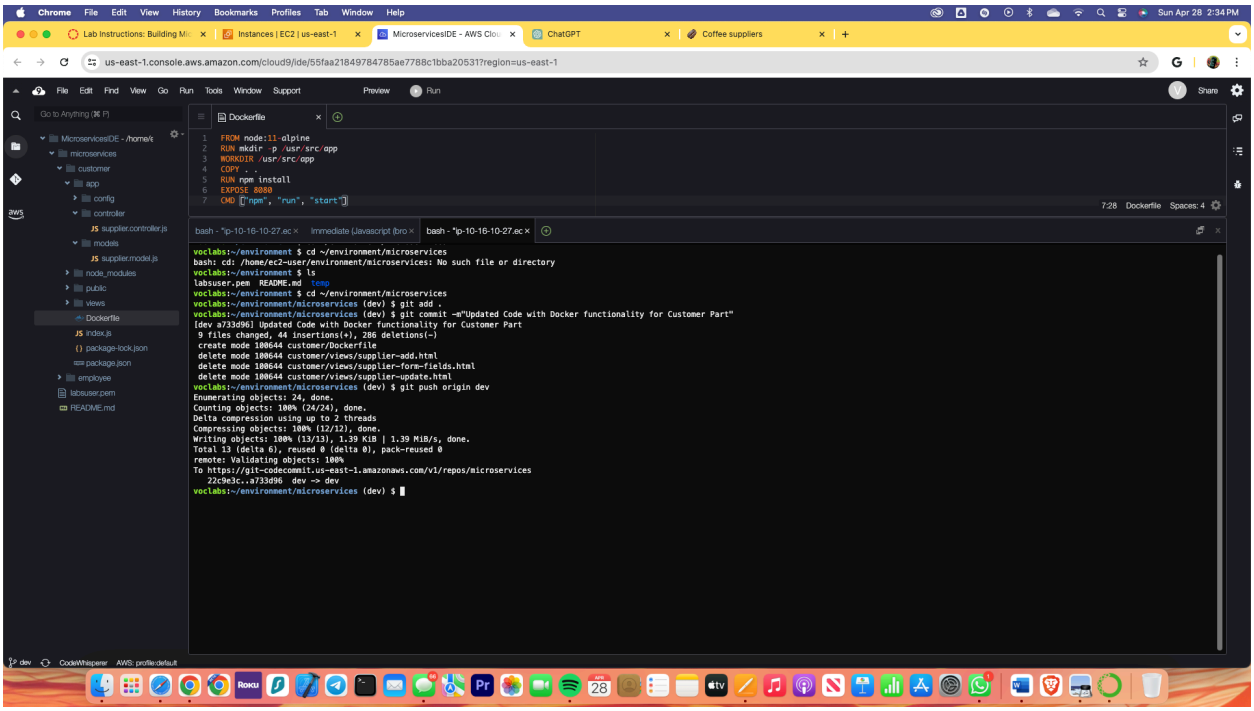


Now we load the following page in a new browser tab. Then we replaced the IP address placeholder with the public IPv4 address of the AWS Cloud9 instance http://3.88.20.15:8080. The web application microservice then got loaded in the browser as shown above.

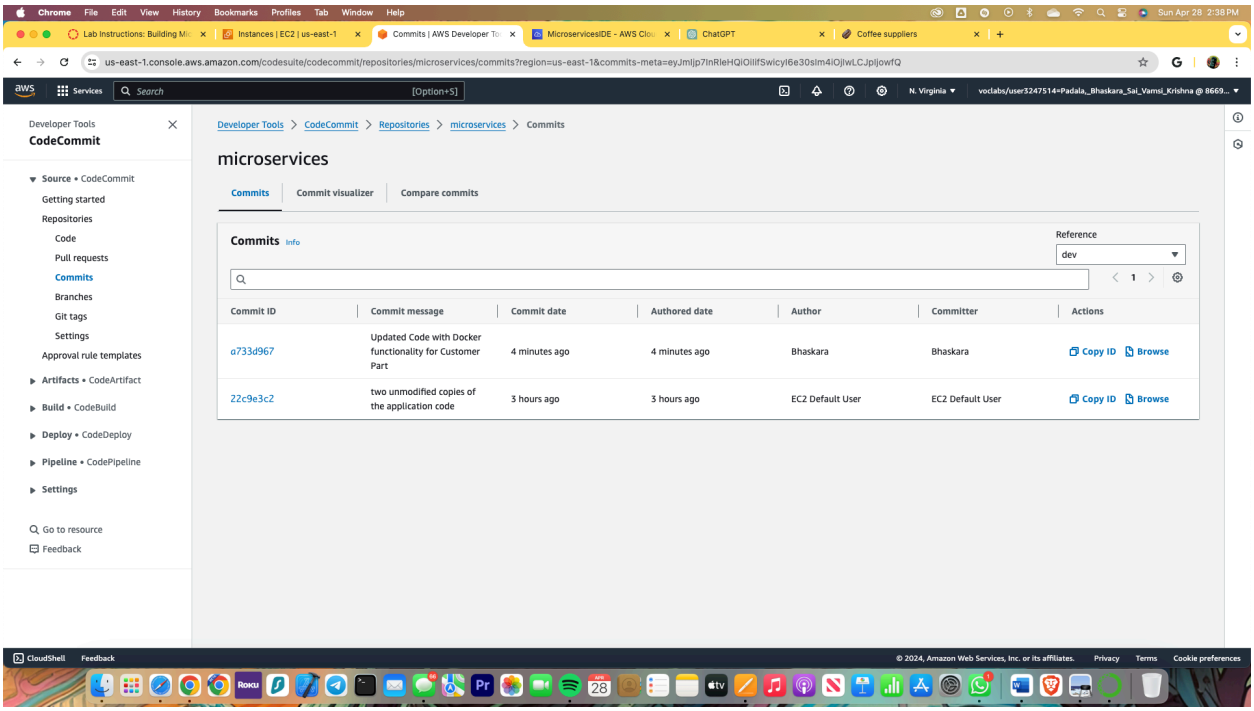
Now click on the list of suppliers and then we'll get the following screen.



Now we will push the the code commits to git as follows:



We can verify our commits as follows:



Task 4.4: Modify the source code of the employee microservice :

The AWS Cloud9 IDE, now return to the file view (toggletree view).

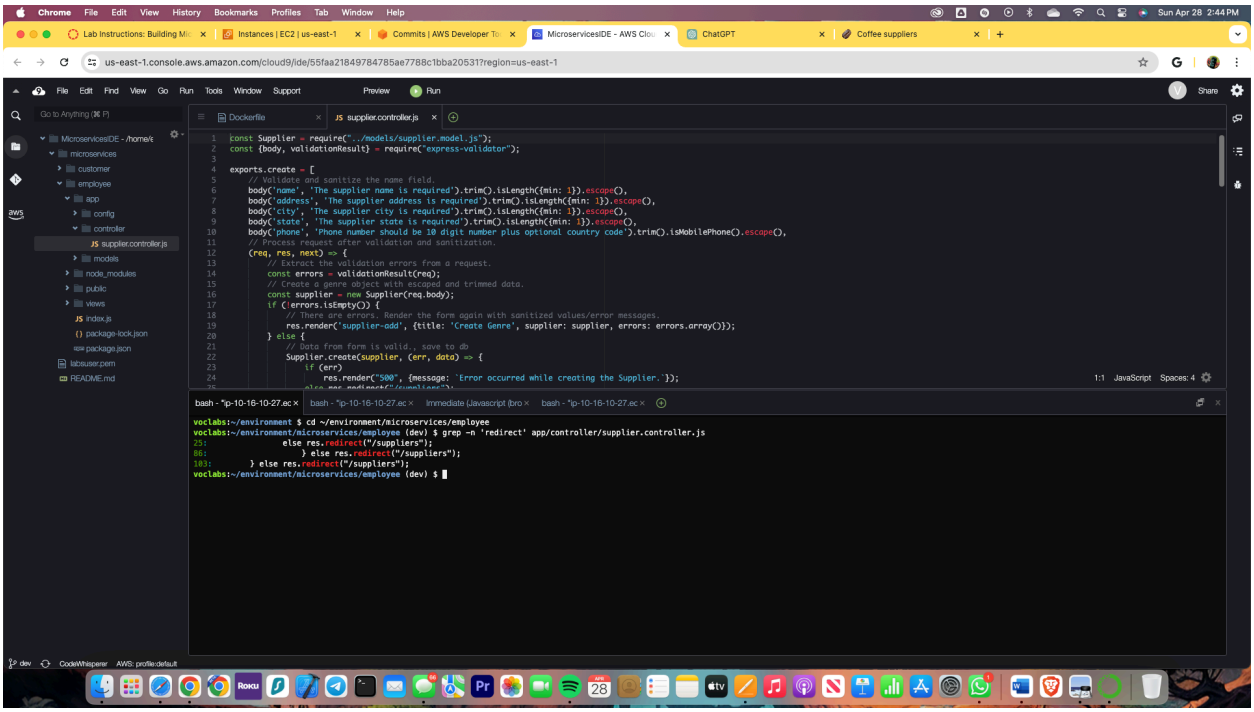
Now we collapsed the customer directory, and then expand the employee directory.

In the employee/app/controller/supplier.controller.js file, for all the redirect calls, we tried to prepend /admin to the path.

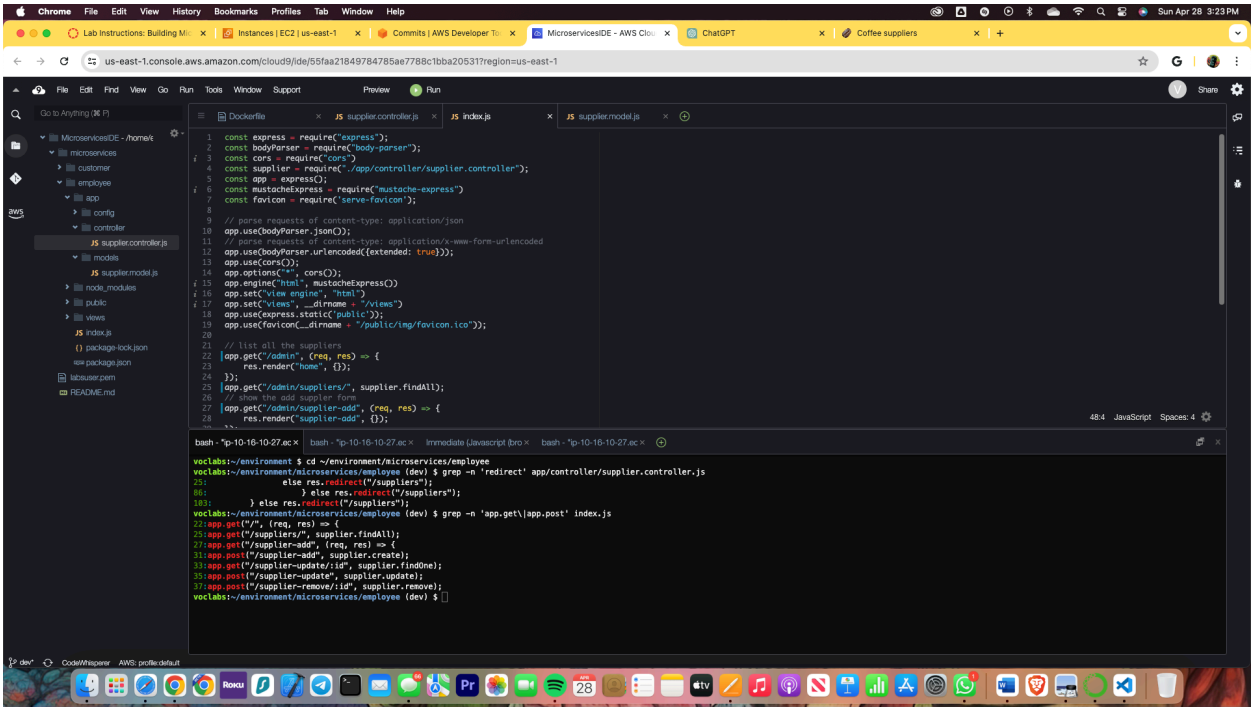
To do the above process, we followed the prompts in the command line argument.

cd ~/environment/microservices/employee
grep -n 'redirect' app/controller/supplier.controller.js

Because of the need to route traffic, much of the work in this task is to configure the employee microservice to add /admin/ to the path of the pages that it serves..
For all app.get and app.post calls, we prepend /admin to the first parameter.



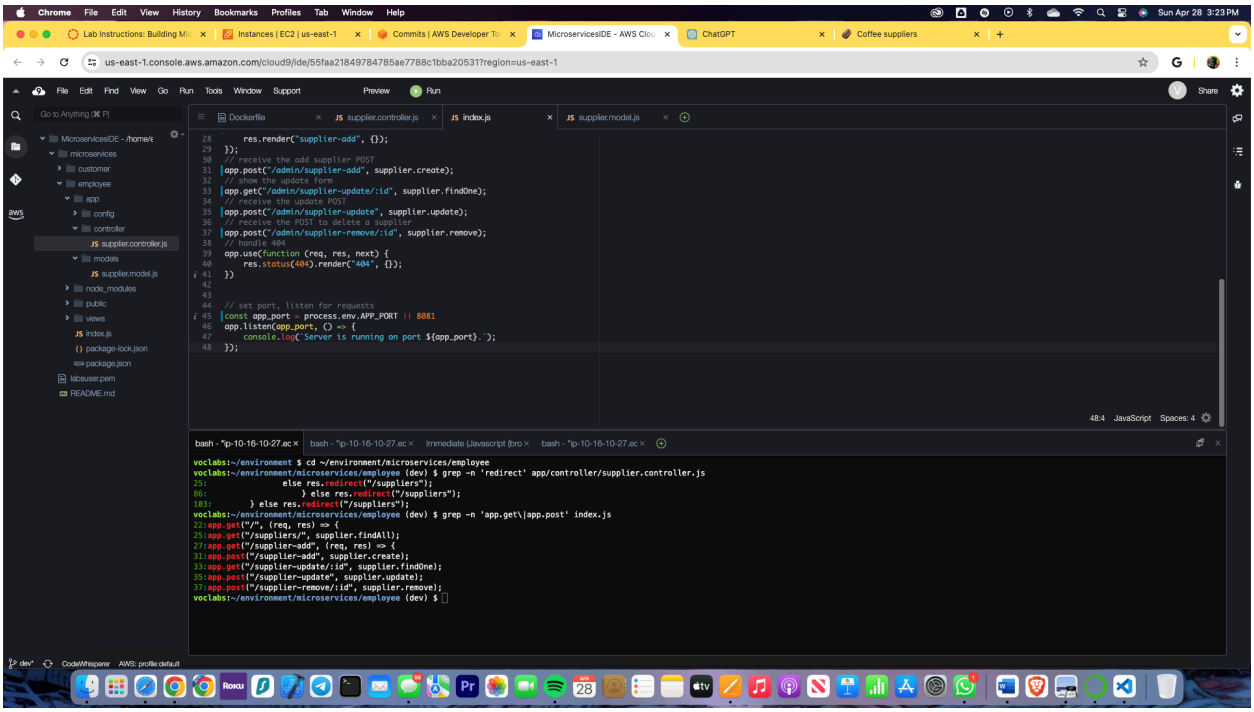
After changes, we get the following output.



The changes are as follows.

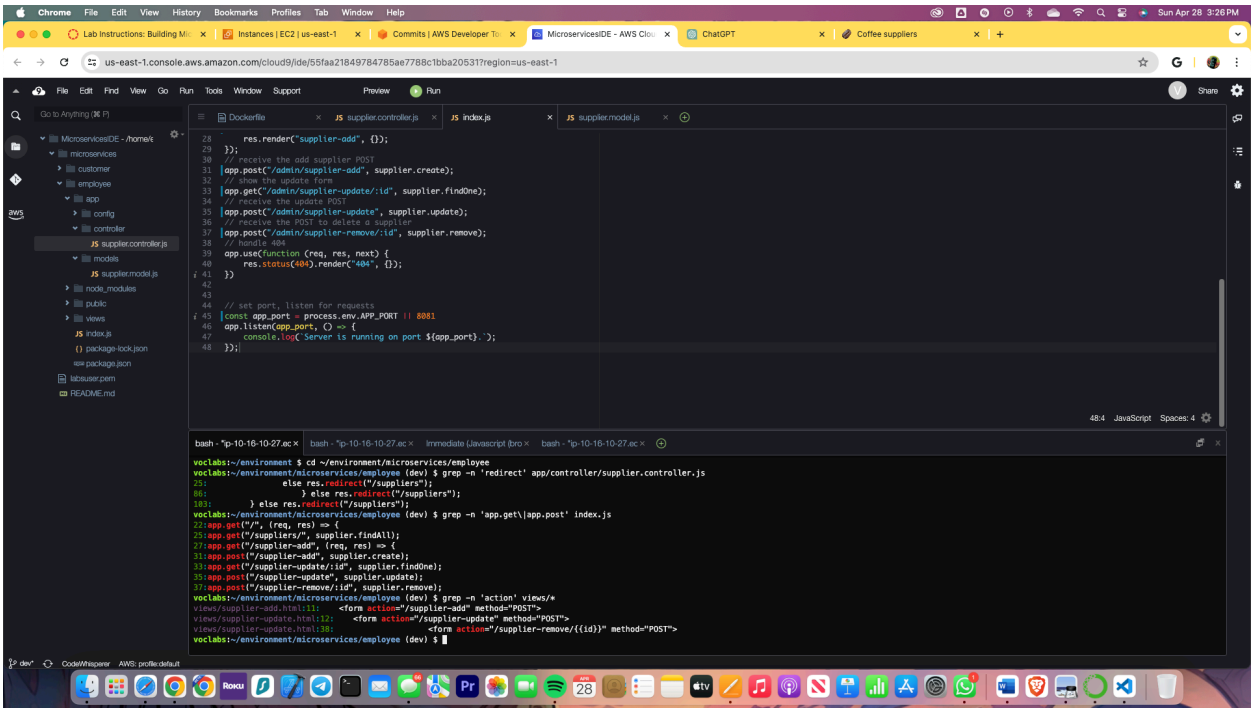
After we edited line 22, the path will become /admin, not /admin/. We should keep that in mind.

On line 45, we changed the port number to 8081



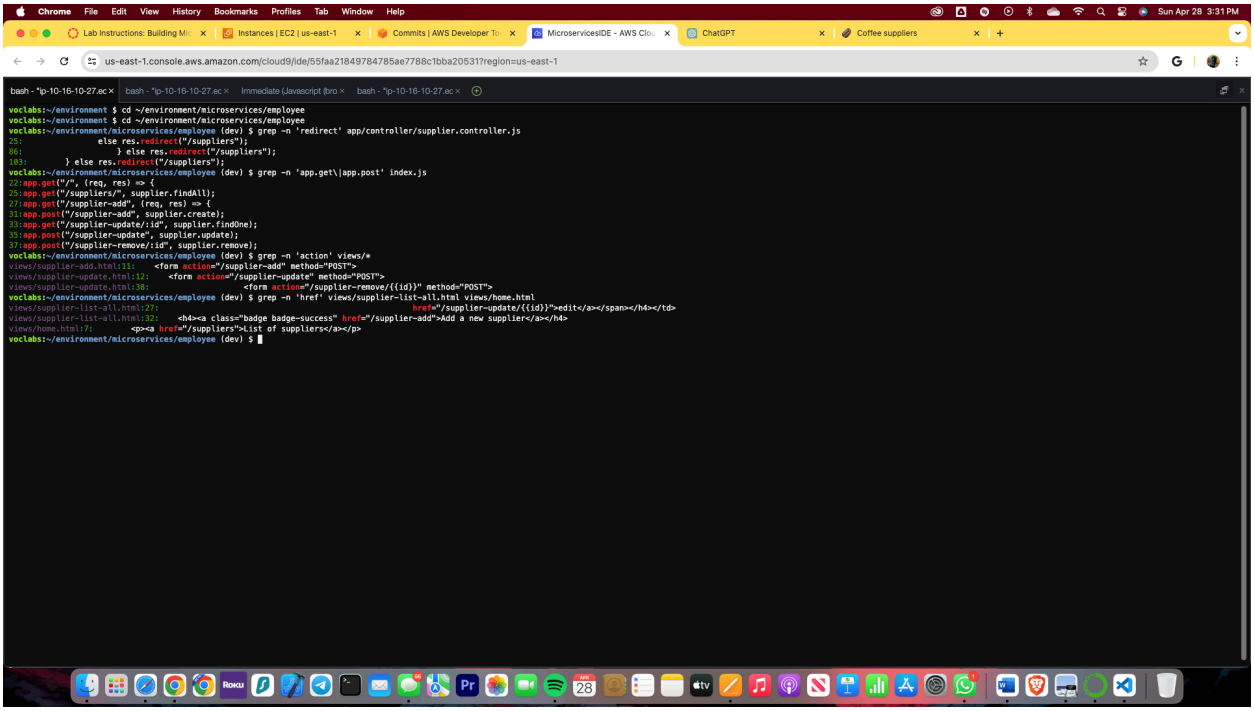
Now in the employee/views/supplier-add.html and employee/views/supplier-update.html files, for the form action paths, we then prepend /admin to the path.

grep -n 'action' views/*

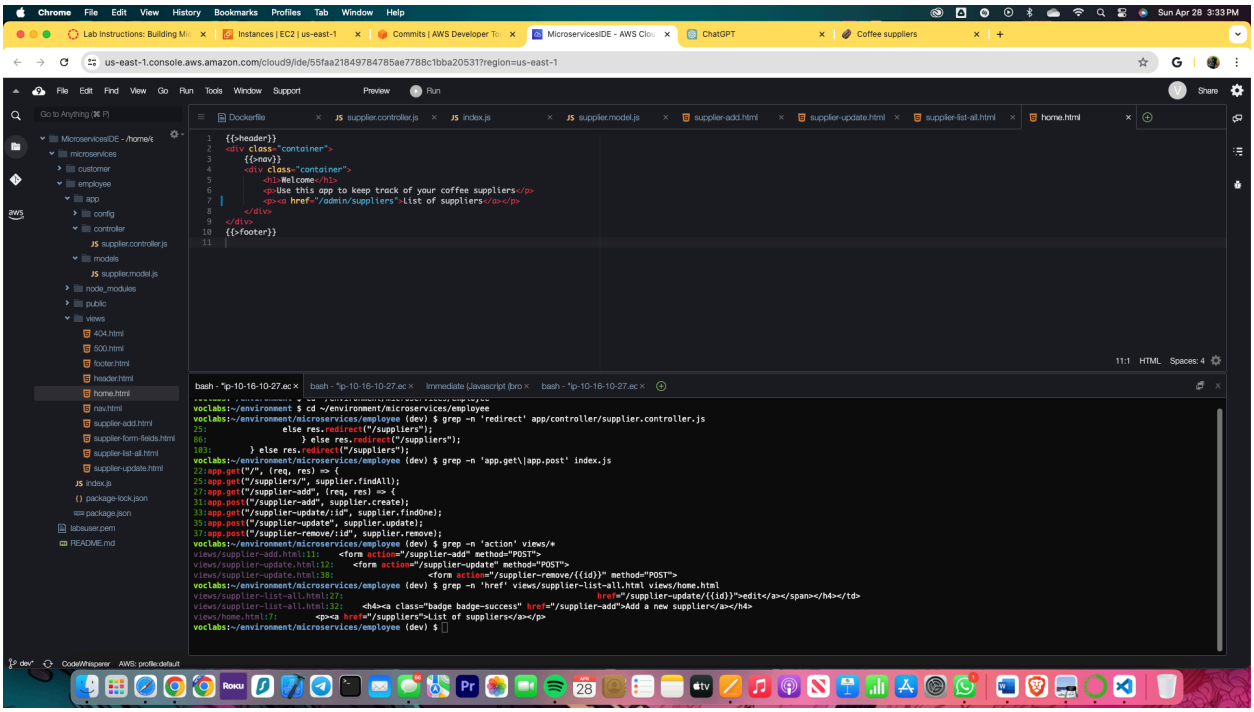


voclabs:~/environment/microservices/employee (dev) $ grep -n 'action' views/*
views/supplier-add.html:11:   <form action="/supplier-add" method="POST">
views/supplier-update.html:12:   <form action="/supplier-update" method="POST">
views/supplier-update.html:38:   <form action="/supplier-remove/{{id}}" method="POST">

voclabs:~/environment/microservices/employee (dev) $ grep -n 'href' views/supplier-list-all.html views/home.html



In the employee/views/supplier-list-all.html and employee/views/home.html files, for the HTML paths, prepend /admin to the path.

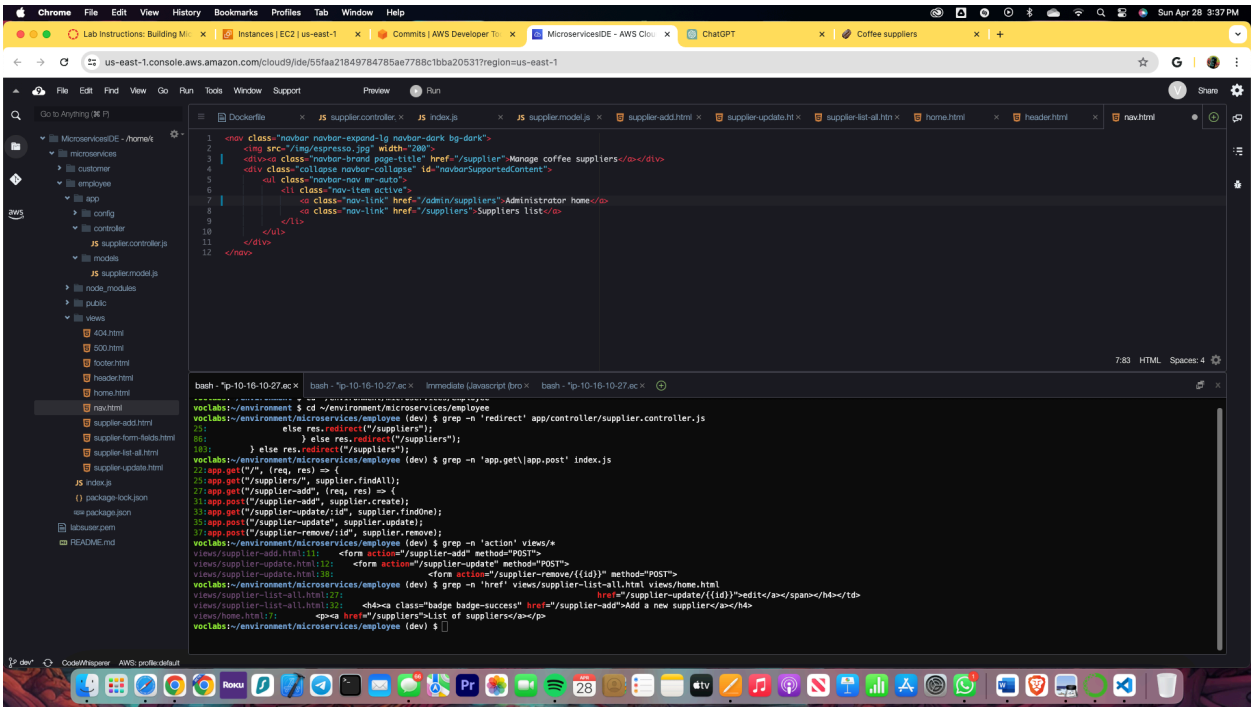


Now we edited the employee/views/nav.html file.

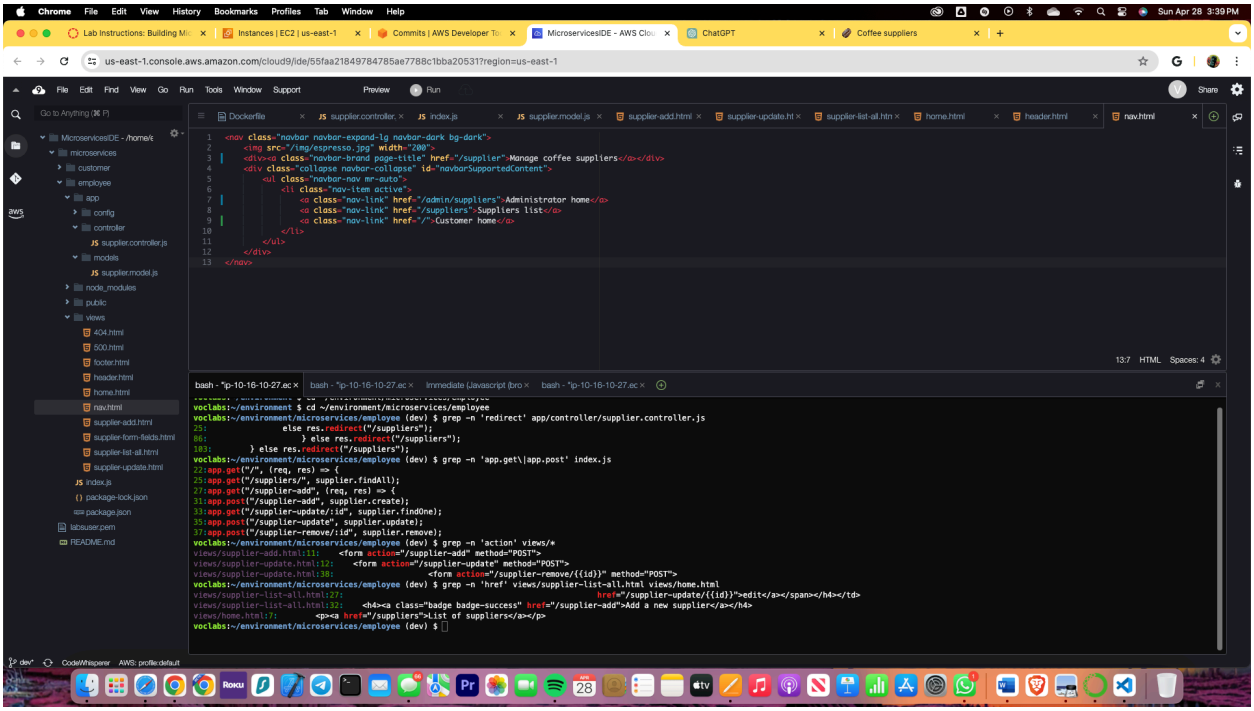
On line 3, we changed Monolithic Coffee suppliers to Manage coffee suppliers

On line 7, we replaced the existing line of code with the following:

Administrator home



Add a new line after line 8 that contains the following HTML:
Customer home

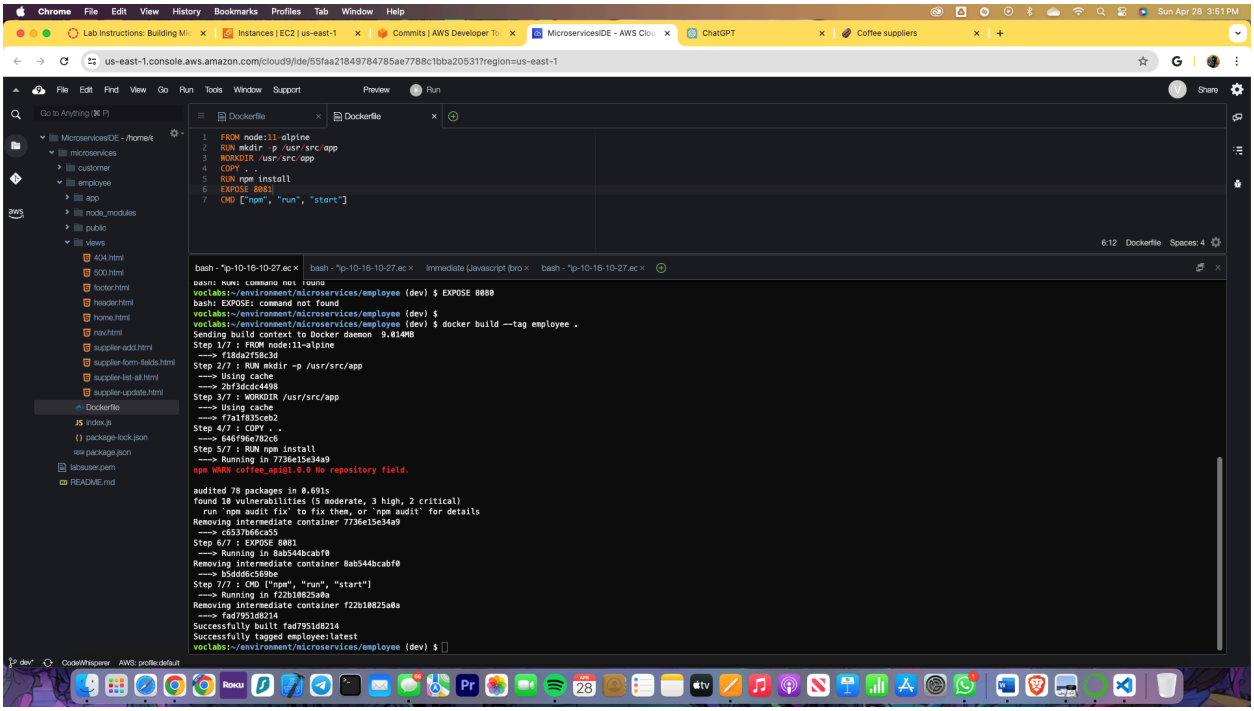


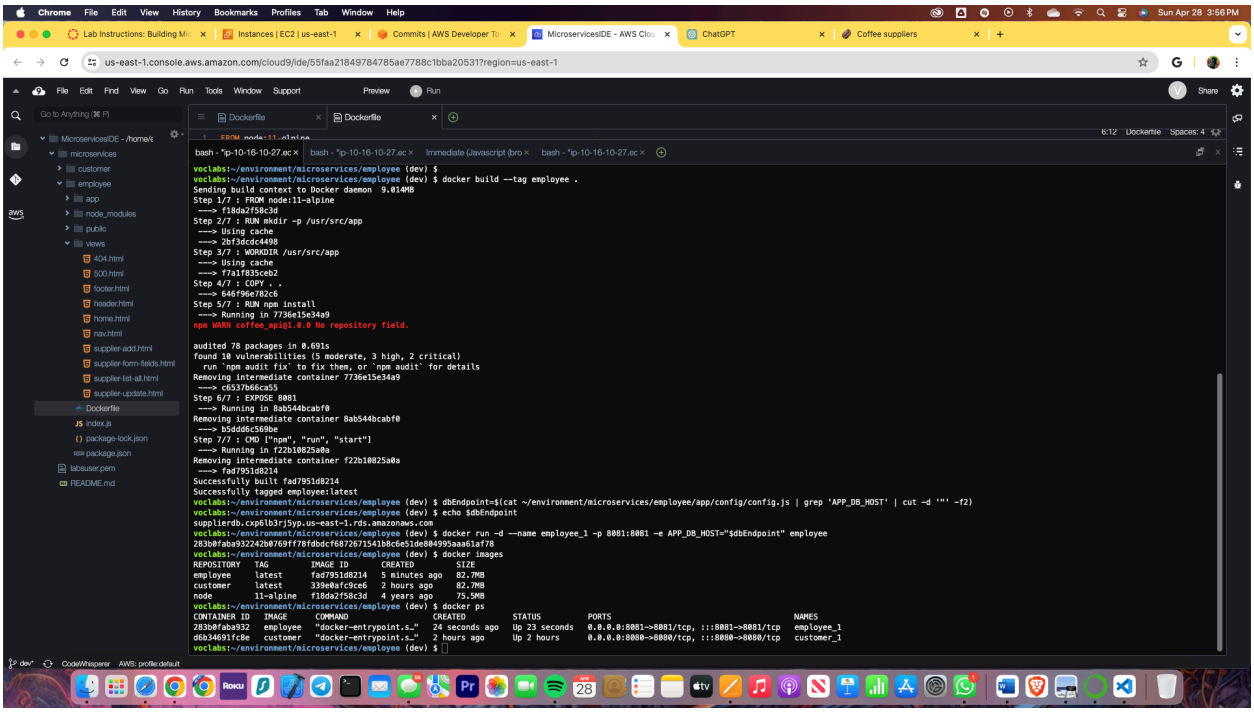
Task 4.5: Create the employee microservice Dockerfile and launch a test container

Here we created a Dockerfile for the employee microservice.

Now we duplicated the Dockerfile from the customer microservice into the employee microservice area. Then we edited the employee/Dockerfile to change the port number on the EXPOSE line to 8081

And finally we built the Docker image for the employee microservice. Specify employee as the tag.



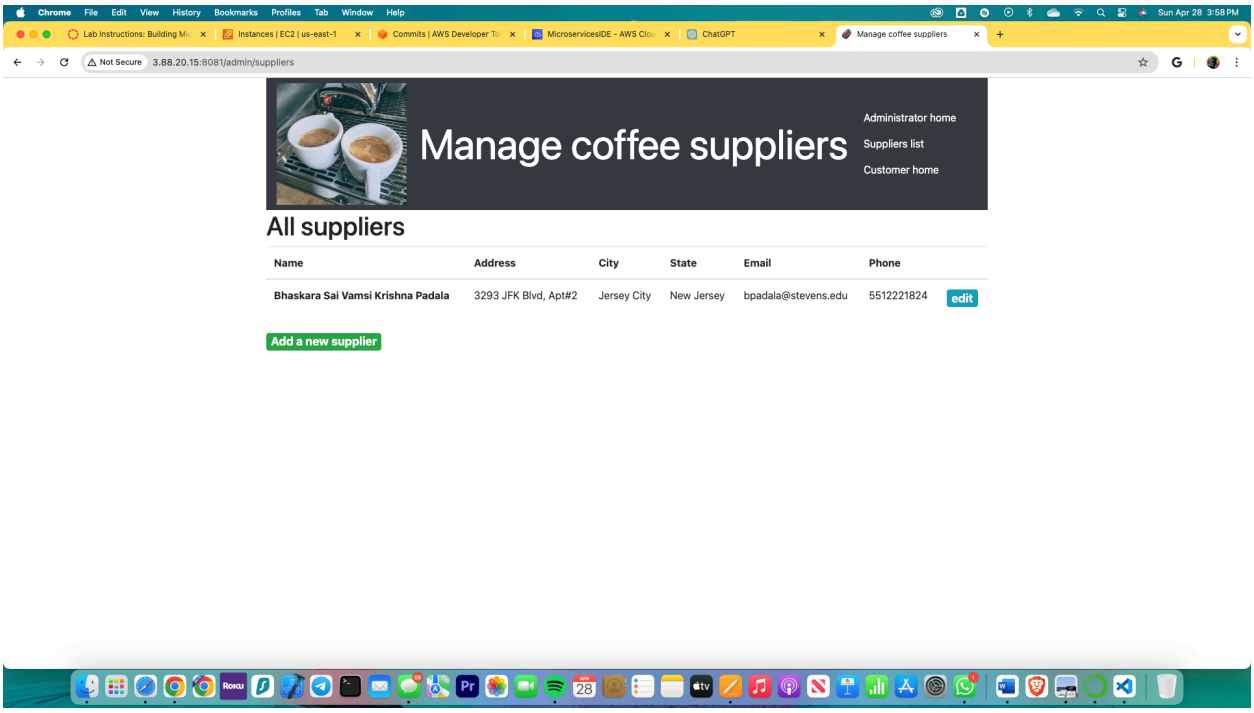


For the above screen,
We Build the Docker image for the employee microservice. Specify employee as the tag. And then we run a container named employee_1 based on the employee image. Run it on port 8081 and be sure to pass in the database endpoint.

Here in the below image, we can verify that the employee microservice is running in the container and that the microservice functions as intended.

Now we will load the microservice web page in a new browser tab at http://http://3.88.20.15:8081/admin/suppliers

And the application is being loaded

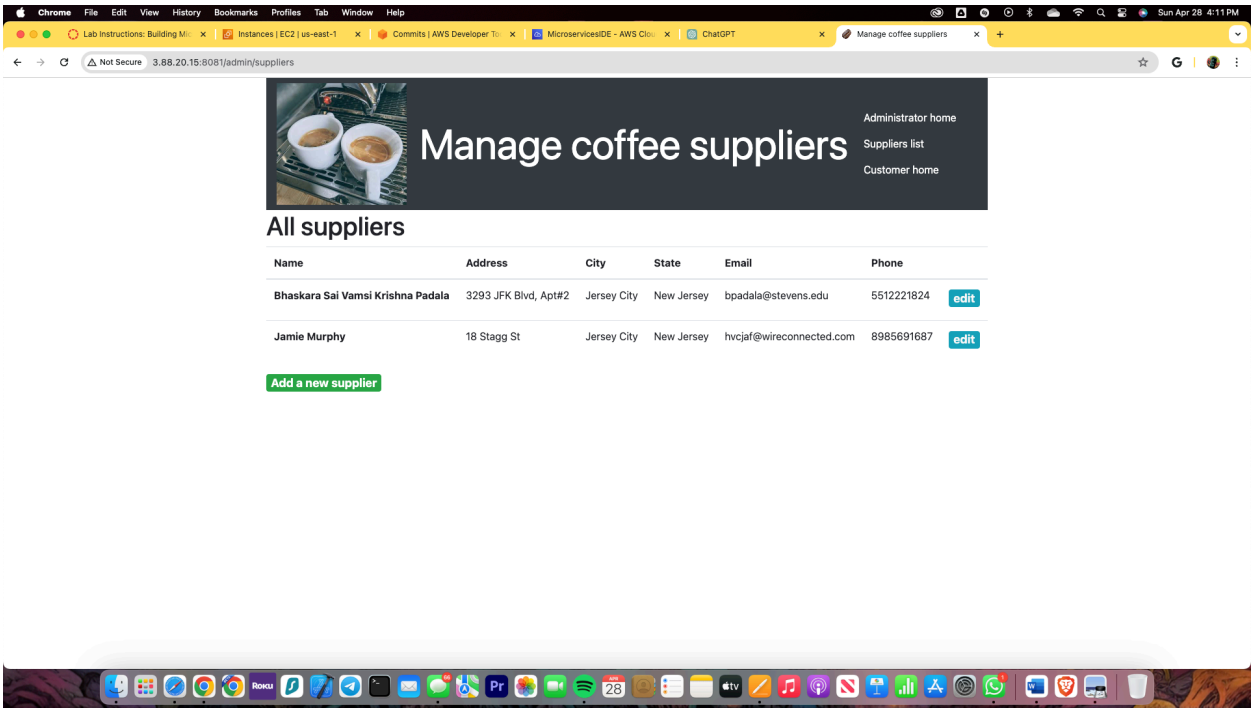


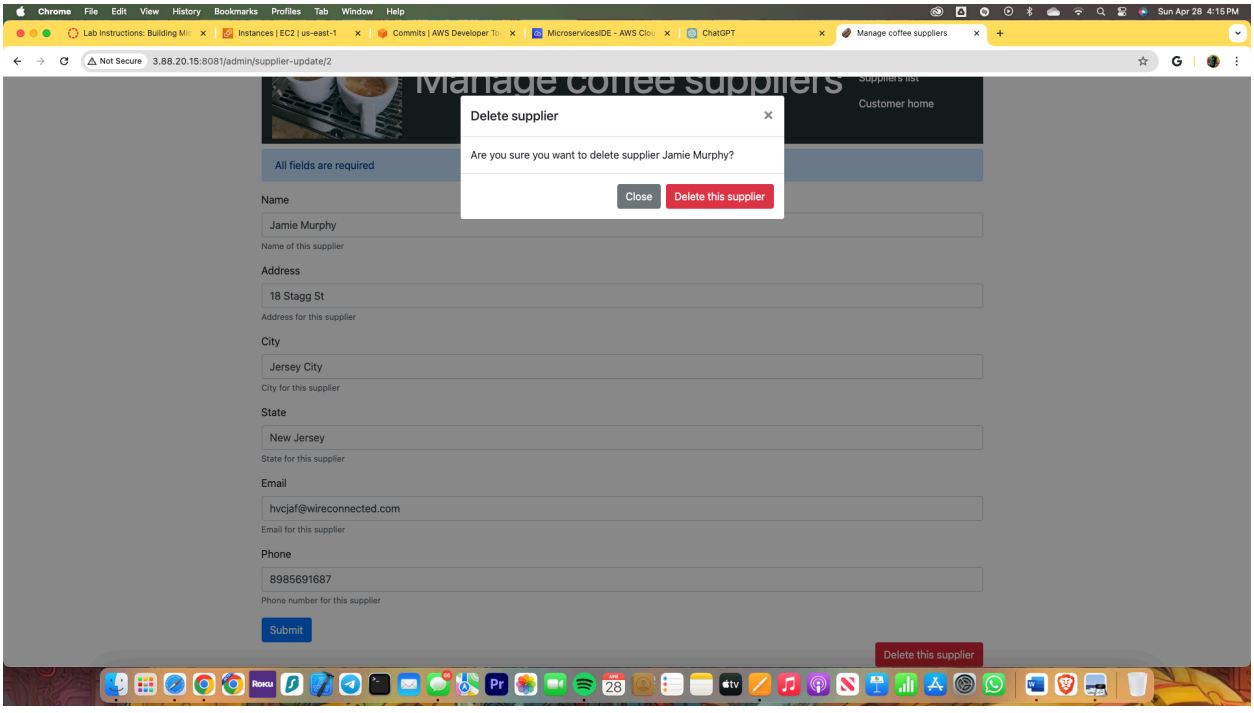
Now we will Test adding a new supplier.

Then we will verify that the new supplier appears on the suppliers page.

Now we gave a new supplier to the list and we will now see the following output in the image below that the new supplier has been added successfully.

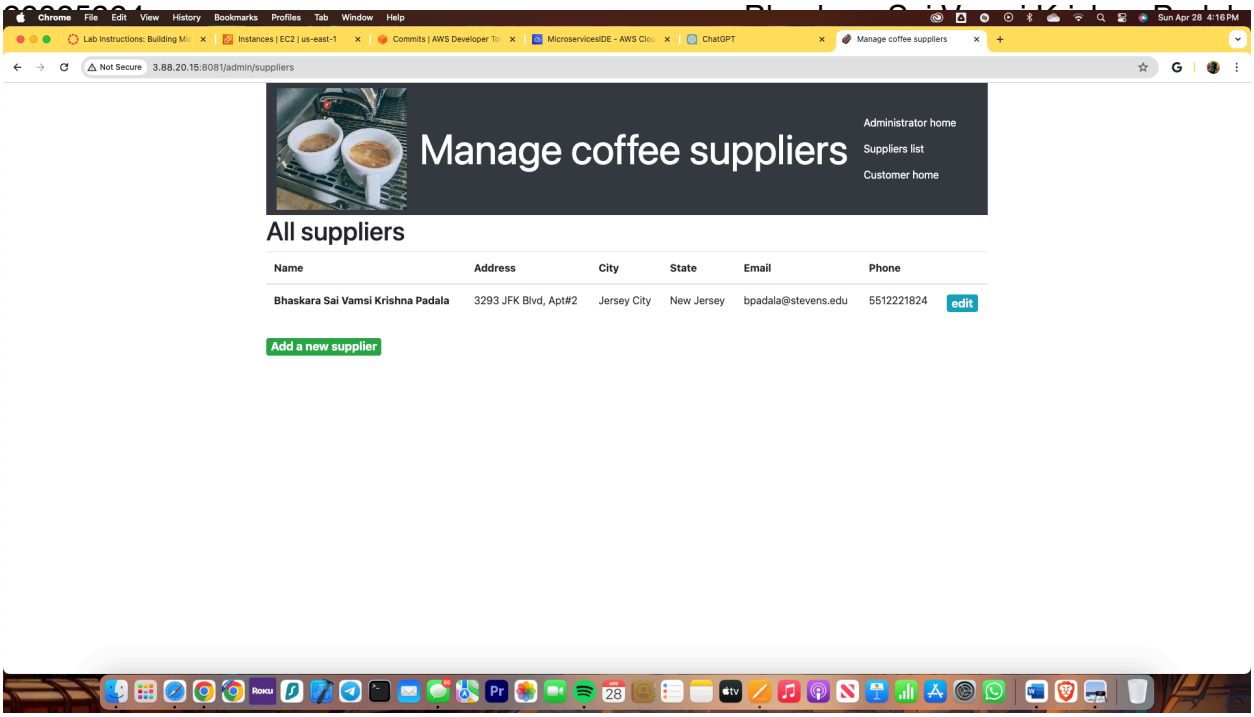
Also in the next image we showed that we can delete the supplier and we will delete as follows.



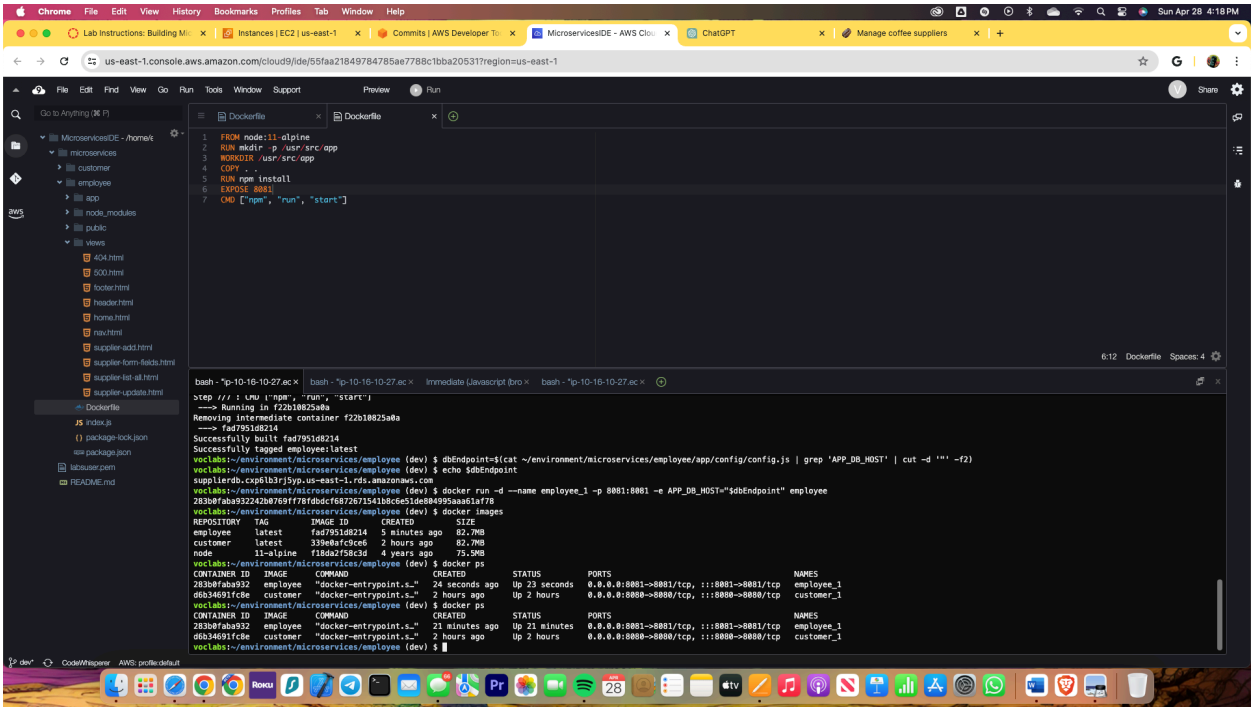


Hence, we deleted the supplier successfully as shown in the figure below.

To do that we followed these steps.
On the row for a particular supplier entry, choose edit. At the bottom of the page, choose Delete this supplier, and then choose Delete this supplier again. Verify that the supplier no longer appears on the suppliers page.

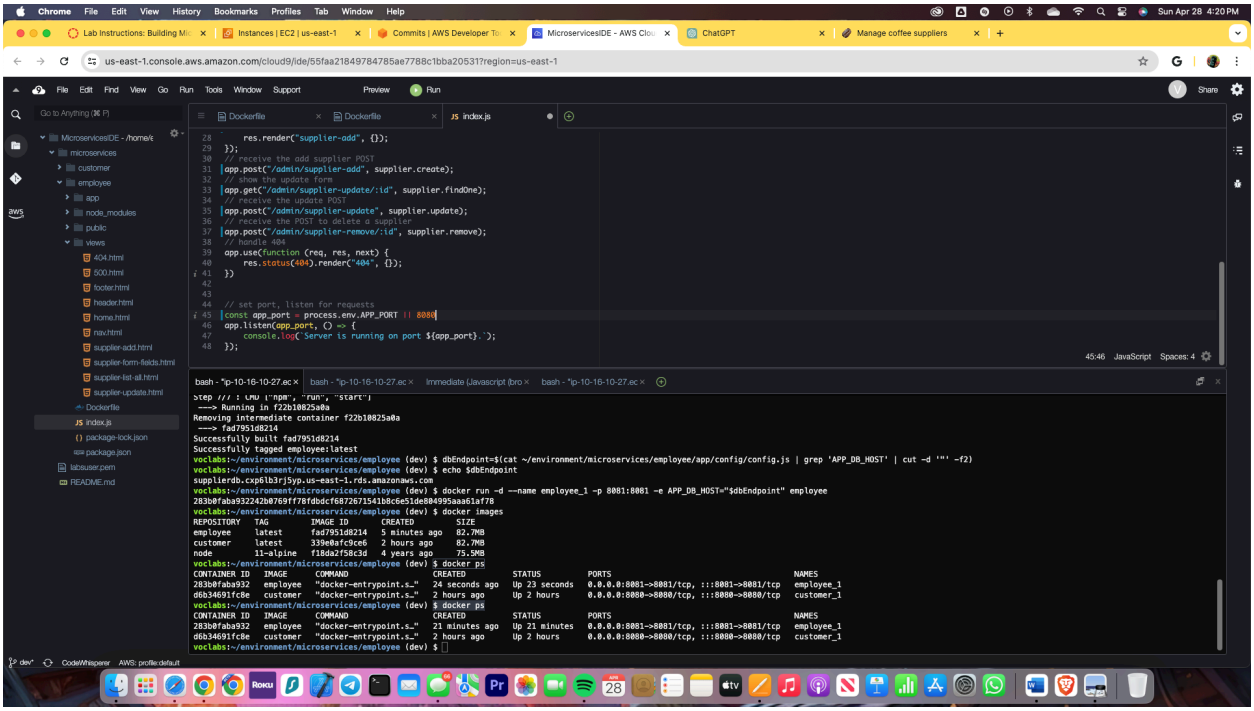


To observe the both running dockers, we will check them by using the following command.
$ docker ps

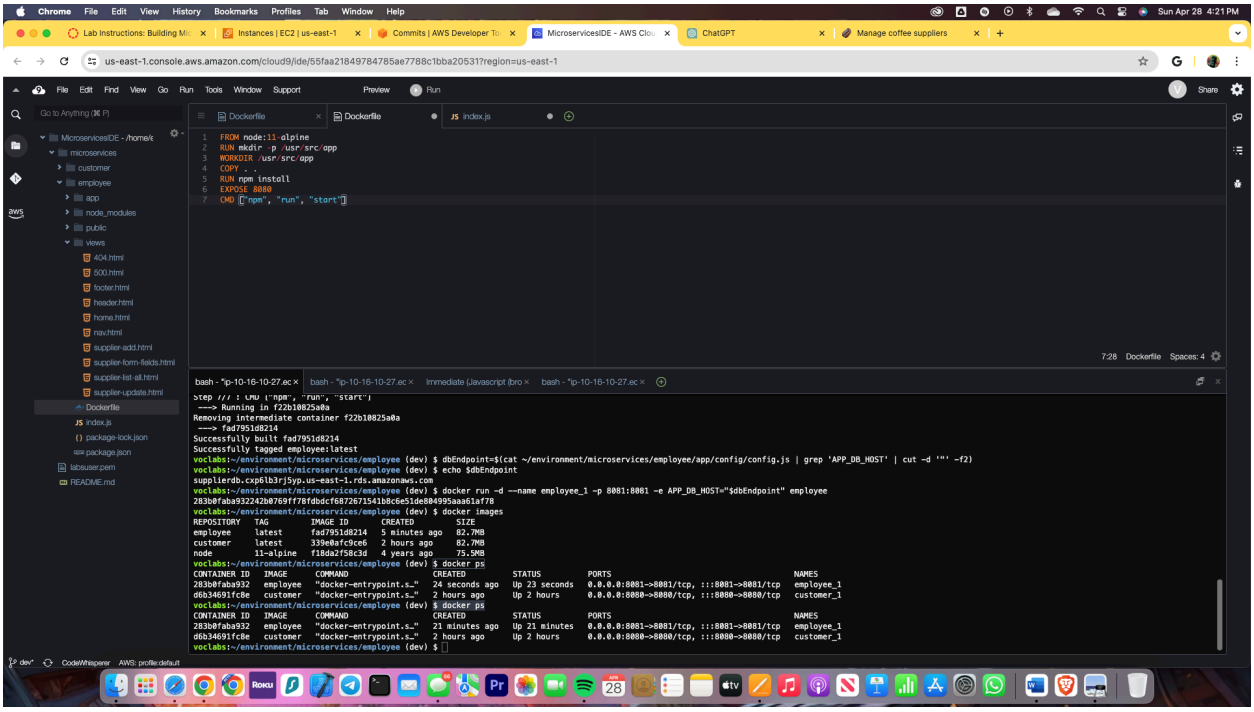


Task 4.6: Adjust the employee microservice port and rebuild the image

Here we will edit the employee/index.js and employee/Dockerfile files to change the port from 8081 to 8080



Now we will change the port number on docker file.

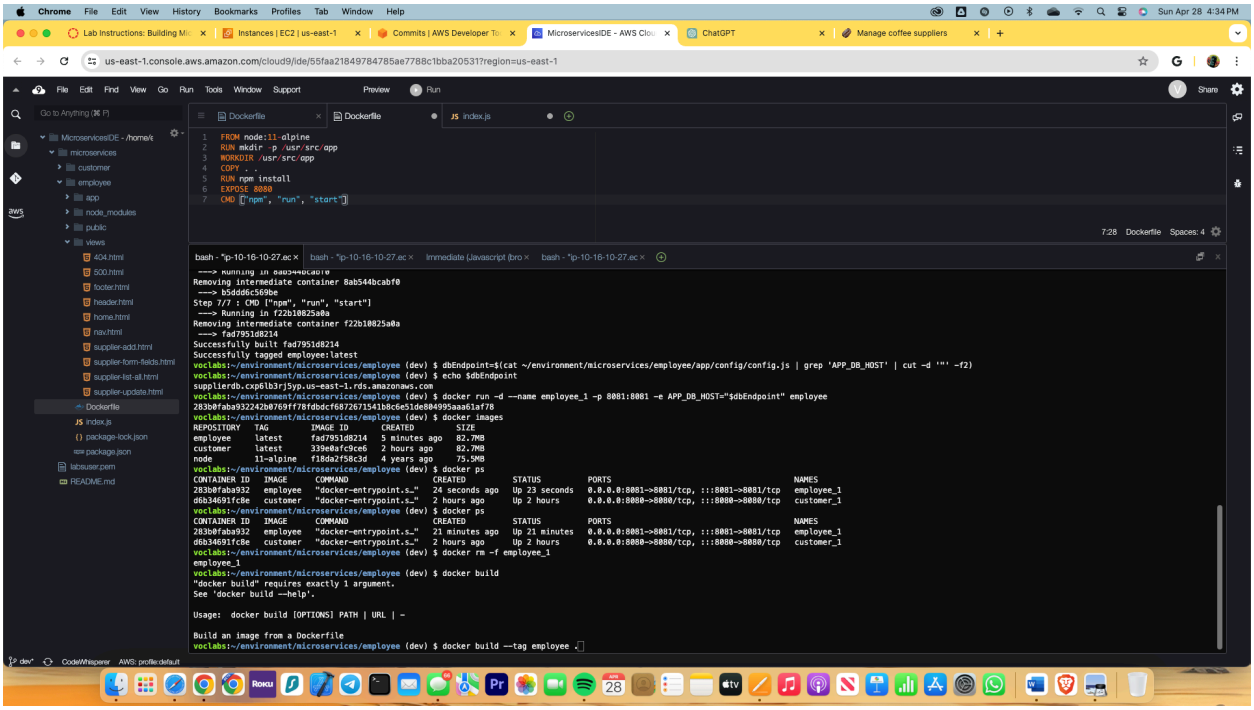


Now we will Rebuild the Docker image for the employee microservice.

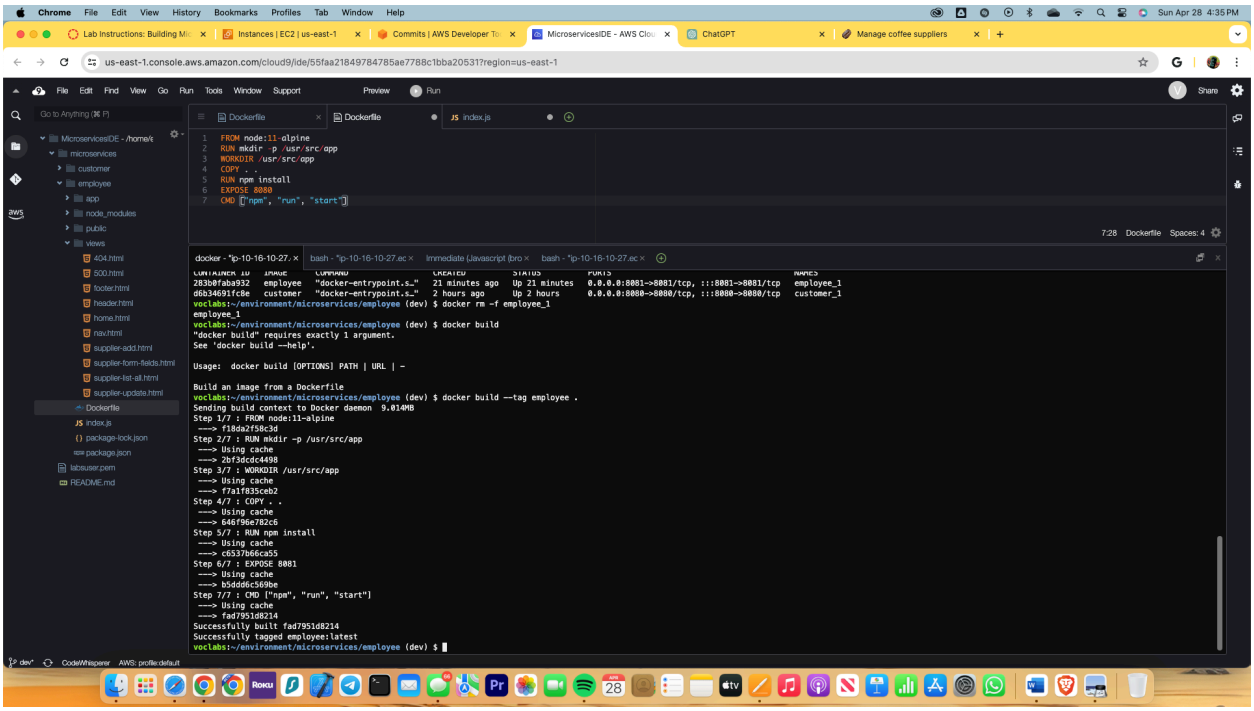
To stop and delete the existing container (assumes that the container name is employee_), run the following command:

docker rm -f employee_1

Now we will build docker

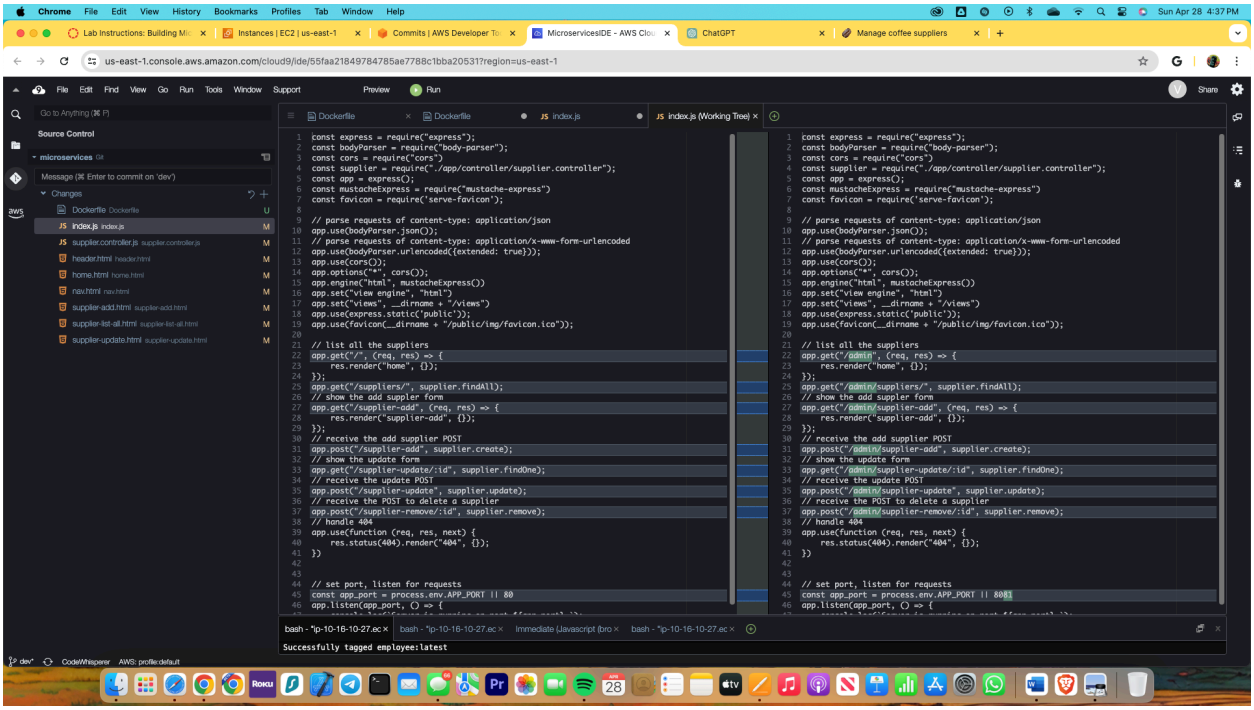


We've successfully built the docker and it is as follows.

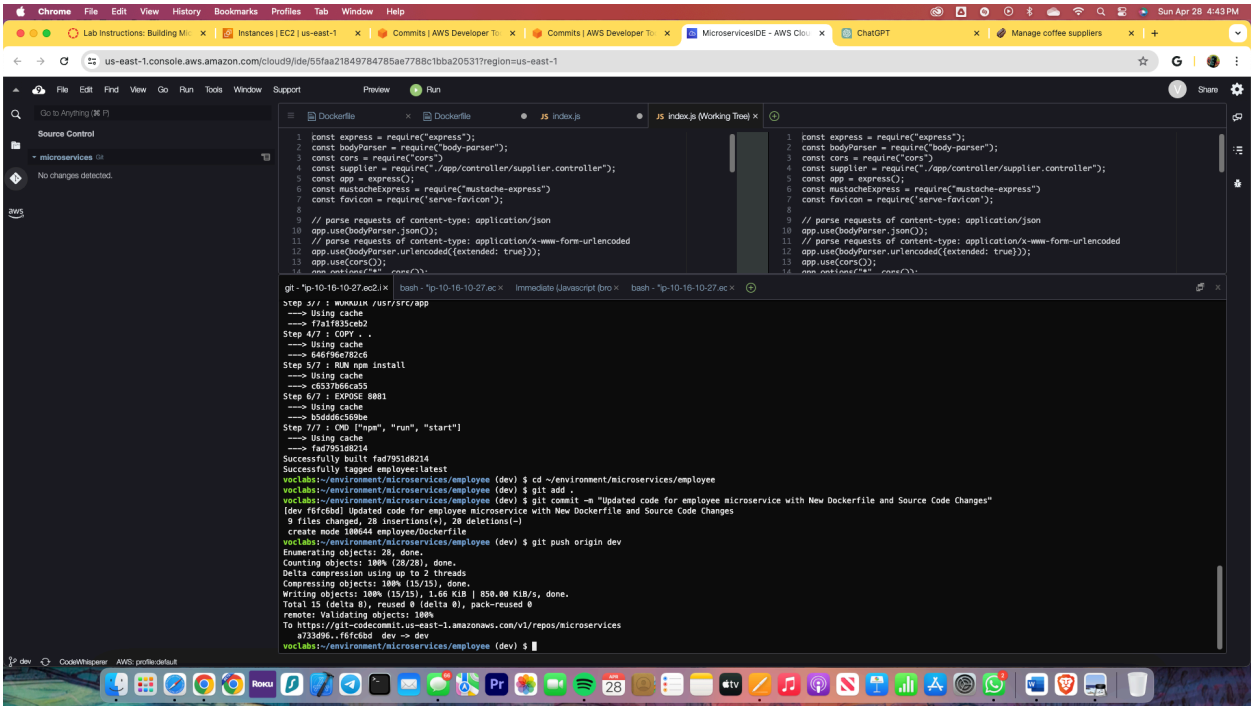


Task 4.7: Check code into CodeCommit
In this task, we will commit and push the changes that we made to the employee microservice to CodeCommit.

Here we can compare the changes that are happened in the Index.js



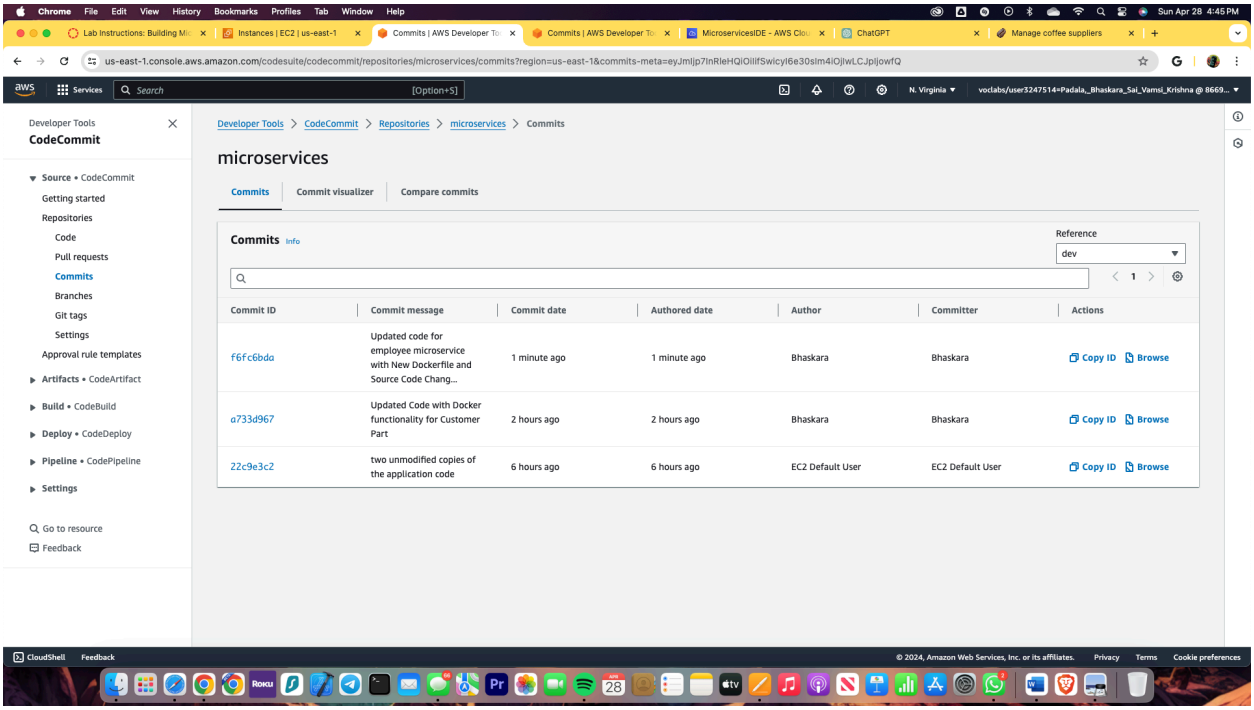
Now we will push all the code changes that we performed as shown.

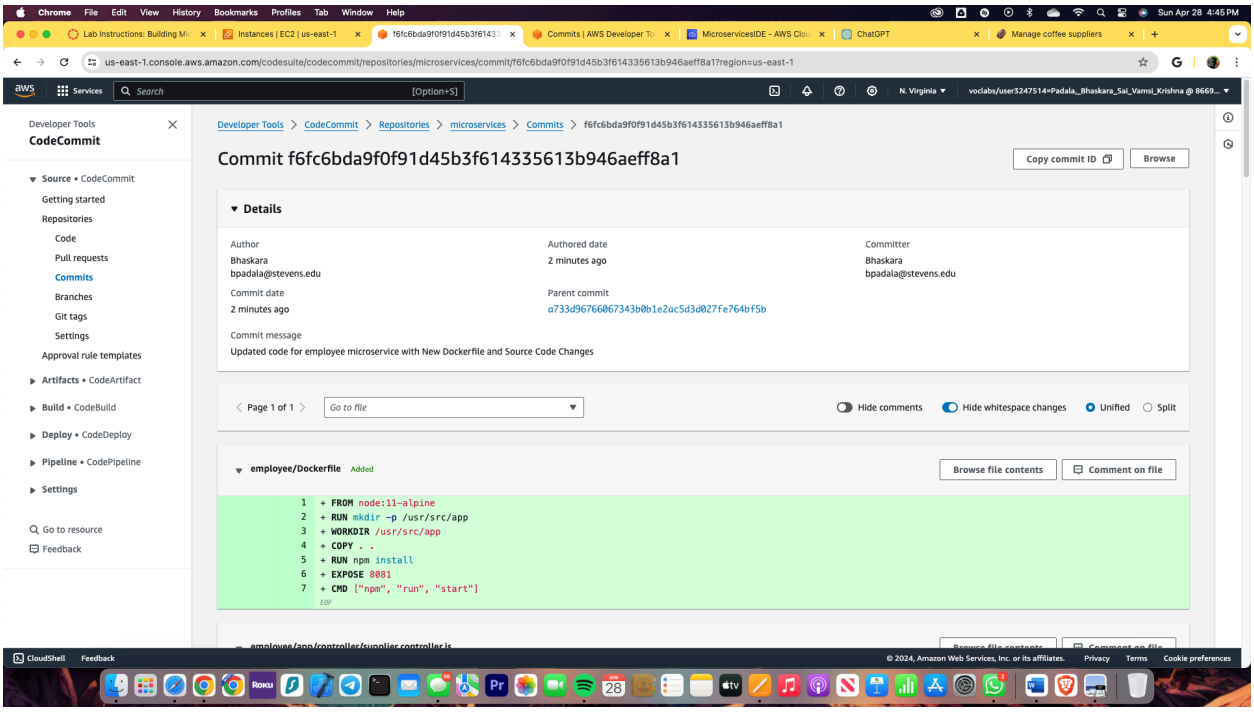


We used the below commands to perform the action :
cd ~/environment/microservices/employee
git add .

git commit -m "Updated code for employee microservice"
git push origin dev

So, after updating the new commits, we can observe changes in the CodeCommits as follows.



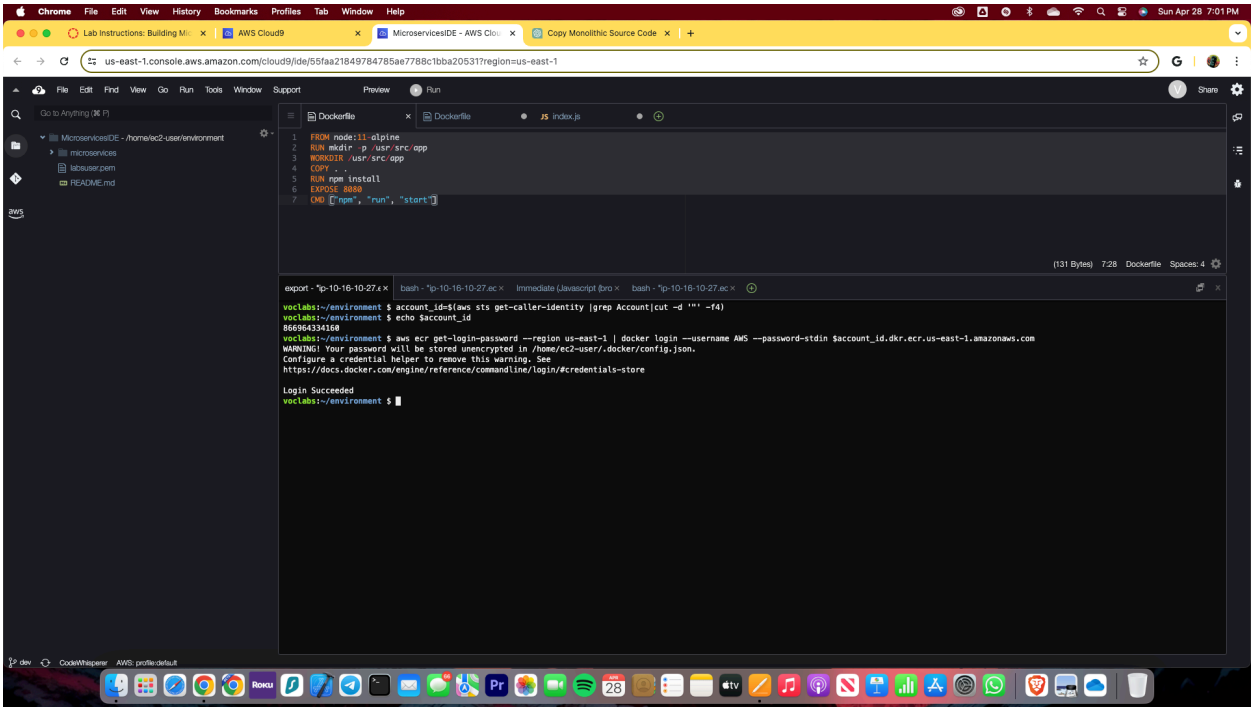


Phase 5: Creating ECR repositories, an ECS cluster, task definitions, and AppSpec files

Task 5.1: To Create ECR repositories and upload the Docker images

In this phase, we will upload the latest Docker images of the two microservices to separate Amazon ECR repositories. To authorize our Docker client to connect to the Amazon ECR service, run the following commands:

```
account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
echo $account_id
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
```

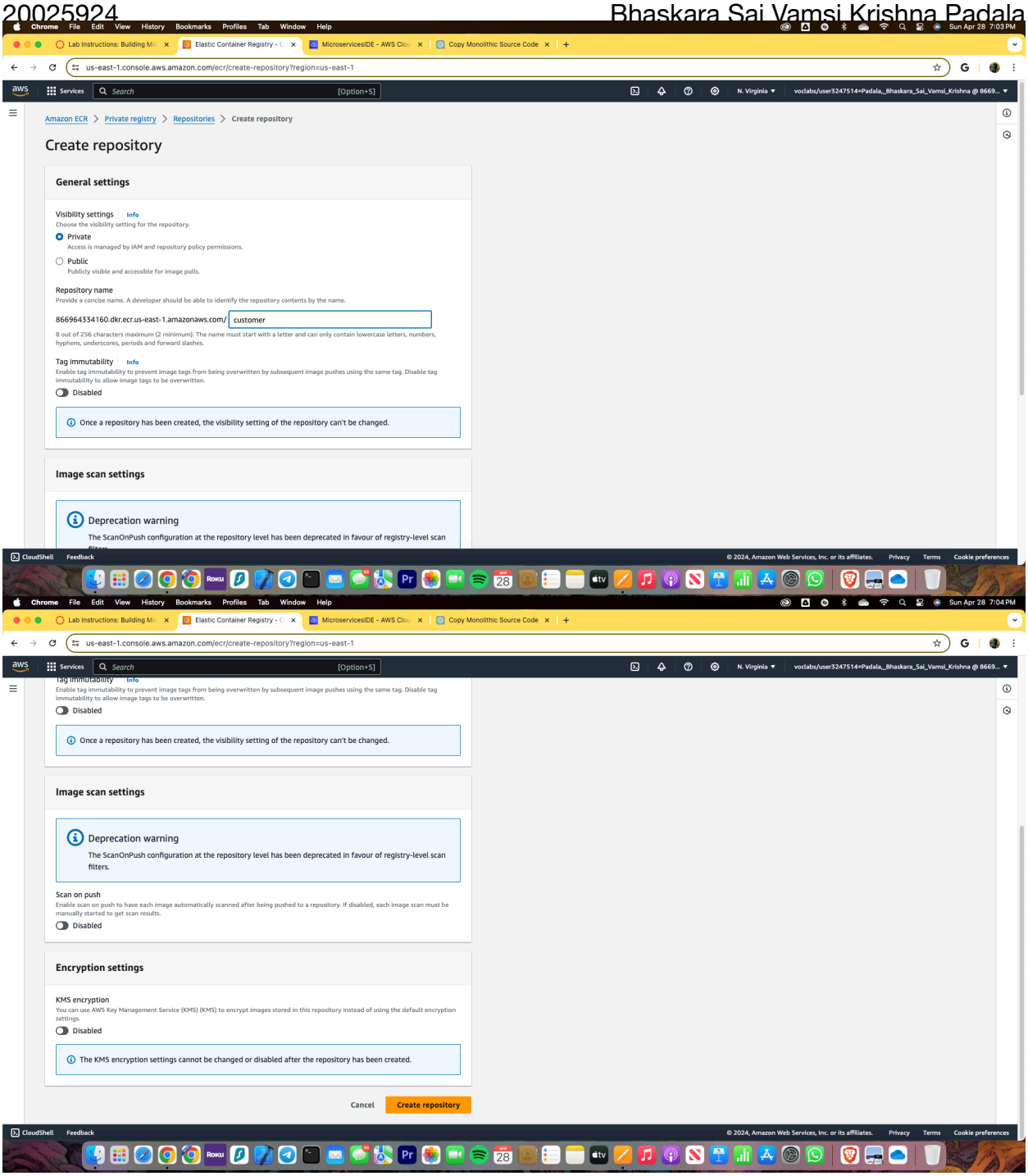


Here, we successfully completed the Login as shown.

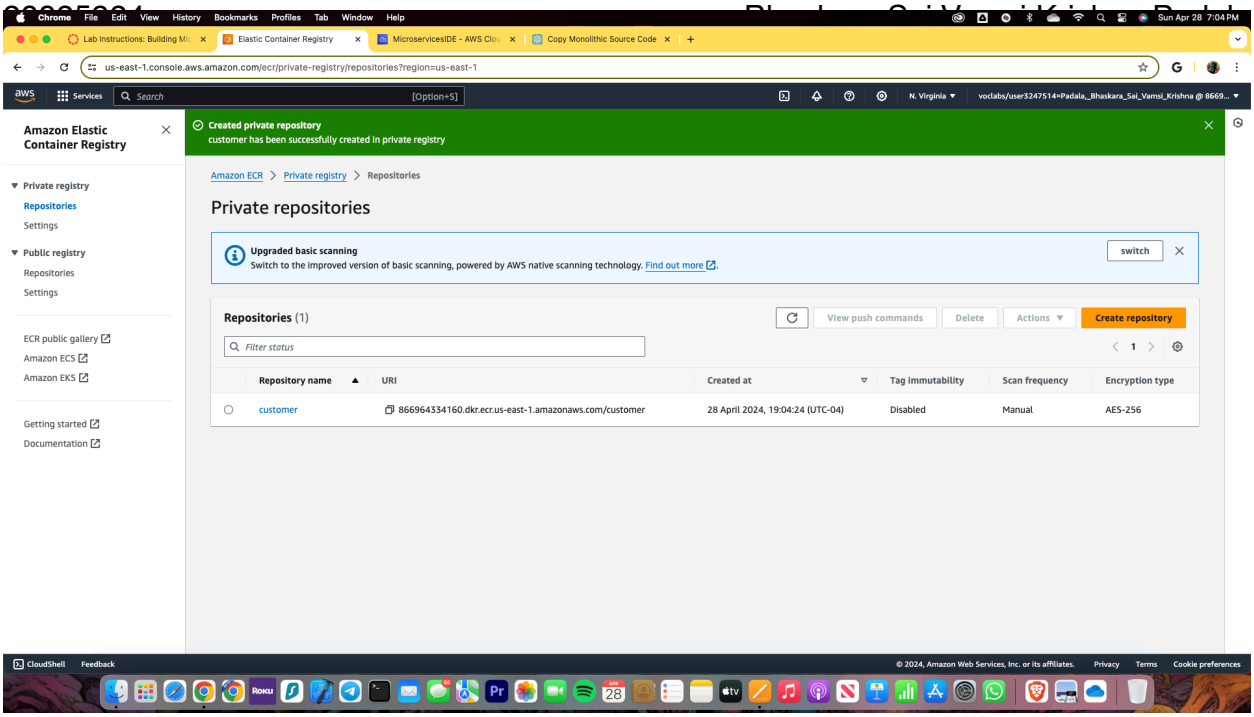
Now we will create a separate private ECR repository for each microservice.

Firstly, we will name the first repository customer

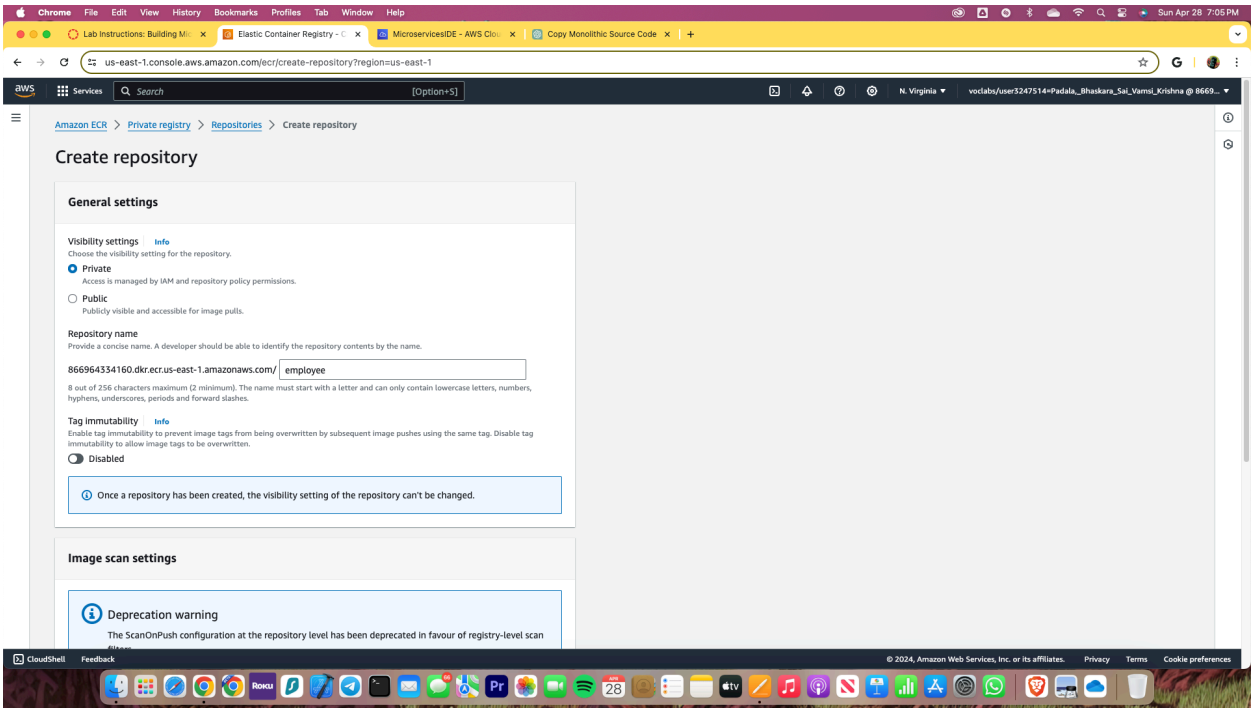
The we will name the second repository employee



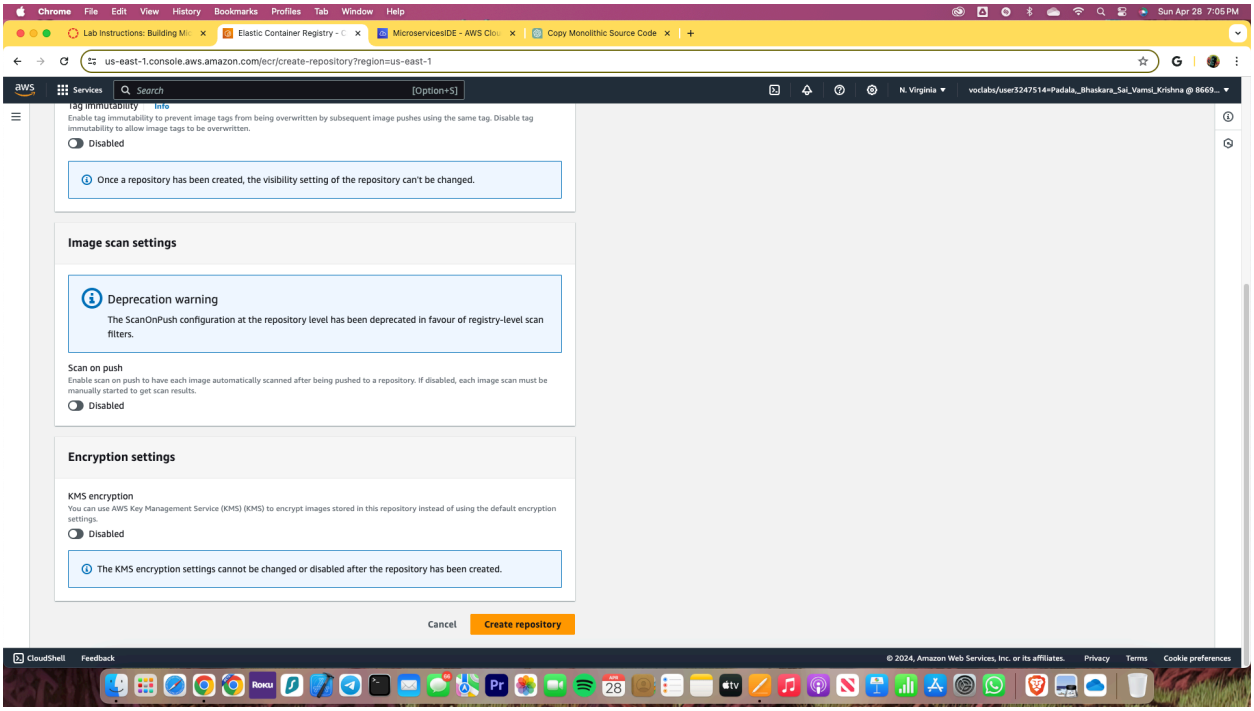
Hence, we successfully created the repository named customer.



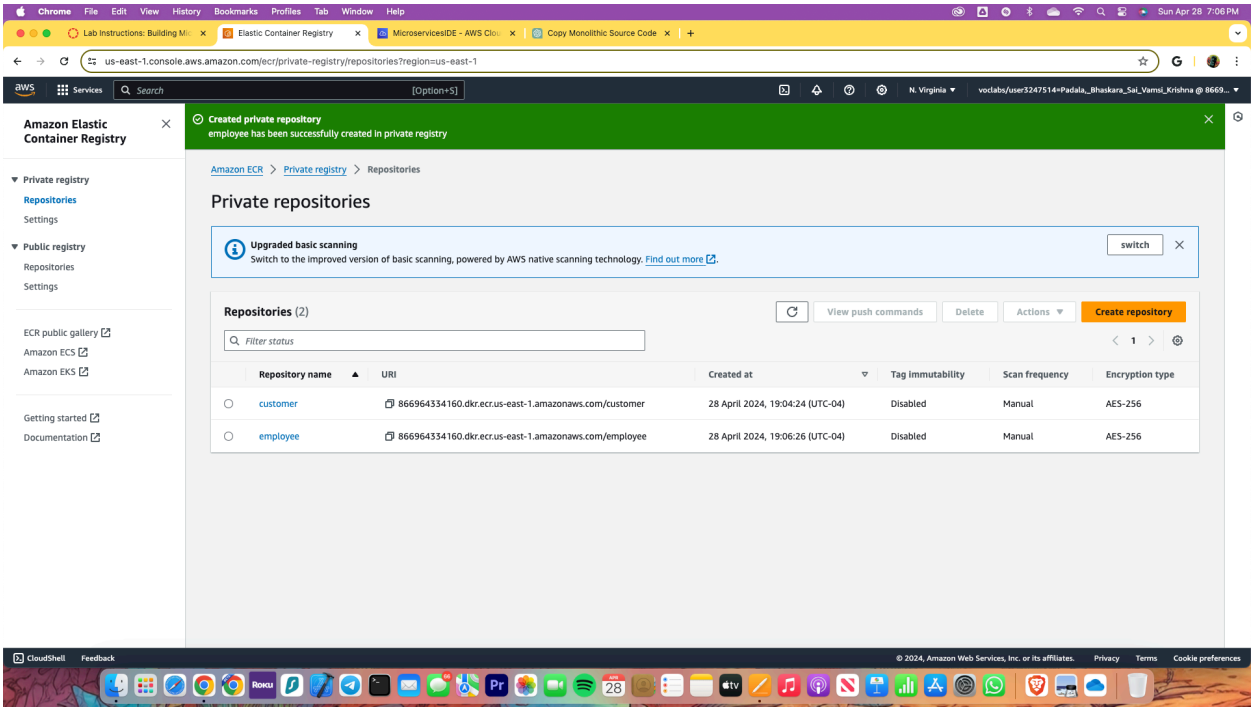
Now we will create the repository named employee.



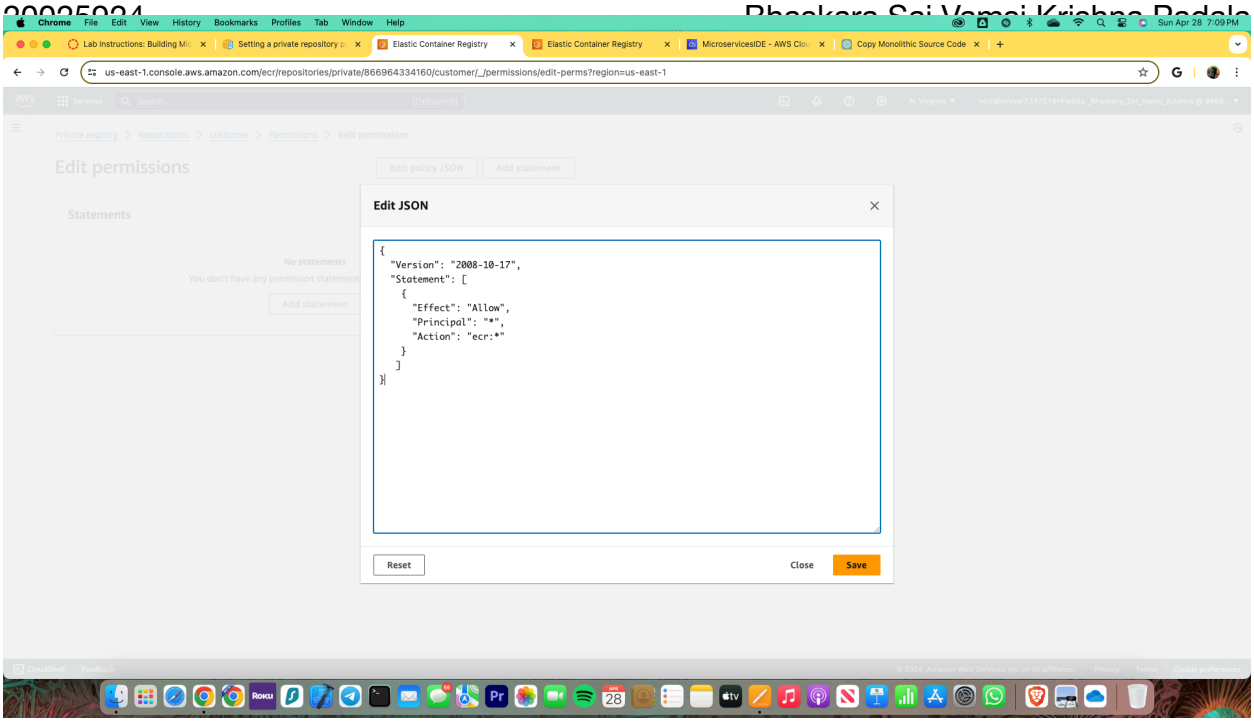
Now, we need to click on the 'Create Repository' Button so that repository will be created.

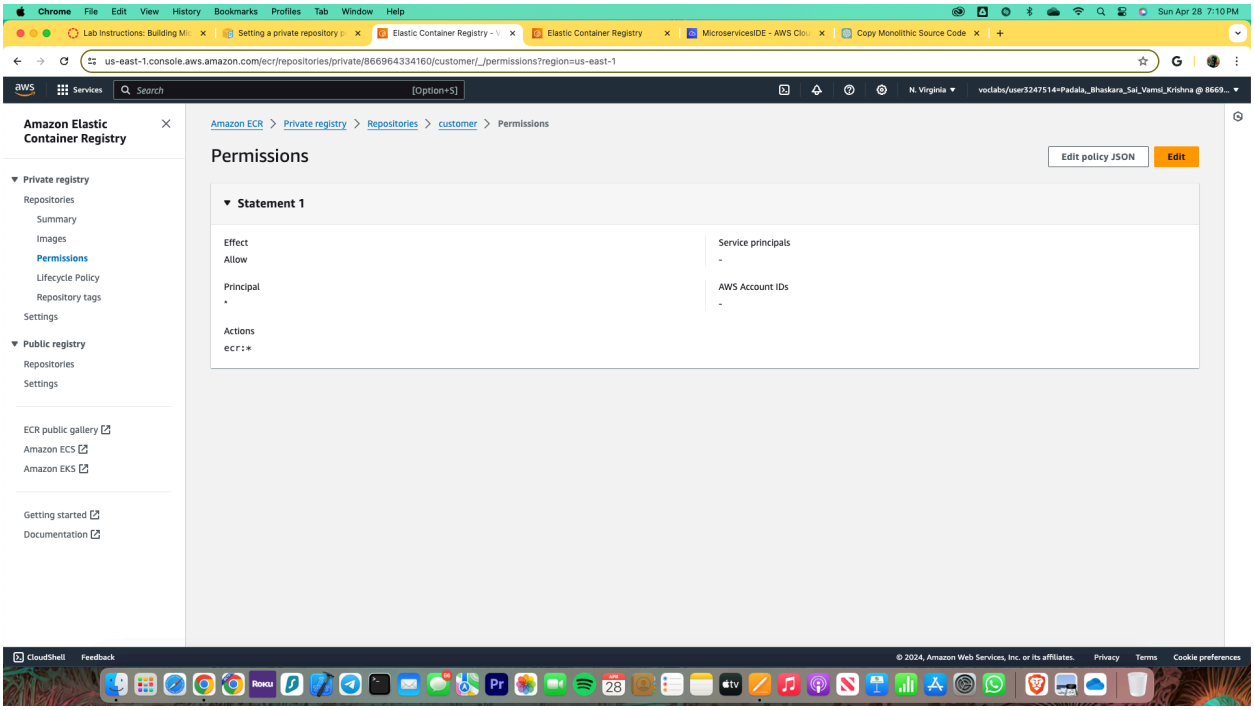


Hence, we successfully created the repository.



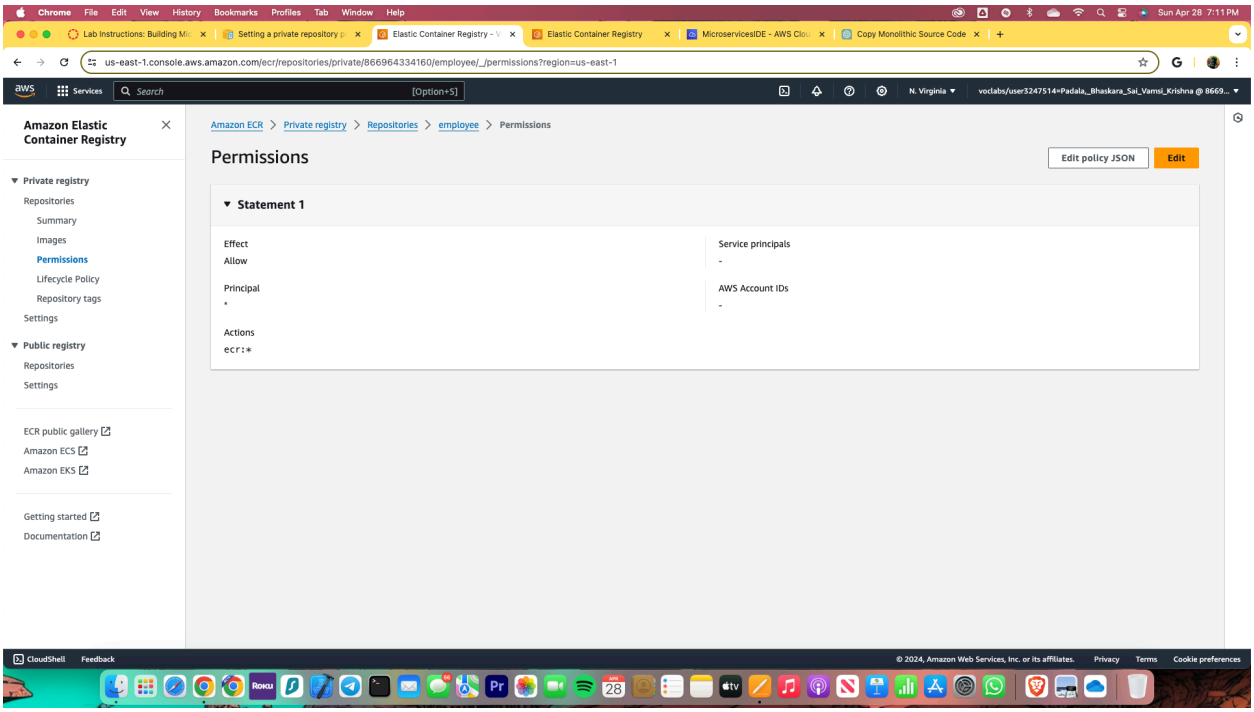
Now we need to Set permissions on the customer ECR repository.
To perform the task we need to proceed as follows.





Now that we edited the permissions for customer repository, now we will proceed to the editing of the employee part.

Now we use the same approach to set the same permissions on the employee ECR repository.



Now we will proceed to tagging the Docker images with your unique registryId (account ID) value to make it easier to manage and keep track of these images.

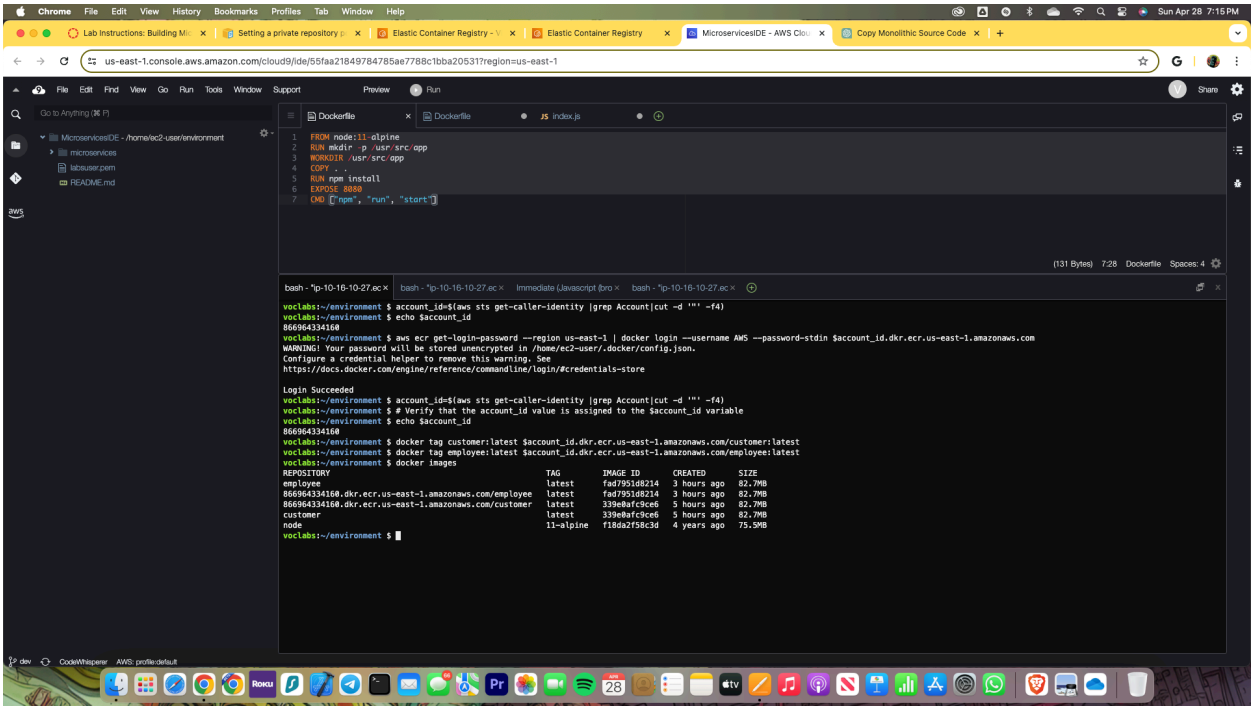
In the AWS Cloud9 IDE, we will run the following commands:

account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)

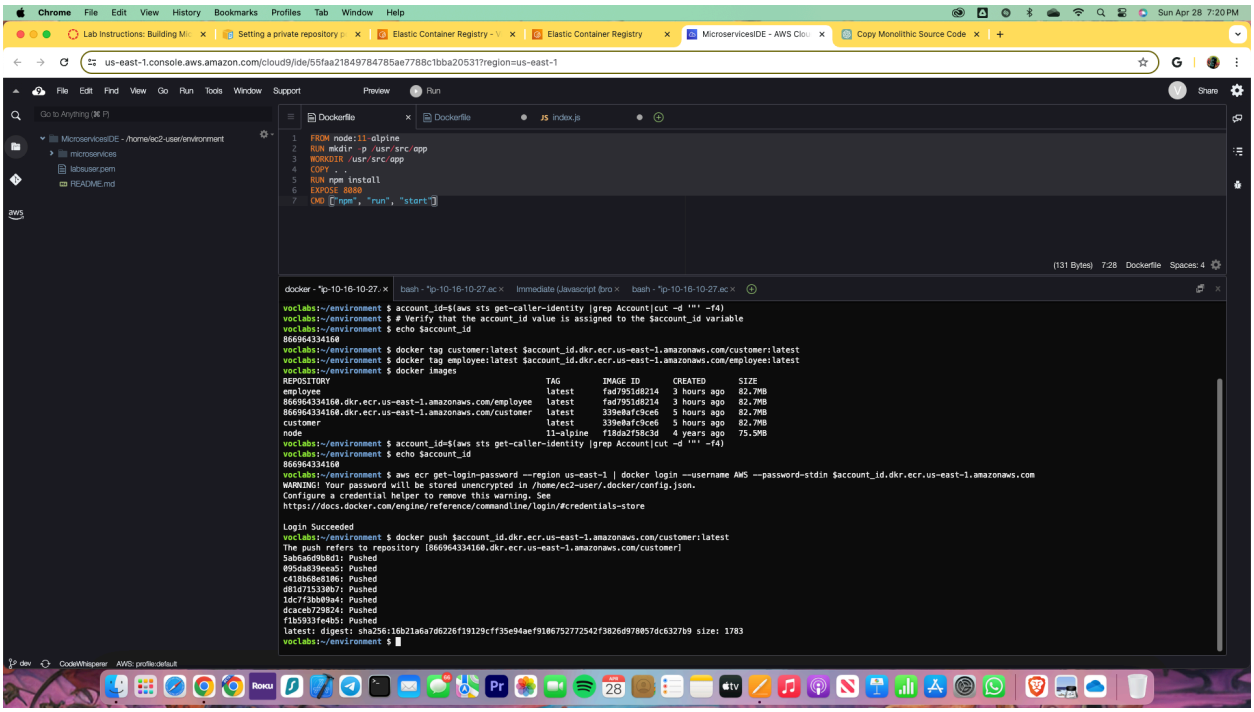
Verify that the account_id value is assigned to the $account_id variable
echo $account_id

Tag the customer image
docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest

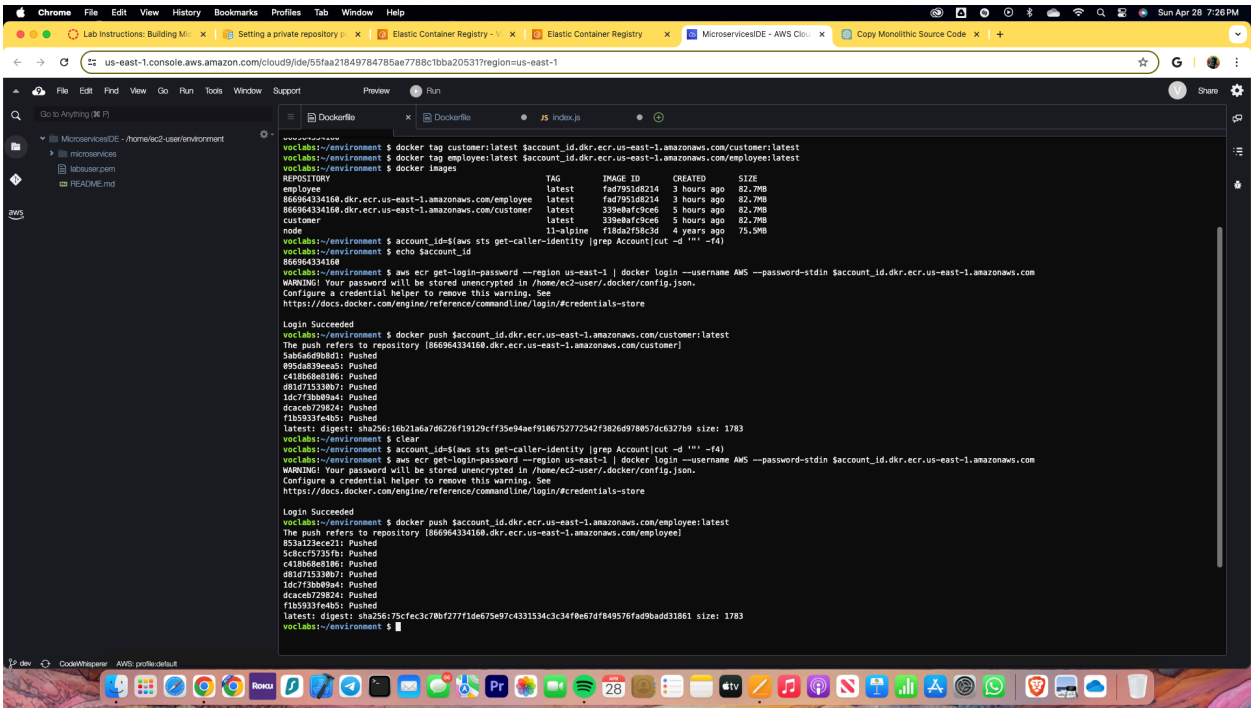
Tag the employee image
docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest

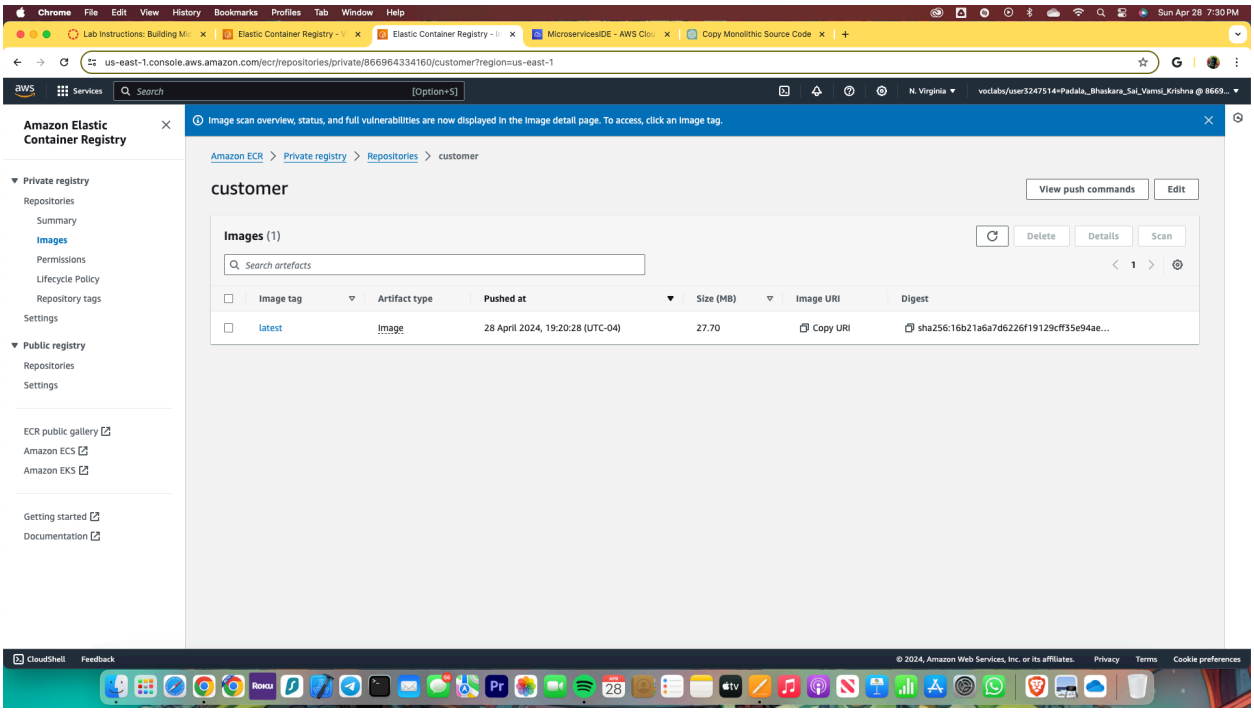


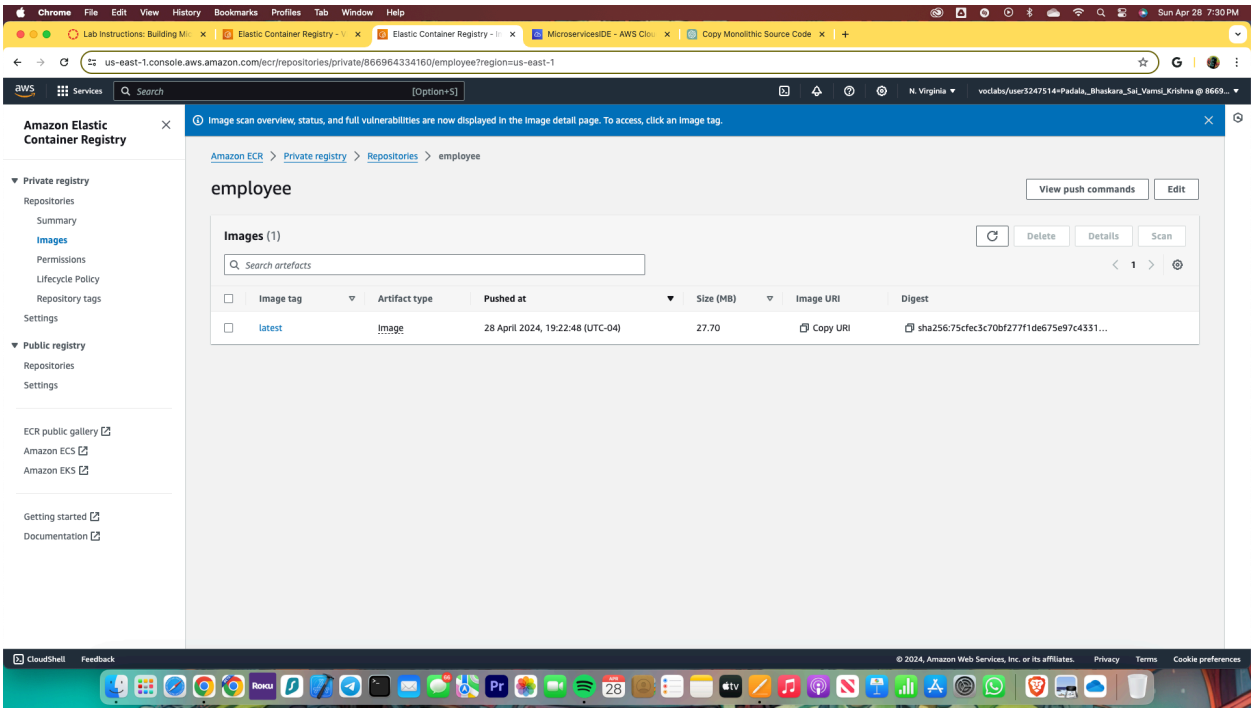
Now we will run the appropriate docker command to push each of the Docker images to Amazon ECR.



Now we successfully pushed both customer and employee



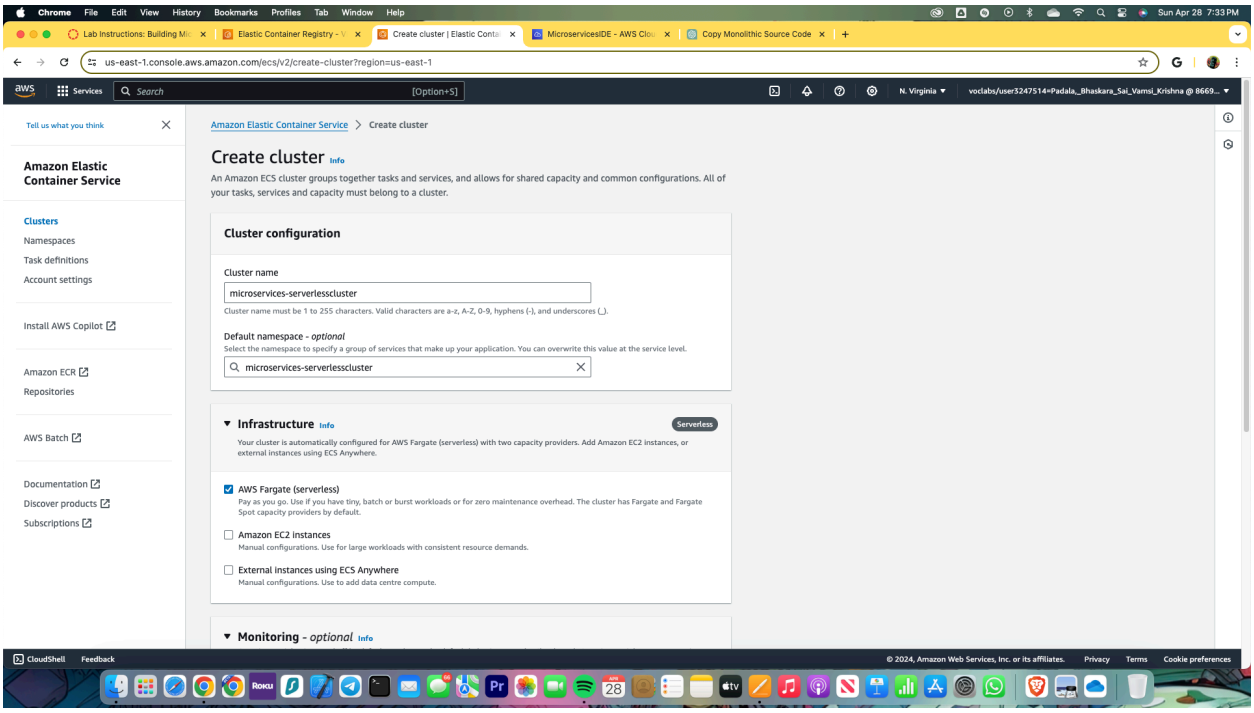


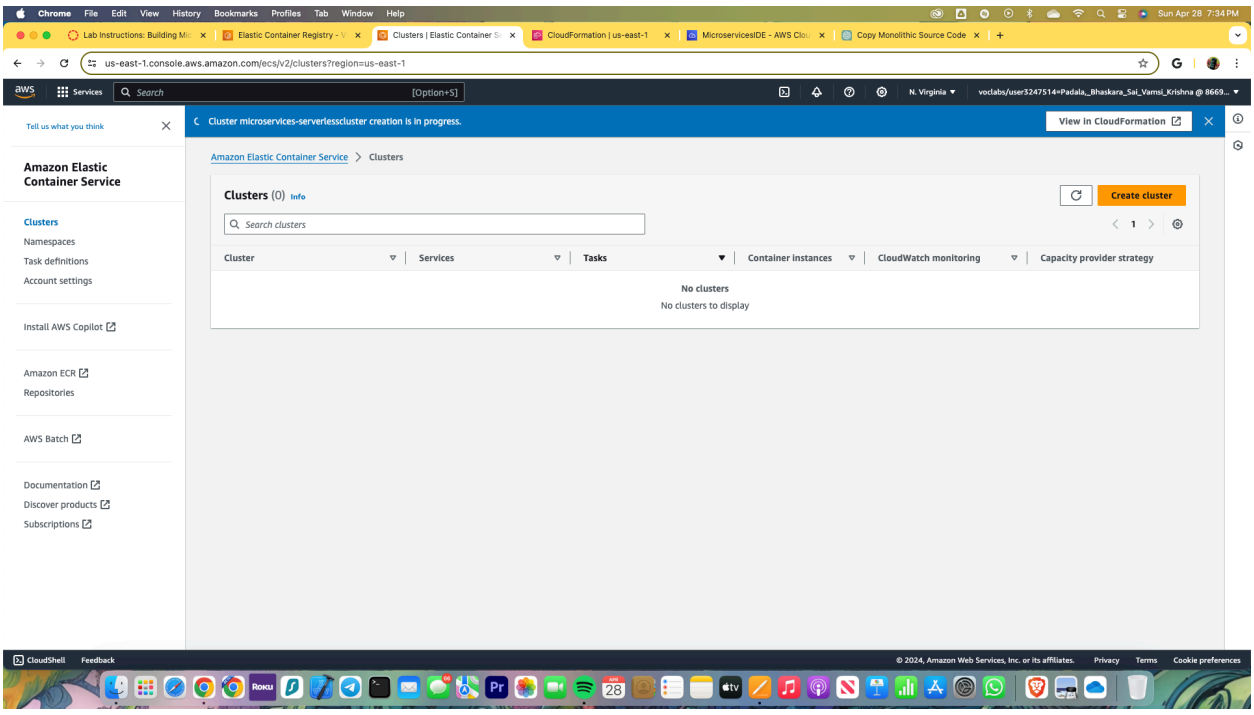


Task 5.2: Create an ECS cluster

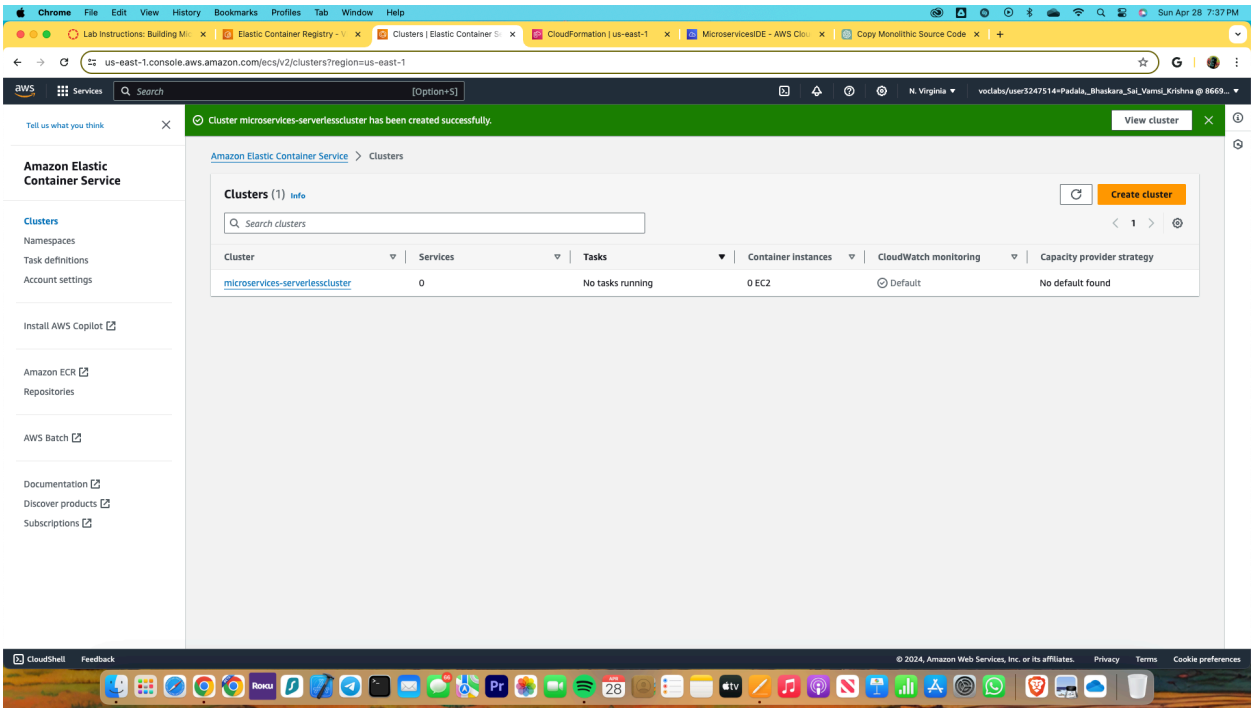
Here we will create an Amazon ECS cluster. Then we will create a serverless AWS Fargate cluster that is named microservices-serverlesscluster

We need to ensure that it's configured to use LabVPC, PublicSubnet1, and PublicSubnet2 (remove any other subnets).





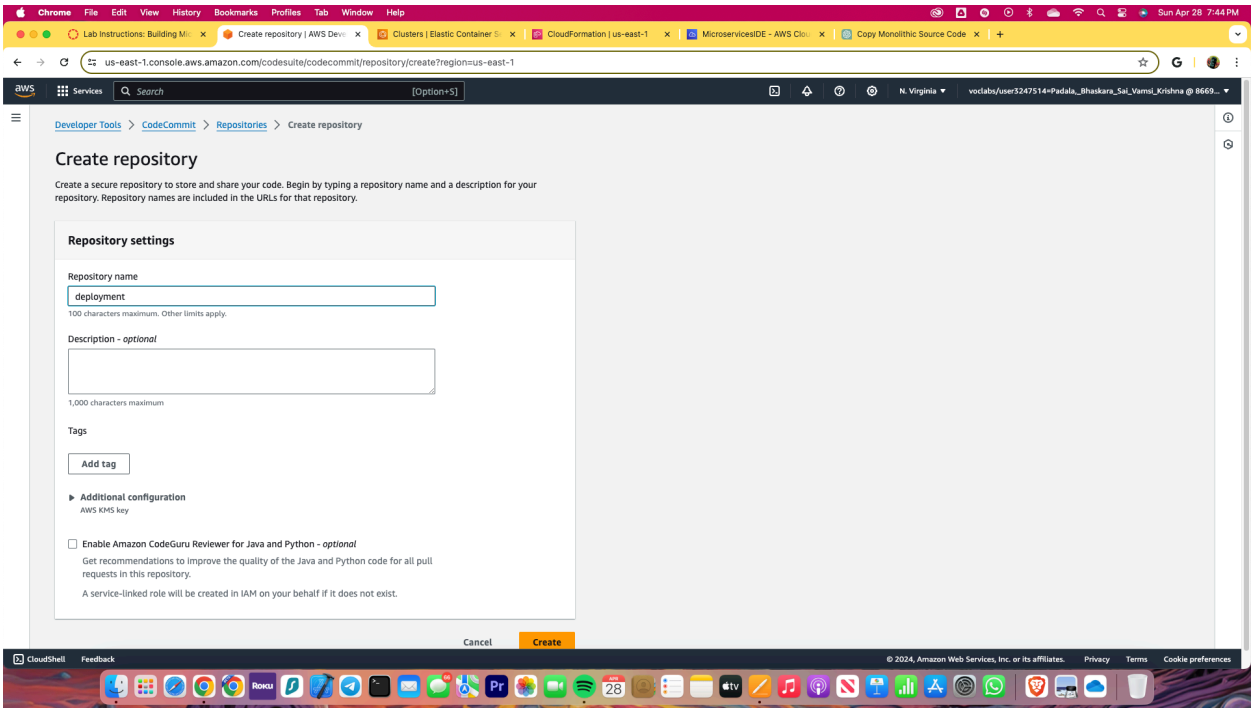
After choosing the button to create the cluster, in the banner that appears across the top of the page, choose View in CloudFormation. We need to Wait until the stack that creates the cluster attains the status CREATE_COMPLETE before you proceed to the next task. If the stack fails to

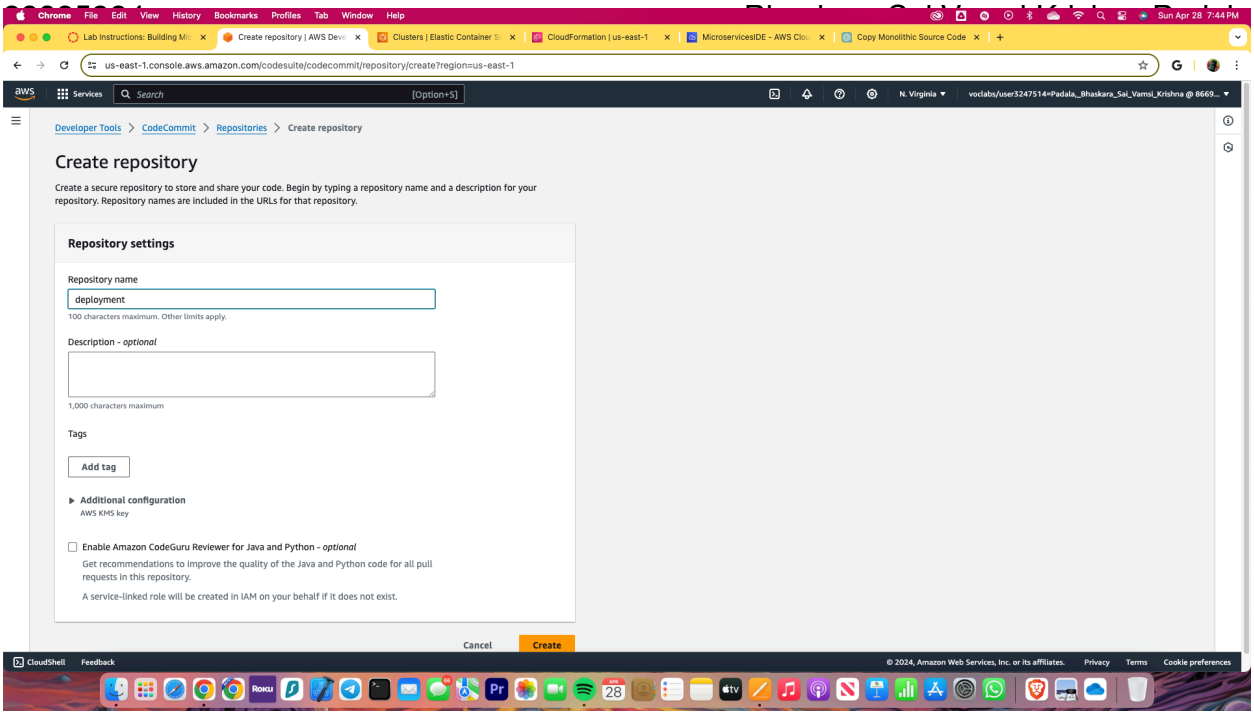


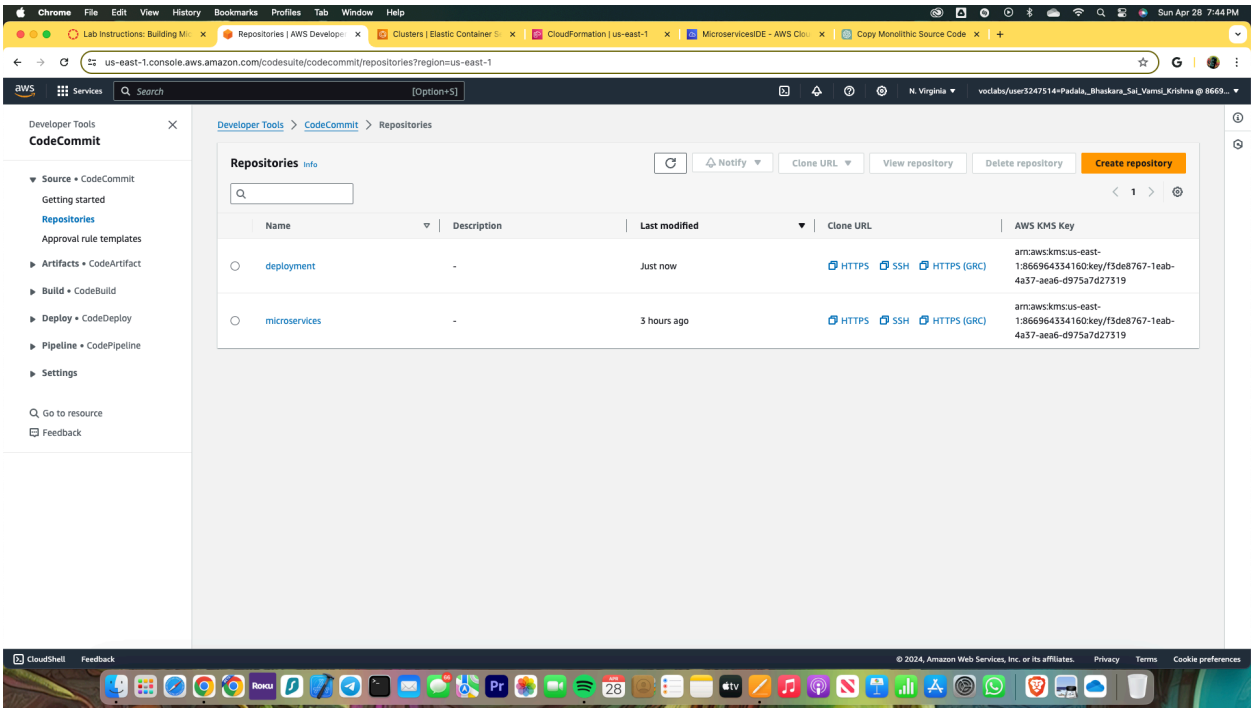
create for any reason and therefore rolls back, repeat these steps to try again. It should succeed the second time.

Task 5.3: Create a CodeCommit repository to store deployment files

In this task, we will create another CodeCommit repository. This repository will store the task configuration specification files that Amazon ECS will use for each microservice. The repository will also store AppSpec specification files that CodeDeploy will use for each microservice. Here we will Create a new CodeCommit repository that is named deployment to store deployment configuration files.

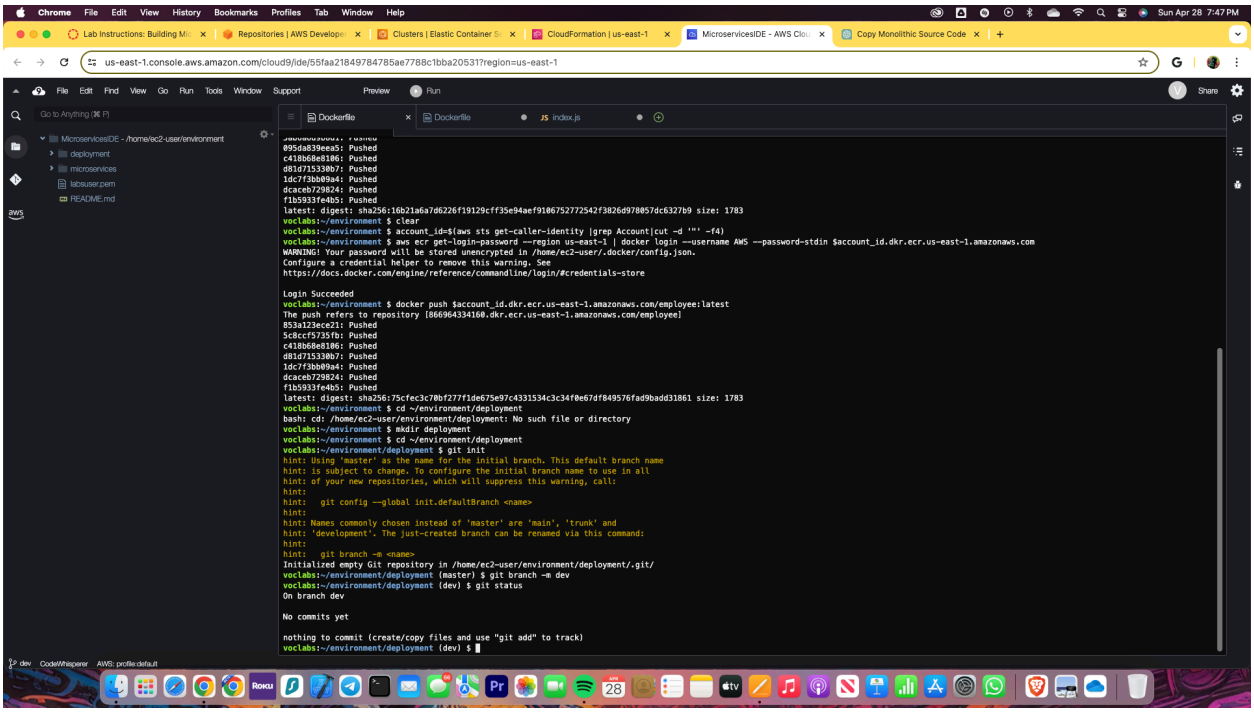






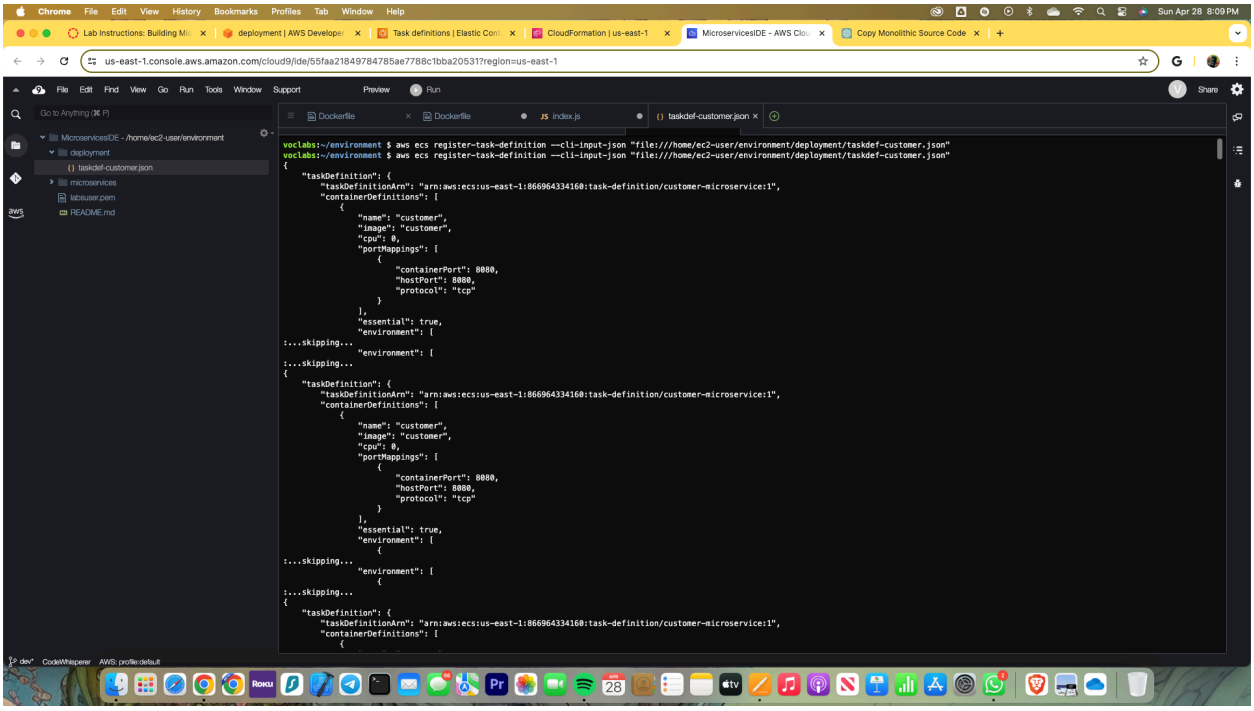
Now In AWS Cloud9, in the environment directory, we need to create a new directory that is named deployment. Initialize the directory as a Git repository with a branch named dev.

Now, let’s perform the Git functions One by one, and we will get the following output on the screen.



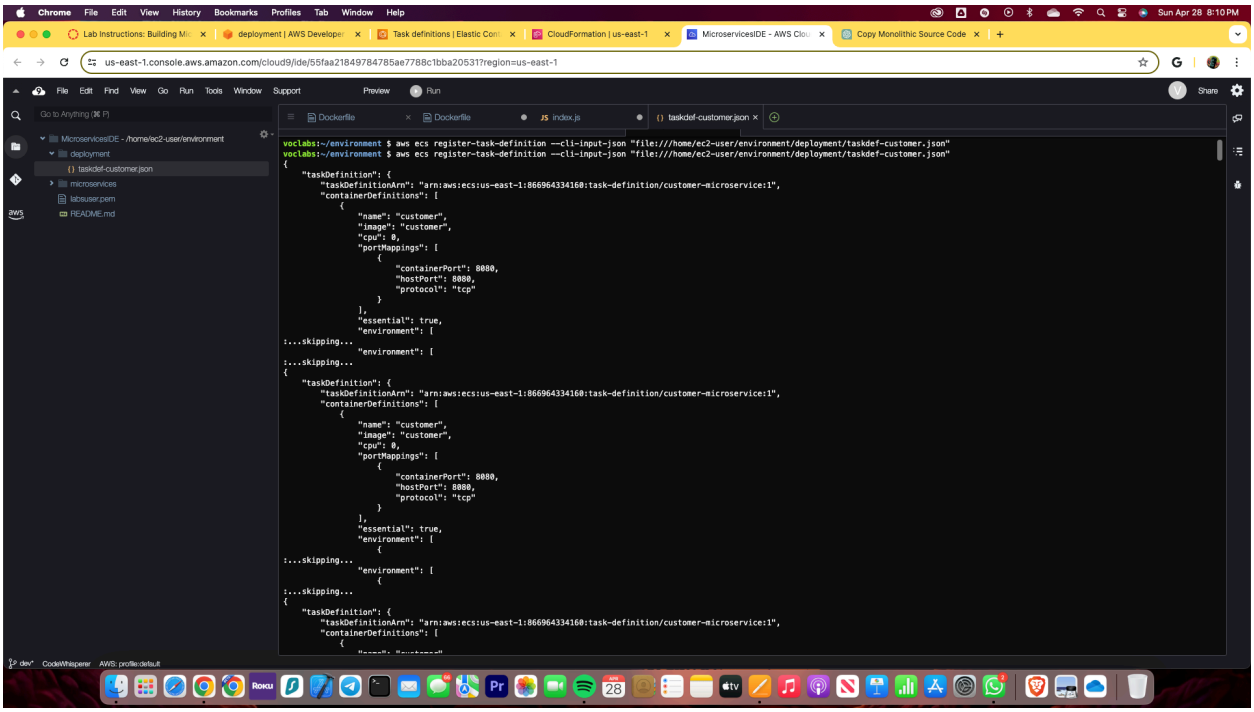
Task 5.4: Create task definition files for each microservice and register them with Amazon ECS
In this task, we will create a task definition file for each microservice and then register the task definitions with Amazon ECS. In the new deployment directory, we will create an empty file named taskdef-customer.json

Now we will Edit the taskdef-customer.json file.

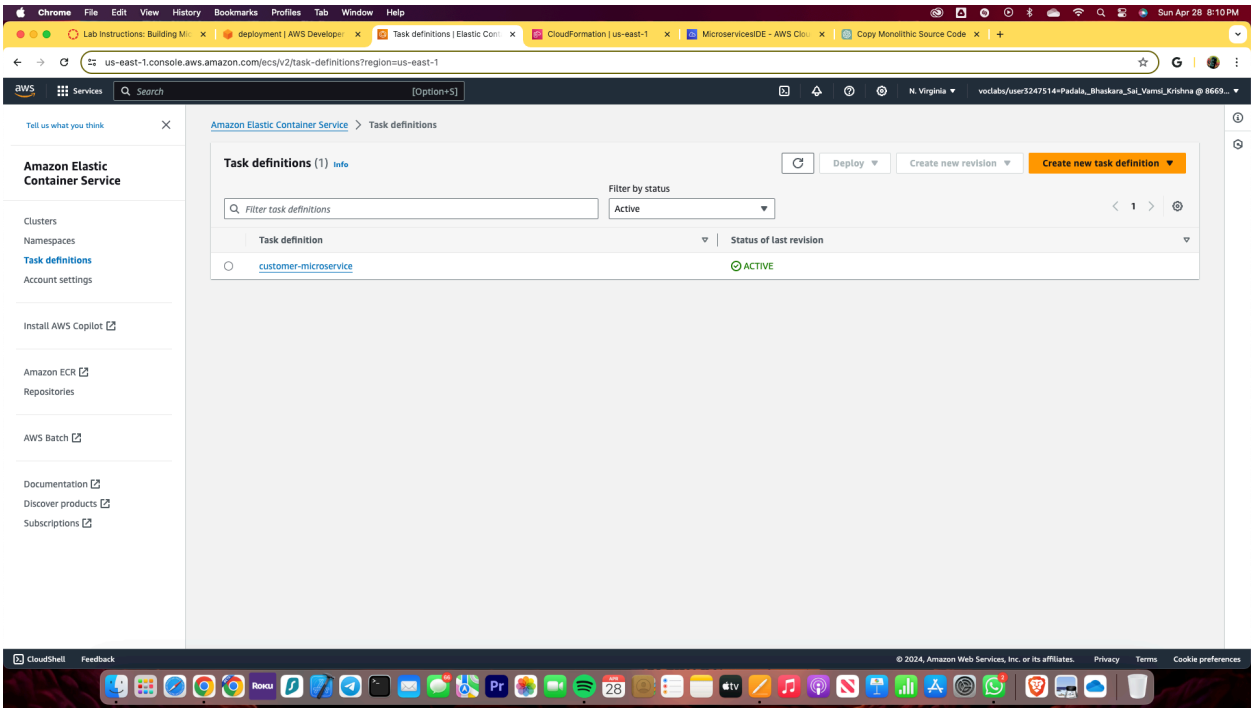


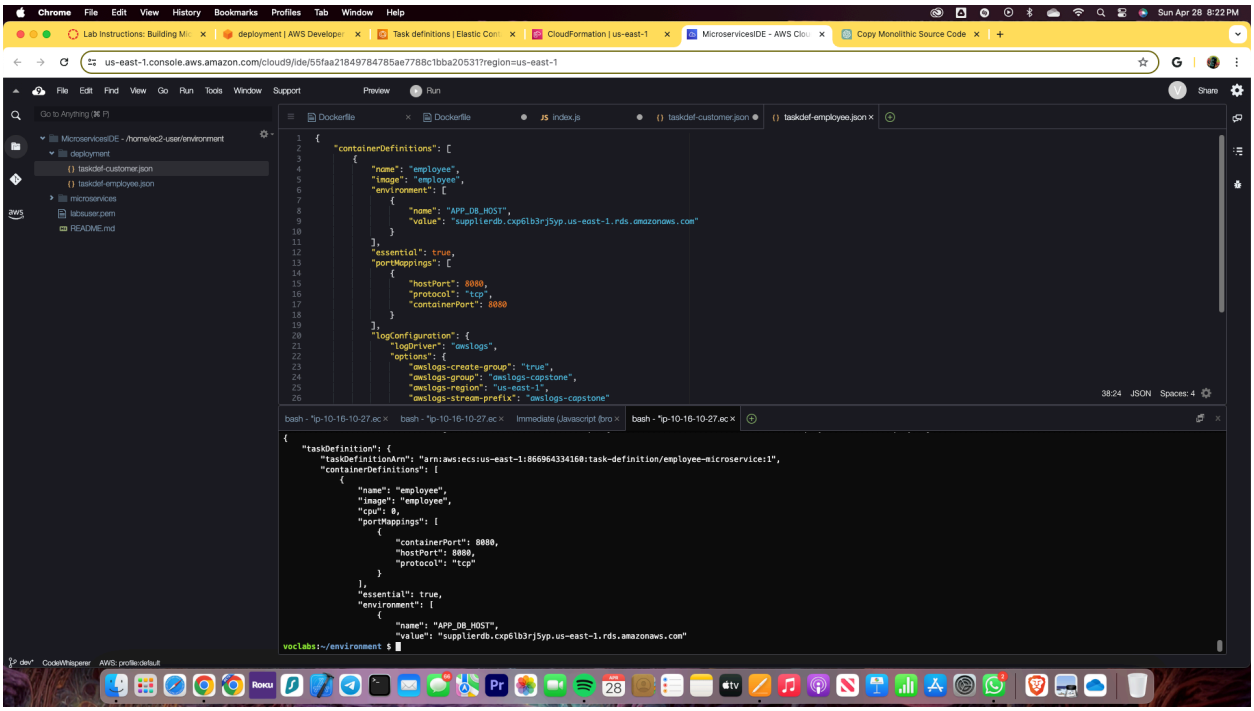
Now we will register the customer microservice task definition in Amazon ECS, run the following command:

aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-customer.json"



In the Amazon ECS console, we need to verify that the customer-microservice task definition now appears in the Task definitions pane. Also, we can notice that the revision number displays after the task definition name.





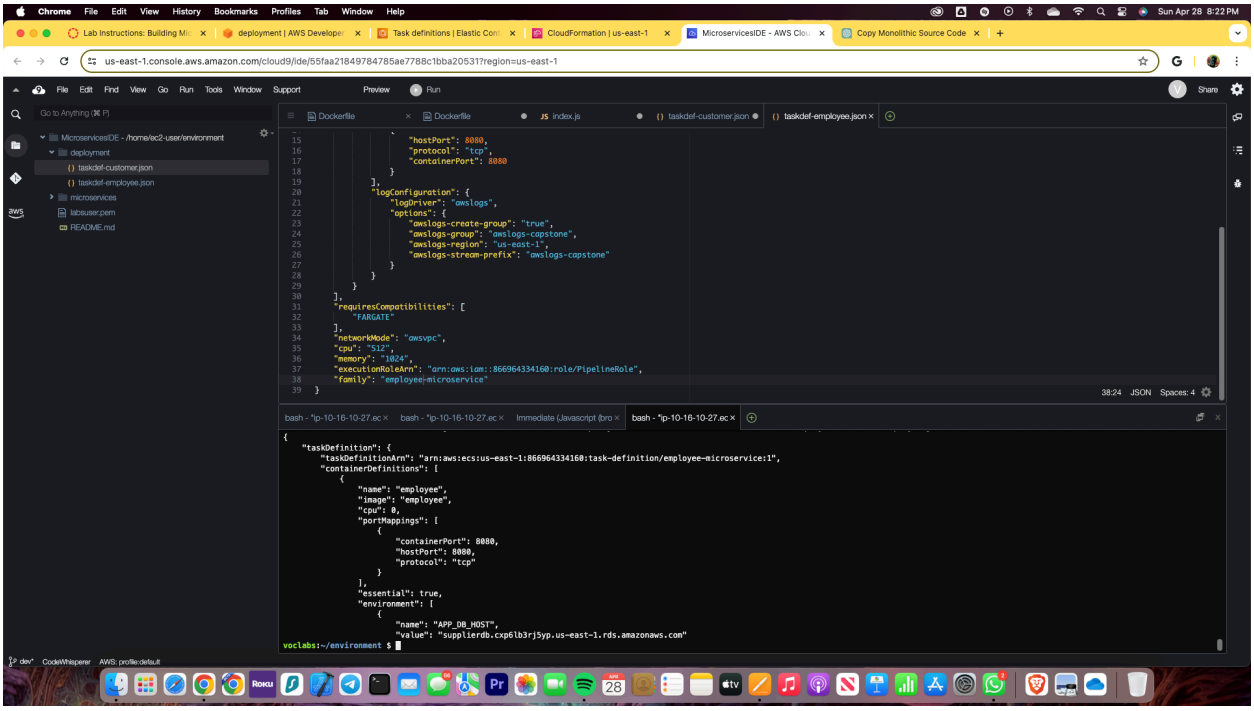
In the deployment directory, create a taskdef-employee.json specification file.

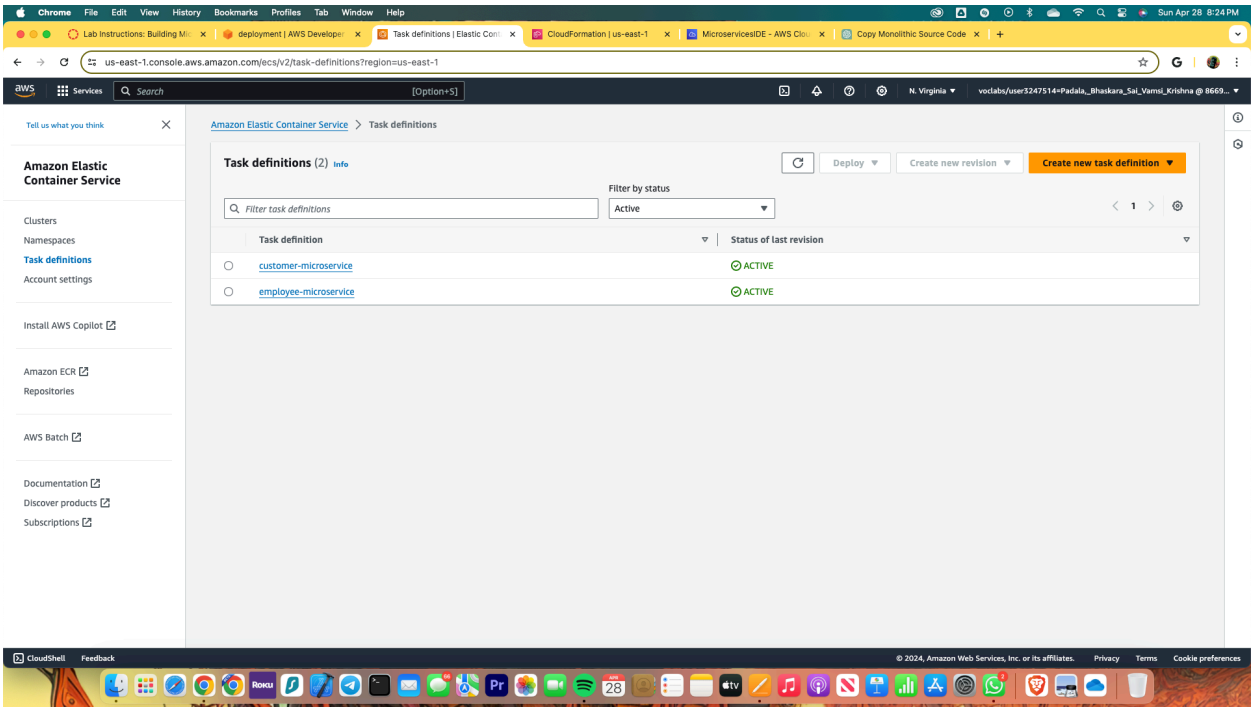
Add the same JSON code to it that currently exists in the taskdef-customer.json file (where we have already set the account ID and RDS endpoints).

After we paste in the code, change the three occurrences of customer to employee

To register the employee task definition with Amazon ECS, run an AWS CLI command.

In the Amazon ECS console, we need to verify that the employee-microservice task definition now appears in the Task definitions pane. Also, we notice that the revision number displays after the task definition name.



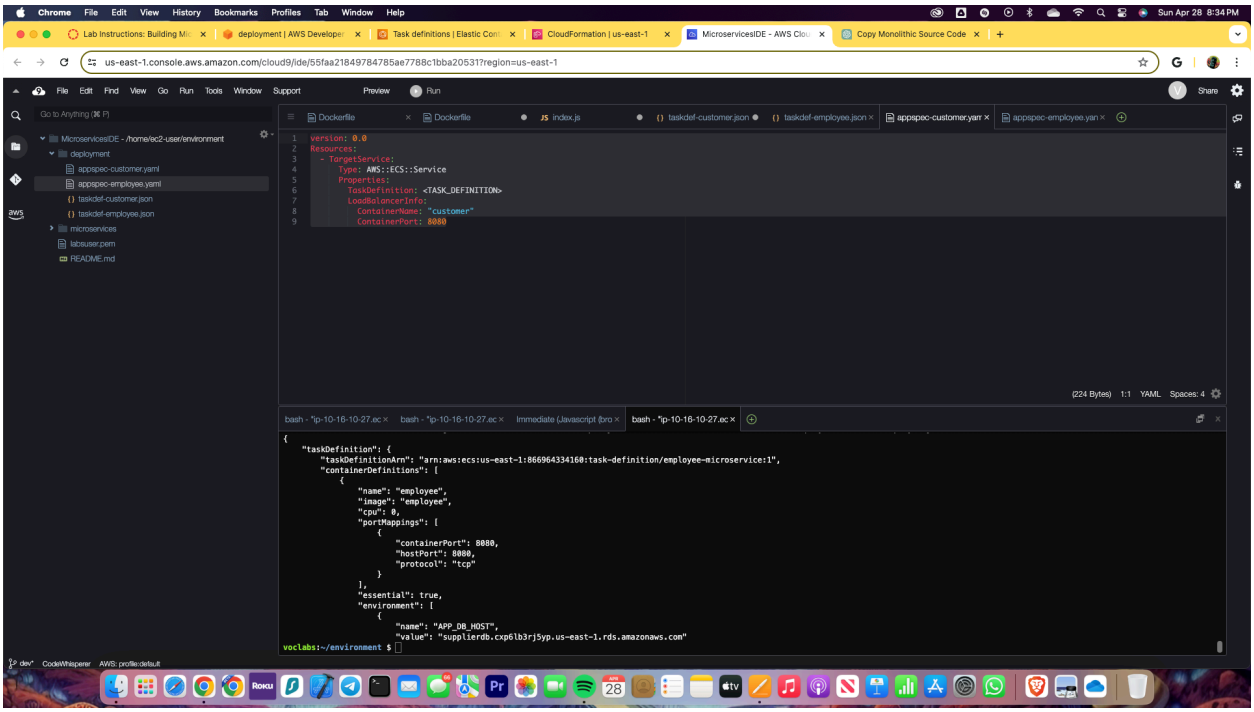


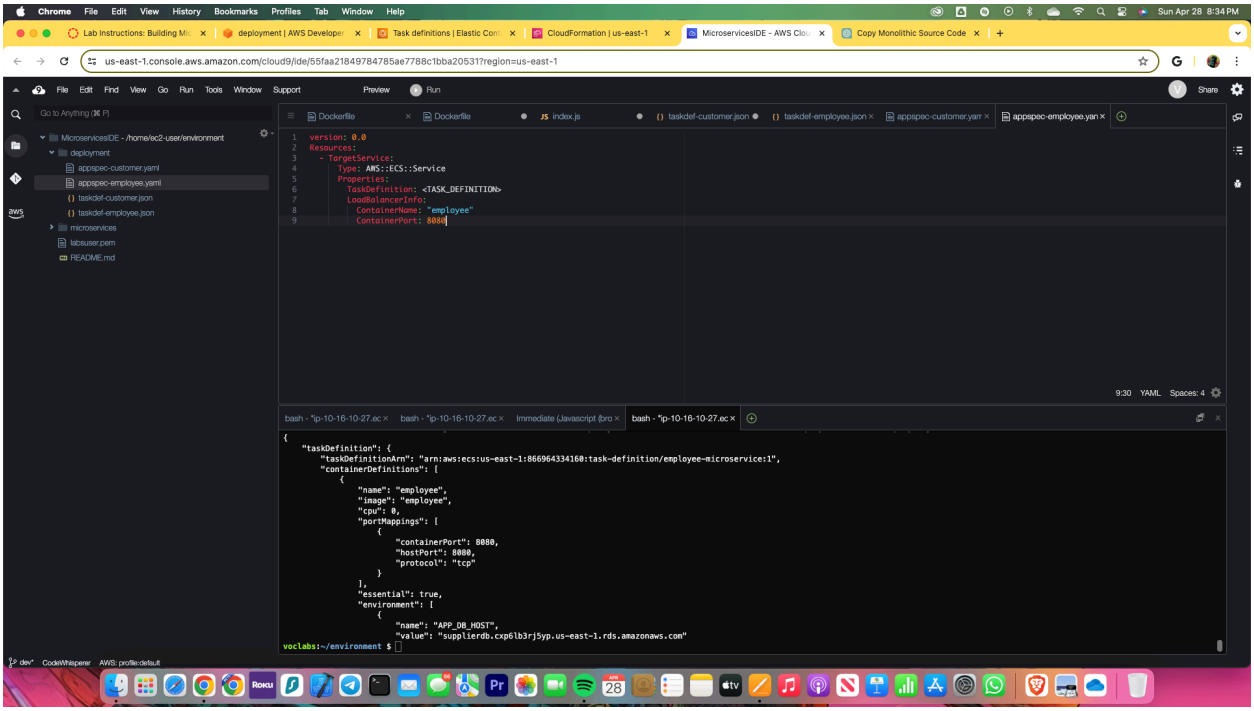
Task 5.5: Create AppSpec files for CodeDeploy for each microservice

In this task, we will continue to complete tasks to support deploying the microservices-based web application to run on an ECS cluster where the deployment is supported by a CI/CD pipeline. In this specific task, we will create two application specification (AppSpec) files, one for each microservice. These files will provide instructions for CodeDeploy to deploy the microservices to the Amazon ECS on Fargate infrastructure.

Firstly, Create an AppSpec file for the customer microservice.

In the deployment directory, create a new file named appspec-customer.yaml





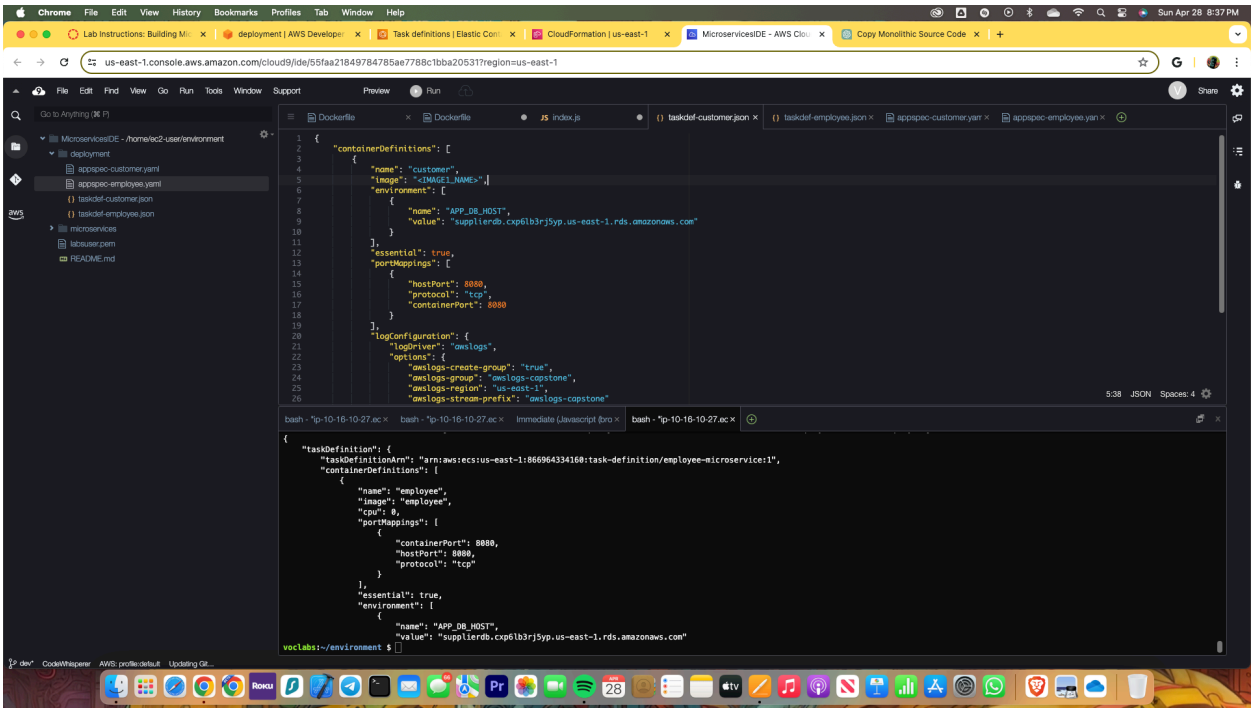
In the same directory, create an AppSpec file for the employee microservice.

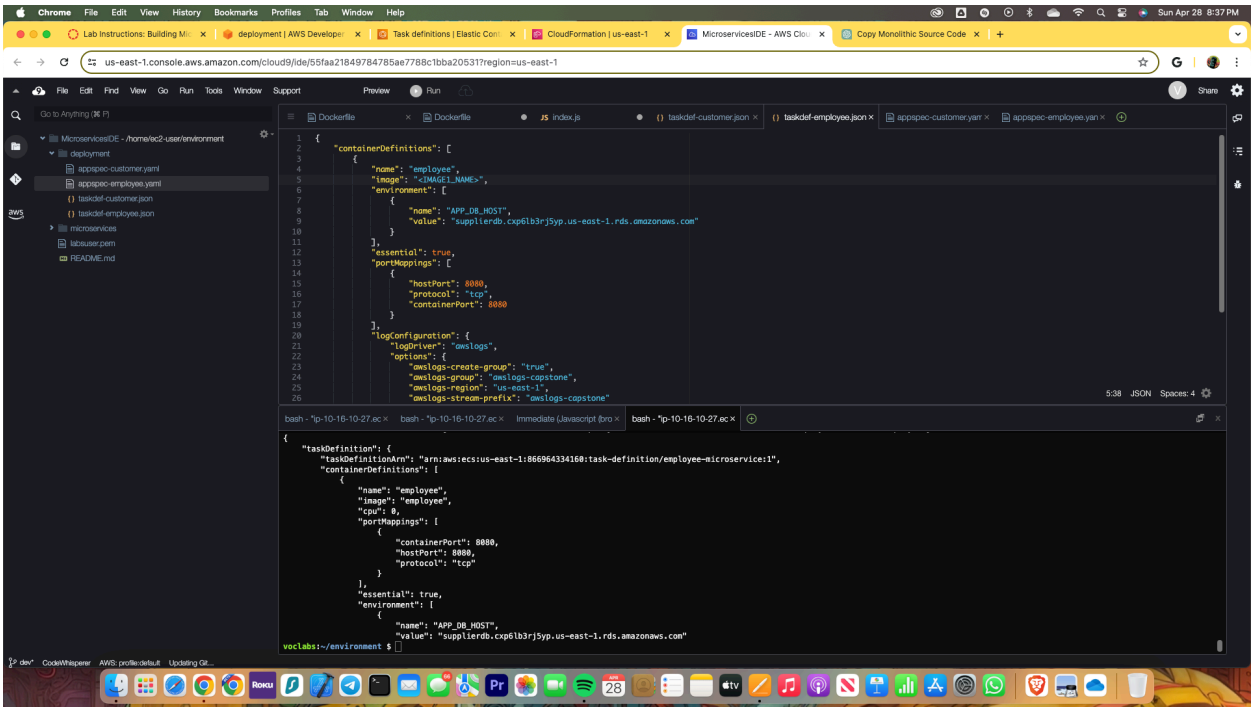
Name the file appspec-employee.yaml.

The contents of the file should be the same as the appspec-customer.yaml file. However, we need to change customer`` on thecontainerNameline to beemployee`

Task 5.6: Update files and check them into CodeCommit

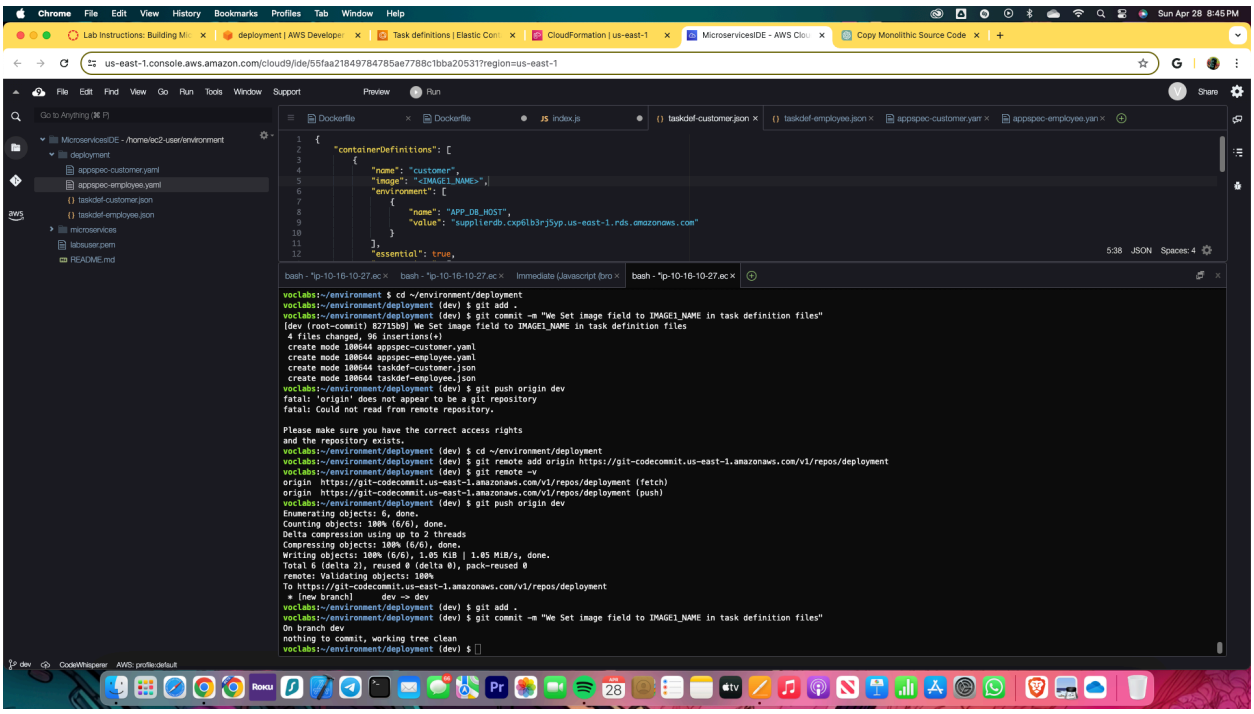
In this task, we will update the two task definition files. Then, we will push the four files that you created in the last two tasks into the deployment repository.





Now we need to Push all four files to CodeCommit.

Here we need to remember that Pushing the latest files to CodeCommit is essential. Later, when we create the CI/CD pipeline, the pipeline will pull these files from CodeCommit and use the details in them as instructions to deploy updates for your microservices to the Amazon ECS cluster.



Phase 6: Creating target groups and an Application Load Balancer

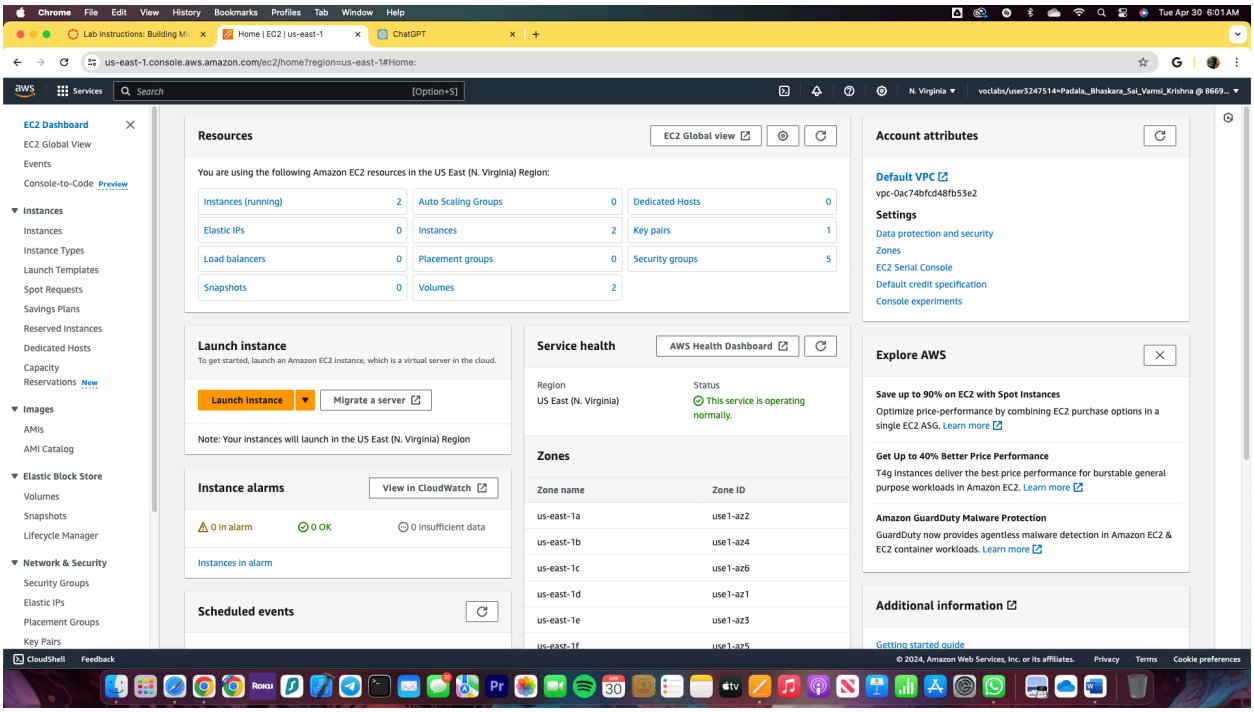
In this phase, we will create an Application Load Balancer, which provides an endpoint URL. This URL will act as the HTTPS entry point for customers and employees to access our application through a web browser.

The load balancer will have listeners, which will have routing and access rules that determine which target group of running containers the user request should be directed to.

Task 6.1: Create four target groups

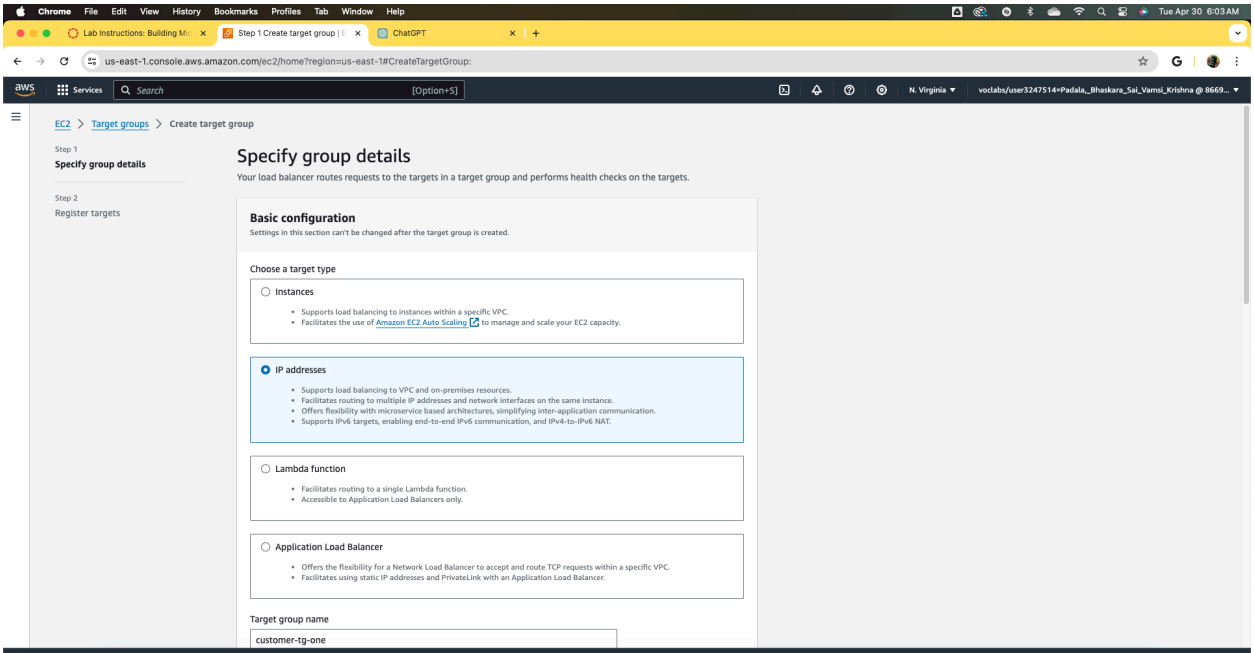
In this task, we will create four target groups—two for each microservice. Because we will configure a blue/green deployment, CodeDeploy requires two target groups for each deployment group.

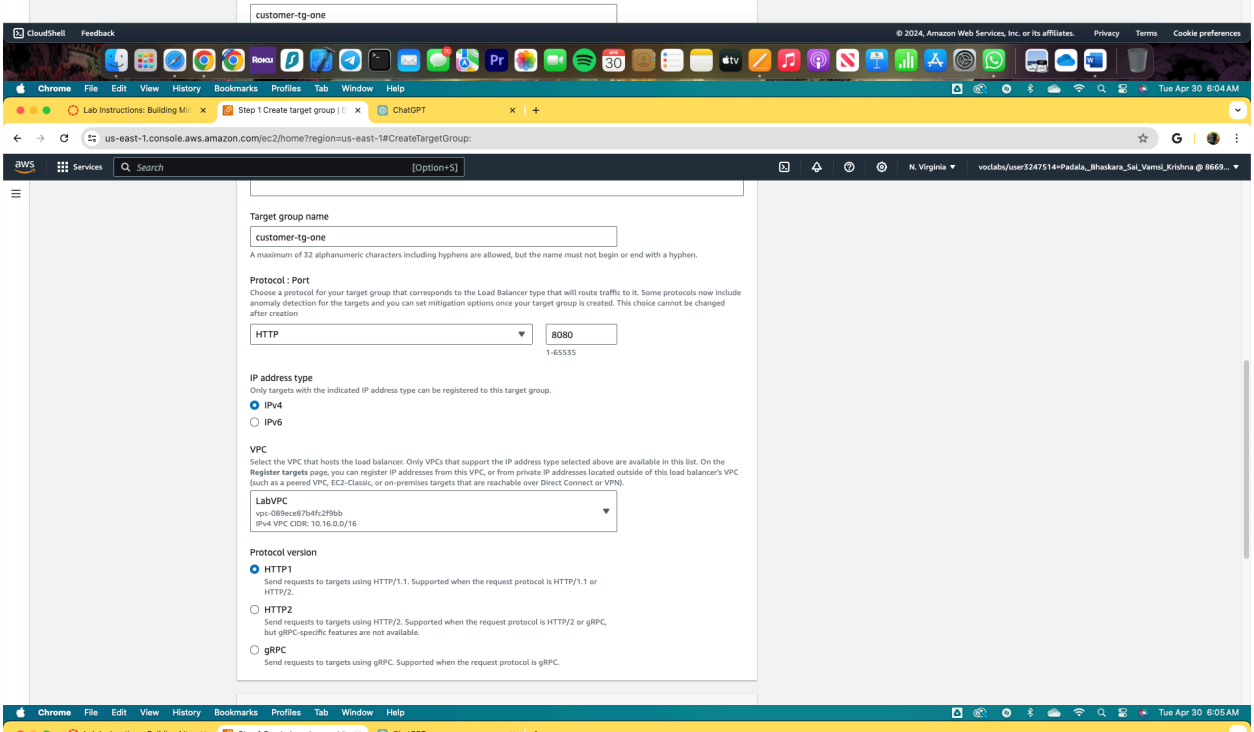
Firstly, we need to go to EC2 page as follows.

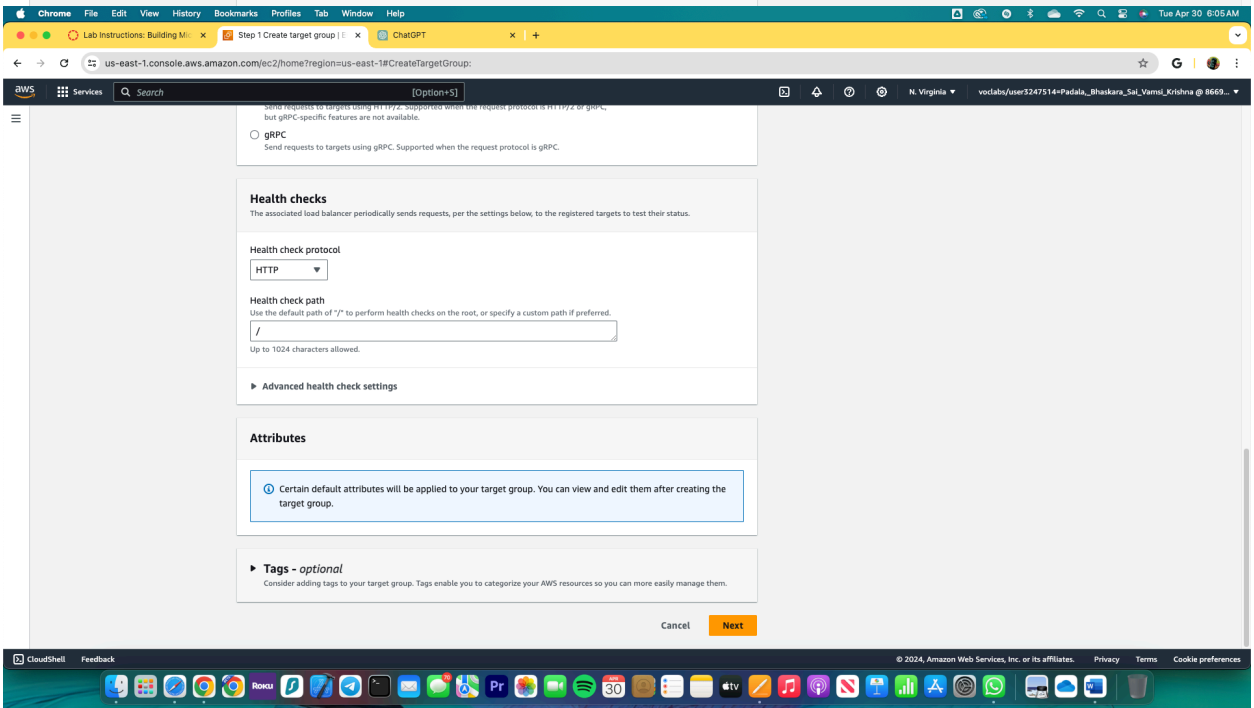


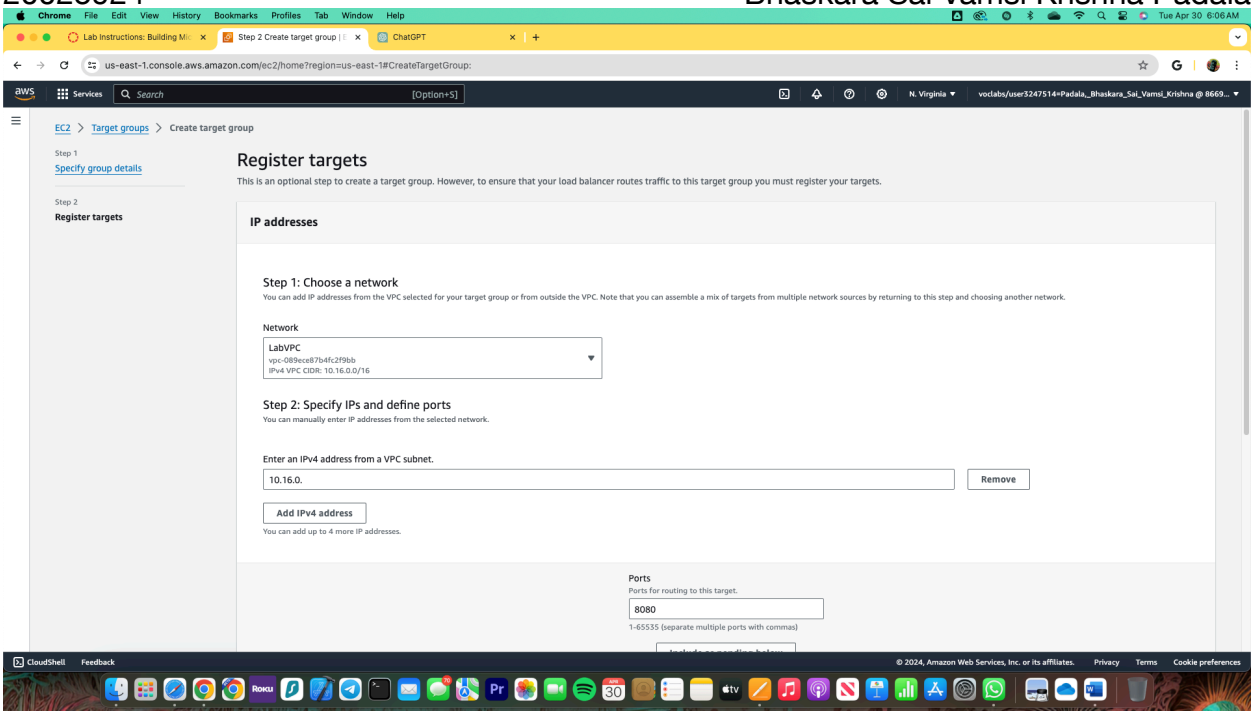
Now go to Target groups and create one as follows. Be sure to use the same configurations as shown.

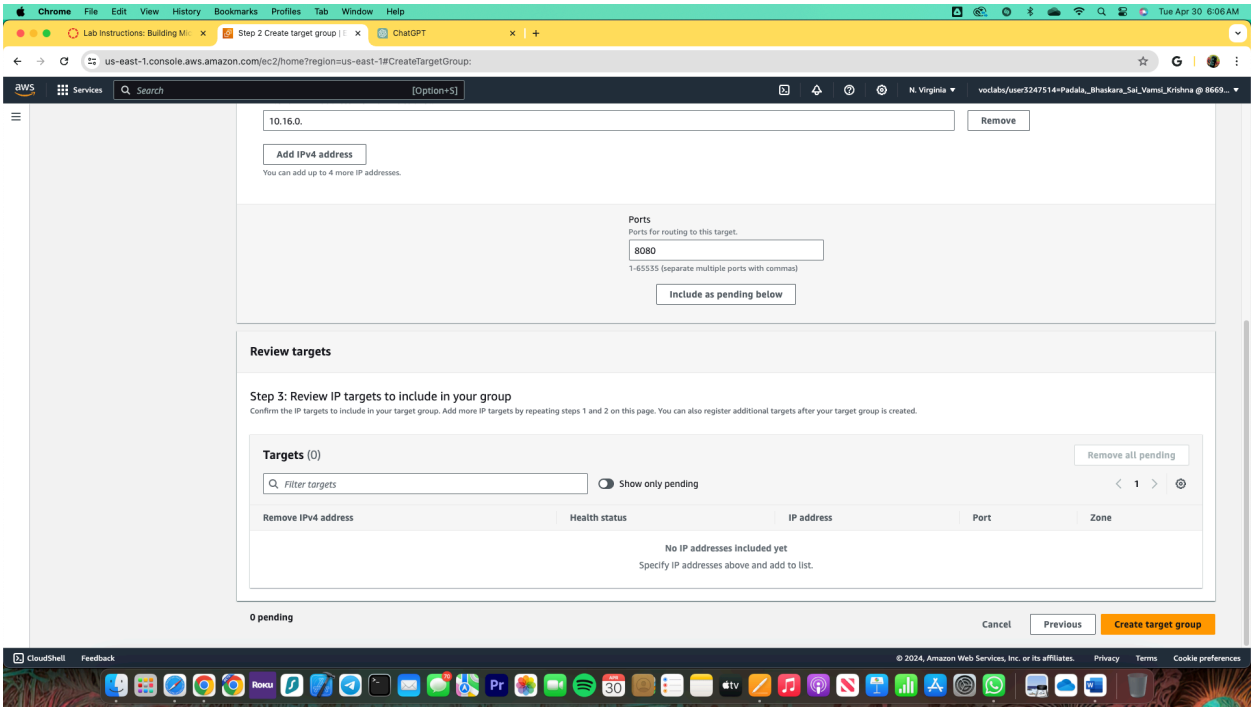
Navigate to the Amazon EC2 console.
In the navigation pane, choose Target Groups.
Choose Create target group and configure the following:
Choose a target type: Choose IP addresses.
Target group name: Enter customer-tg-one
Protocol: Choose HTTP.
Port: Enter 8080
VPC: Choose LabVPC.
Health check path: Enter /
Choose Next.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:03 AM
Lab Instructions: Building Mic
Step 1 Create target group | E
ChatGPT
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:
aws Services Search [Option+S]
N. Virginia
voclabs/user3247514=Padala,_Bhaskara_Sai_Vamsi_Krishna @ 8669...
EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Specify group details
Your load balancer routes requests to the targets in a target group and performs health checks on the targets.
Basic configuration
Settings in this section can't be changed after the target group is created.
Choose a target type
Instances
Supports load balancing to instances within a specific VPC.
Facilitates the use of Amazon EC2 Auto Scaling to manage and scale your EC2 capacity.
IP addresses
Supports load balancing to VPC and on-premises resources.
Facilitates routing to multiple IP addresses and network interfaces on the same instance.
Offers flexibility with microservice based architectures, simplifying inter-application communication.
Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
Lambda function
Facilitates routing to a single Lambda function.
Accessible to Application Load Balancers only.
Application Load Balancer
Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.
Target group name
customer-tg-one

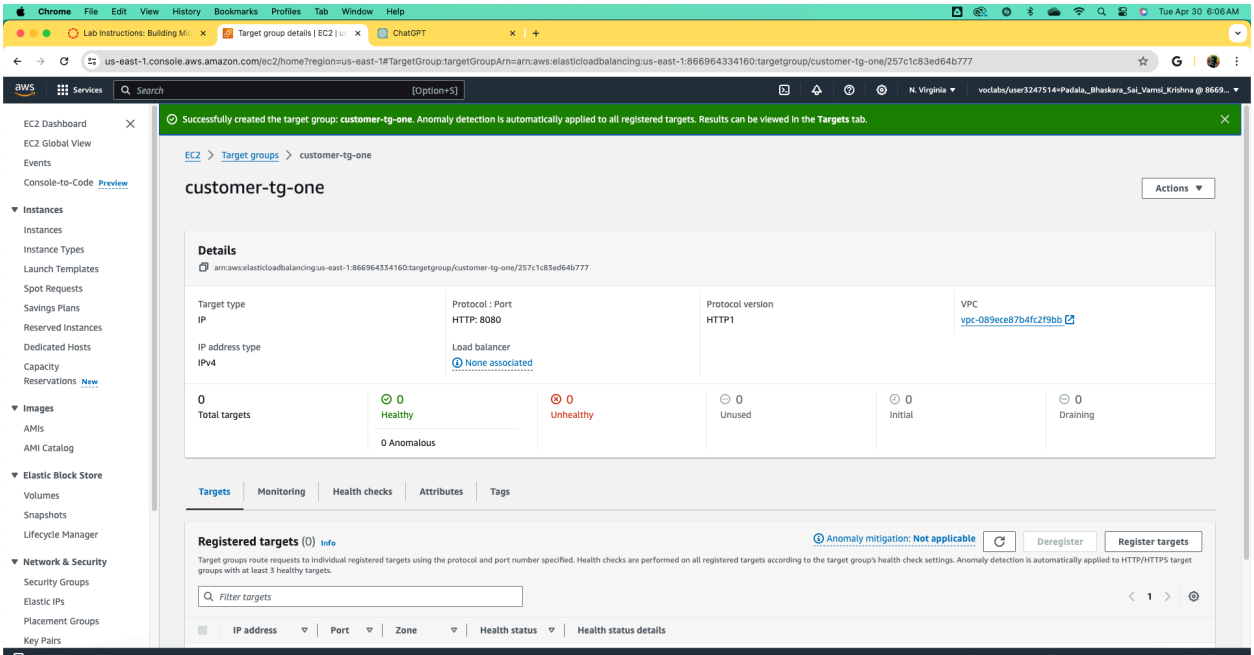

CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:04 AM
Target group name
customer-tg-one
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.
Protocol : Port
Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols now include anomaly detection for the targets and you can set mitigation options once your target group is created. This choice cannot be changed after creation
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).
LabVPC
vpc-089ece87b4fc2f9bb
IPv4 VPC CIDR: 10.16.0.0/16
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.

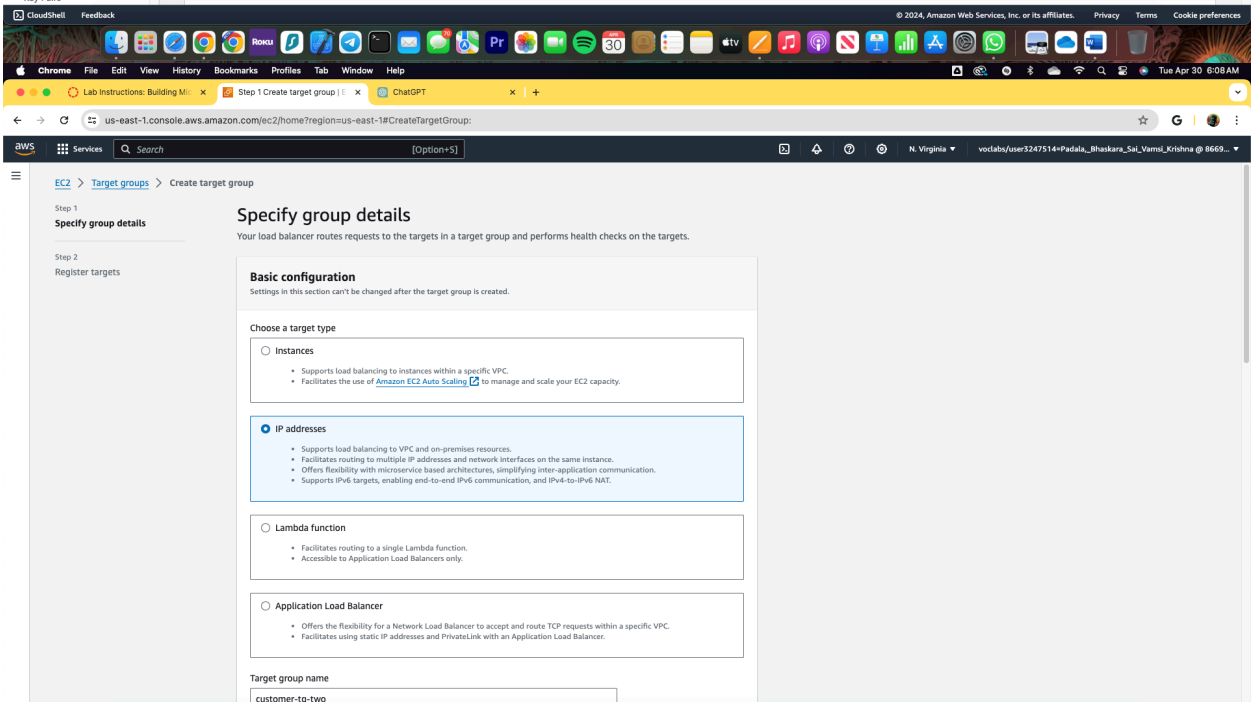

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:05 AM
gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.
Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test 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
Attributes
Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.
Tags - optional
Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.
Cancel
Next
CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

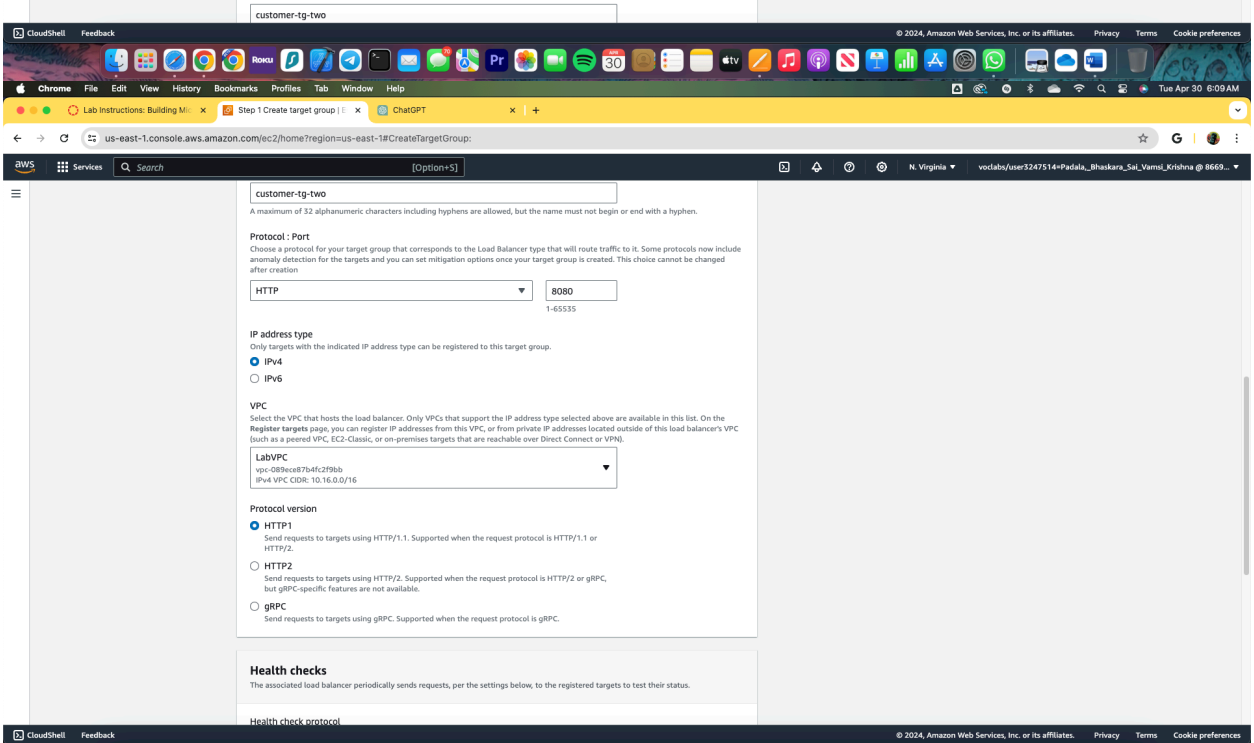


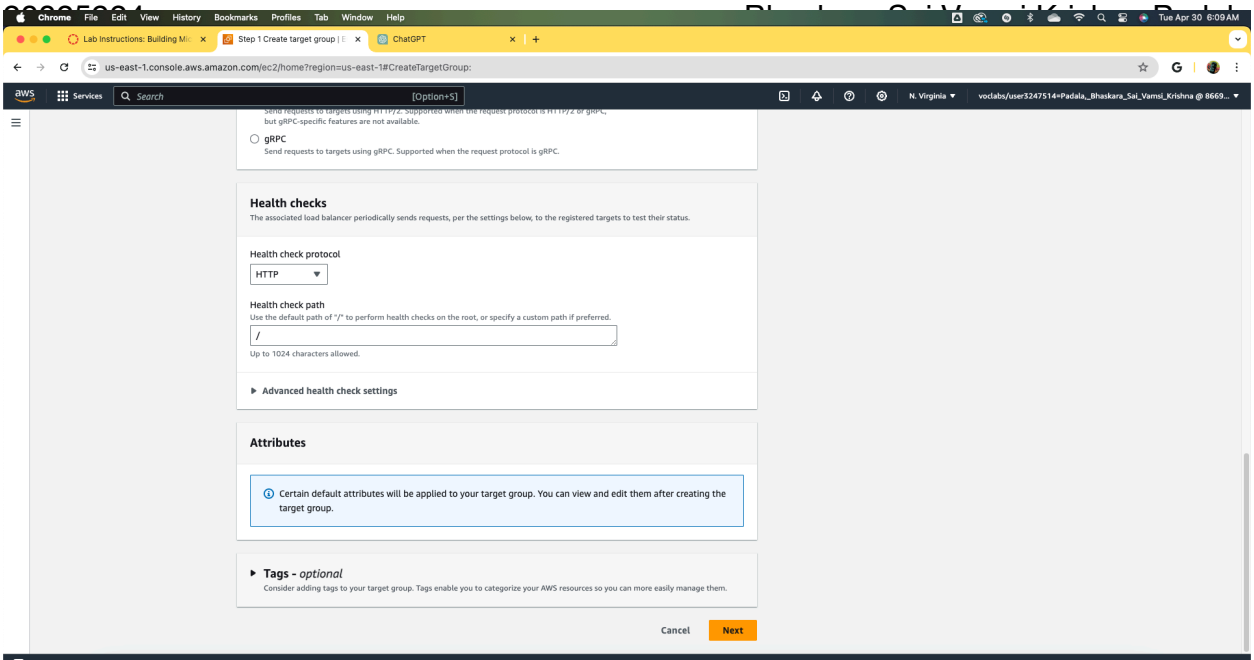


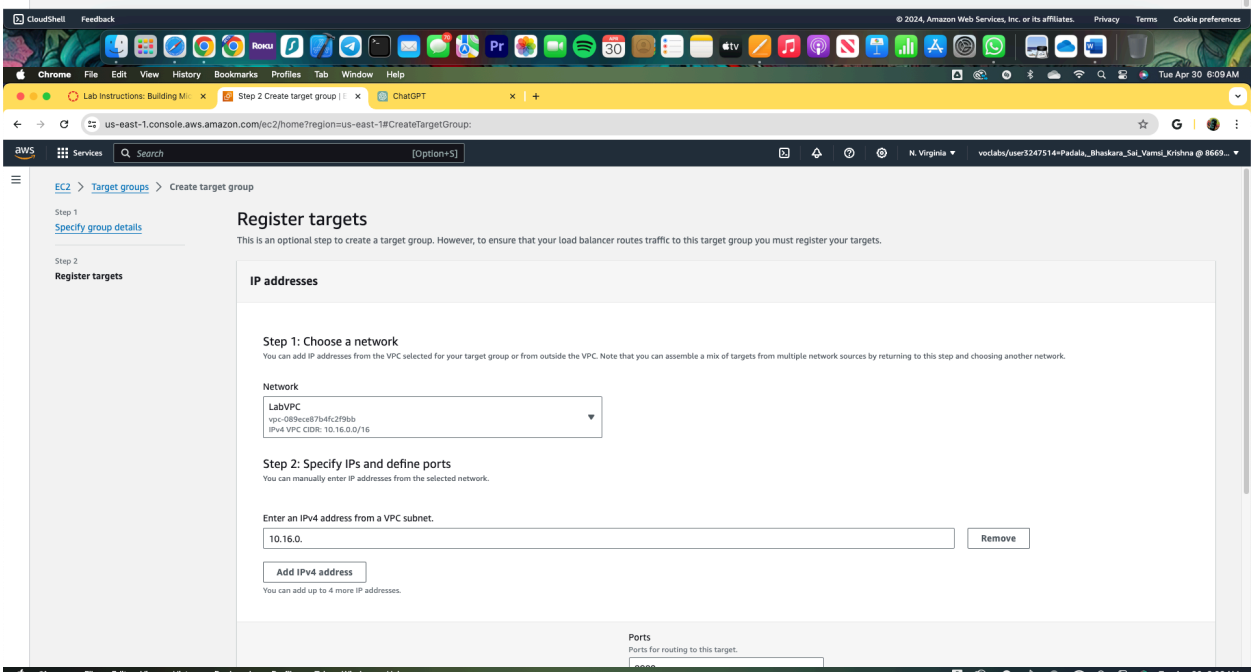
Now click on the Create Target Group and follow the further steps as follows. Here in the next screen we can observe that the Target Group got created.

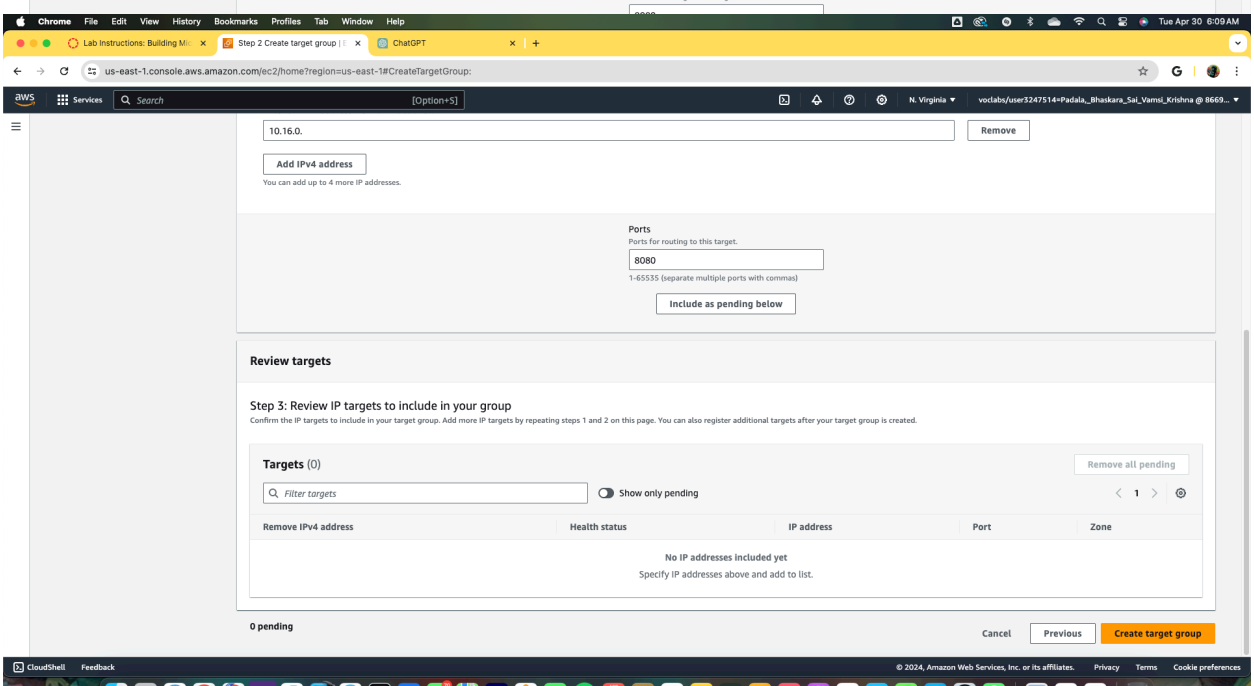

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:06 AM
Lab Instructions: Building Mic
Target group details | EC2 | us
ChatGPT
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#TargetGroup:targetGroupArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-one/257c1c83ed64b777
Services
Search
[Option+S]
N. Virginia
voclabs/user3247514=Padala,_Bhaskara_Sai_Vamsi_Krishna @ 8669...
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Successfully created the target group: customer-tg-one. Anomaly detection is automatically applied to all registered targets. Results can be viewed in the Targets tab.
EC2 > Target groups > customer-tg-one
customer-tg-one
Actions
Details
arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-one/257c1c83ed64b777
Target type
IP
Protocol : Port
HTTP: 8080
Protocol version
HTTP1
VPC
vpc-089ece87b4fc2f9bb
IP address type
IPv4
Load balancer
None associated
0
Total targets
0
Healthy
0 Anomalous
0
Unhealthy
0
Unused
0
Initial
0
Draining
Targets
Monitoring
Health checks
Attributes
Tags
Registered targets (0) Info
Anomaly mitigation: Not applicable
Deregister
Register targets
Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.
Filter targets
IP address
Port
Zone
Health status
Health status details
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:08 AM
Lab Instructions: Building Mic
Step 1 Create target group | E
ChatGPT
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:
Services
Search
[Option+S]
N. Virginia
voclabs/user3247514=Padala,_Bhaskara_Sai_Vamsi_Krishna @ 8669...
EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Specify group details
Your load balancer routes requests to the targets in a target group and performs health checks on the targets.
Basic configuration
Settings in this section can't be changed after the target group is created.
Choose a target type
Instances
Supports load balancing to instances within a specific VPC.
Facilitates the use of Amazon EC2 Auto Scaling to manage and scale your EC2 capacity.
IP addresses
Supports load balancing to VPC and on-premises resources.
Facilitates routing to multiple IP addresses and network interfaces on the same instance.
Offers flexibility with microservice based architectures, simplifying inter-application communication.
Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
Lambda function
Facilitates routing to a single Lambda function.
Accessible to Application Load Balancers only.
Application Load Balancer
Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.
Target group name
customer-tg-two
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

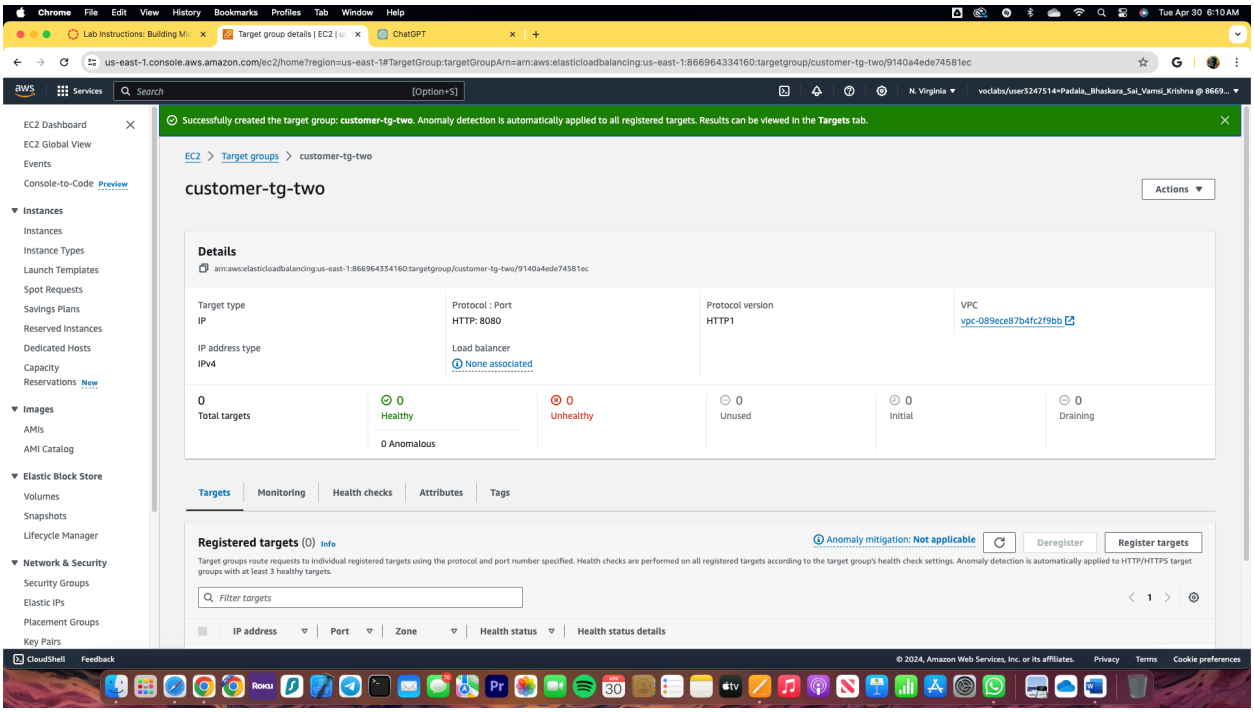

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:09 AM
Lab Instructions: Building Mic
Step 1 Create target group | E
ChatGPT
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:
Services
Search
[Option+S]
N. Virginia
voclabs/user3247514=Padala,_Bhaskara_Sai_Vamsi_Krishna @ 8669...
customer-tg-two
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.
Protocol : Port
Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols now include anomaly detection for the targets and you can set mitigation options once your target group is created. This choice cannot be changed after creation
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).
LabVPC
vpc-089ece87b4fc2f9bb
IPv4 VPC CIDR: 10.16.0.0/16
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
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

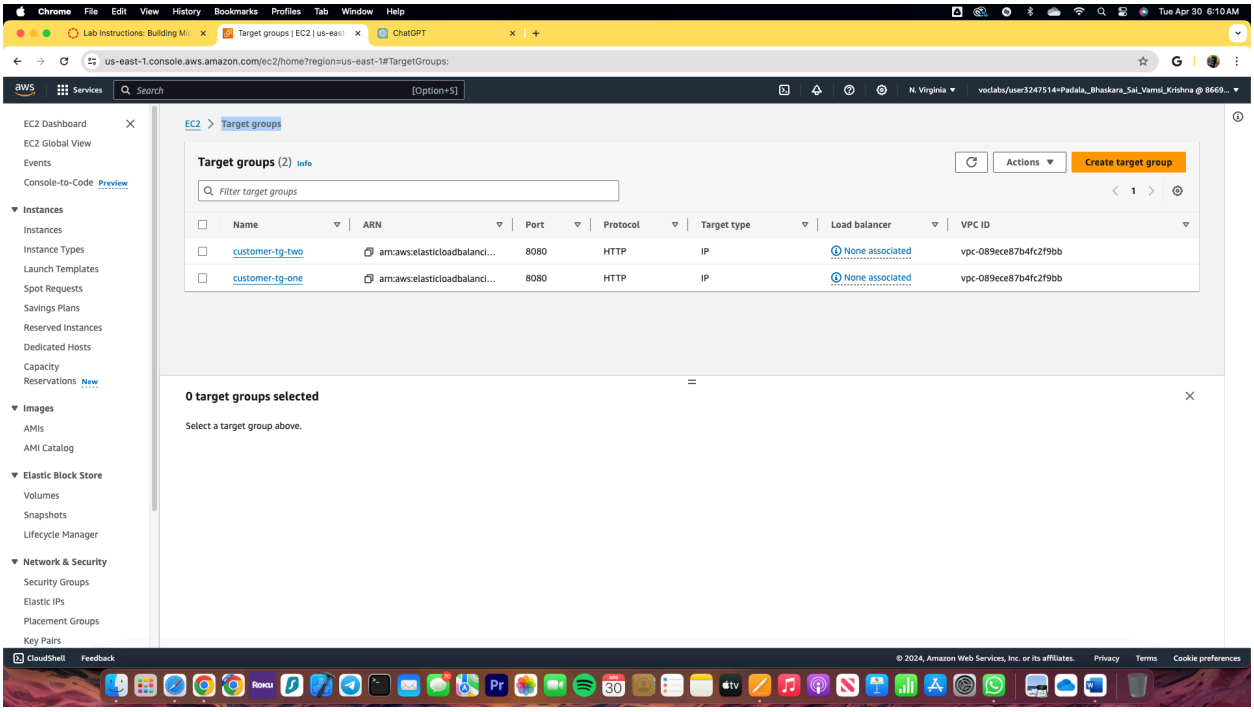

gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.
Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.
Health check protocol
HTTP
Health check path
Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.
/
Up to 1024 characters allowed.
Advanced health check settings
Attributes
Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.
Tags - optional
Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.
Cancel
Next


EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Register targets
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-089ece87b4fc2f9bb
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.
Remove
Add IPv4 address
You can add up to 4 more IP addresses.
Ports
Ports for routing to this target.

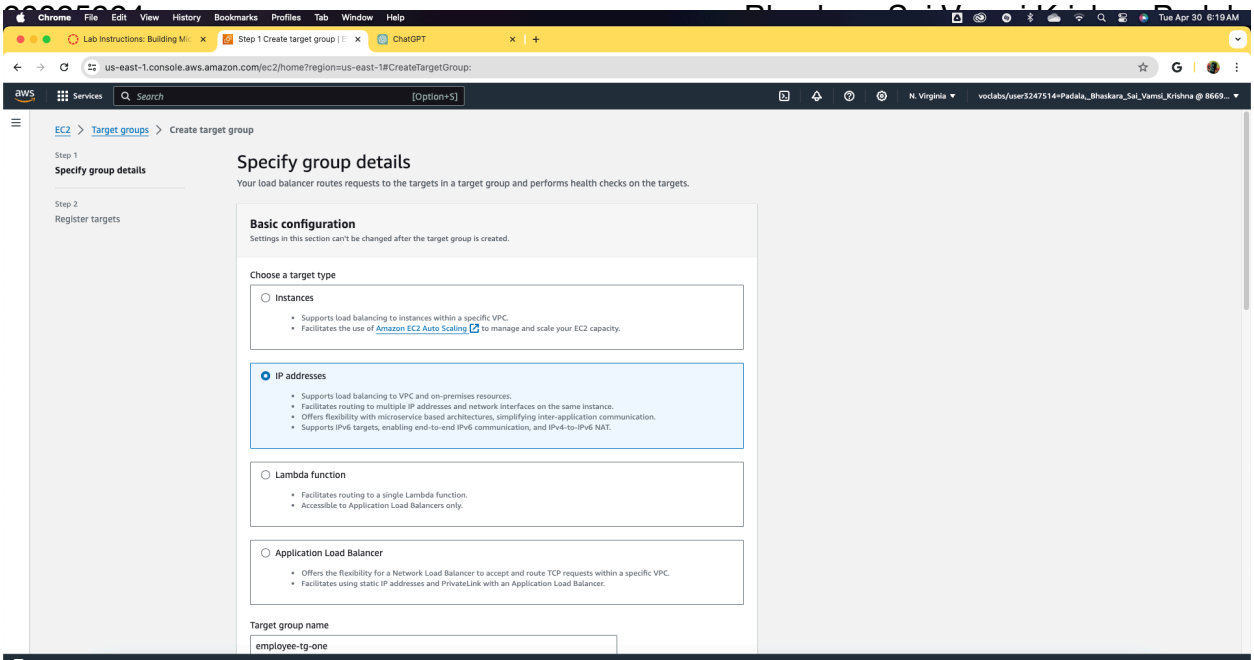

10.16.0.
Remove
Add IPv4 address
You can add up to 4 more IP addresses.
Ports
Ports for routing to this target.
8080
1-65535 (separate multiple ports with commas)
Include as pending below
Review targets
Step 3: Review IP targets to include in your group
Confirm the IP targets to include in your target group. Add more IP targets by repeating steps 1 and 2 on this page. You can also register additional targets after your target group is created.
Targets (0)
Remove all pending
Filter targets
Show only pending
Remove IPv4 address
Health status
IP address
Port
Zone
No IP addresses included yet
Specify IP addresses above and add to list.
0 pending
Cancel
Previous
Create target group

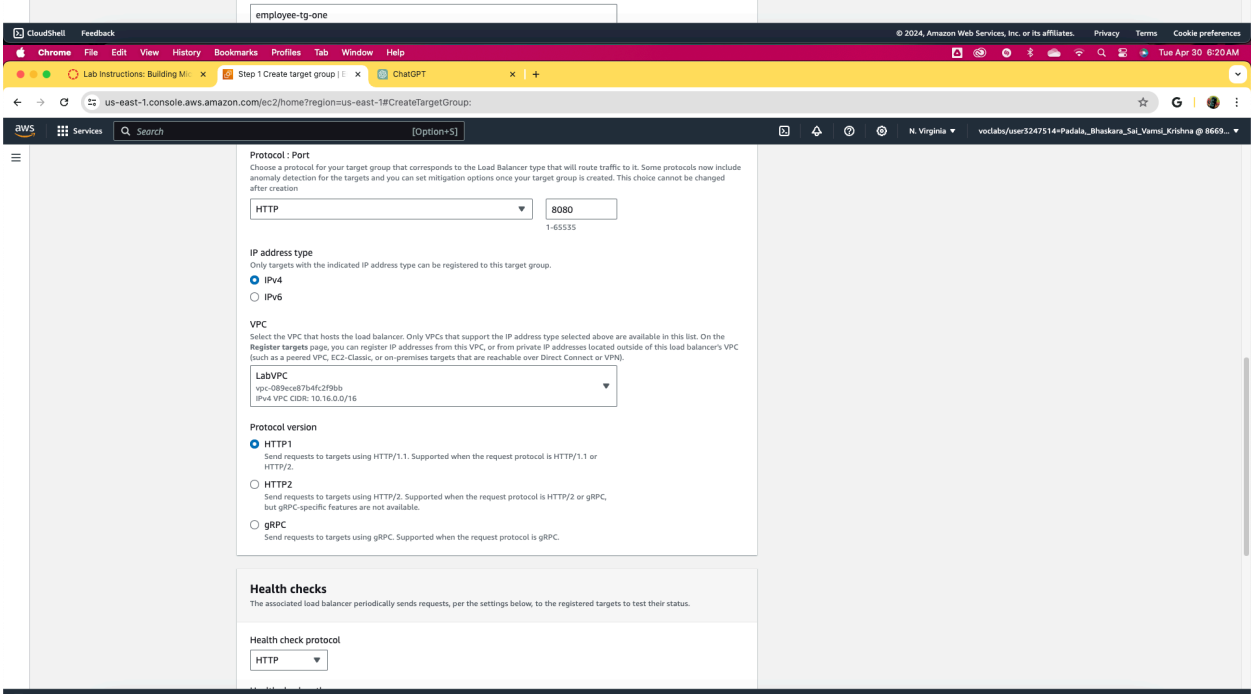
Hence, we successfully created 2nd Target Group as well.

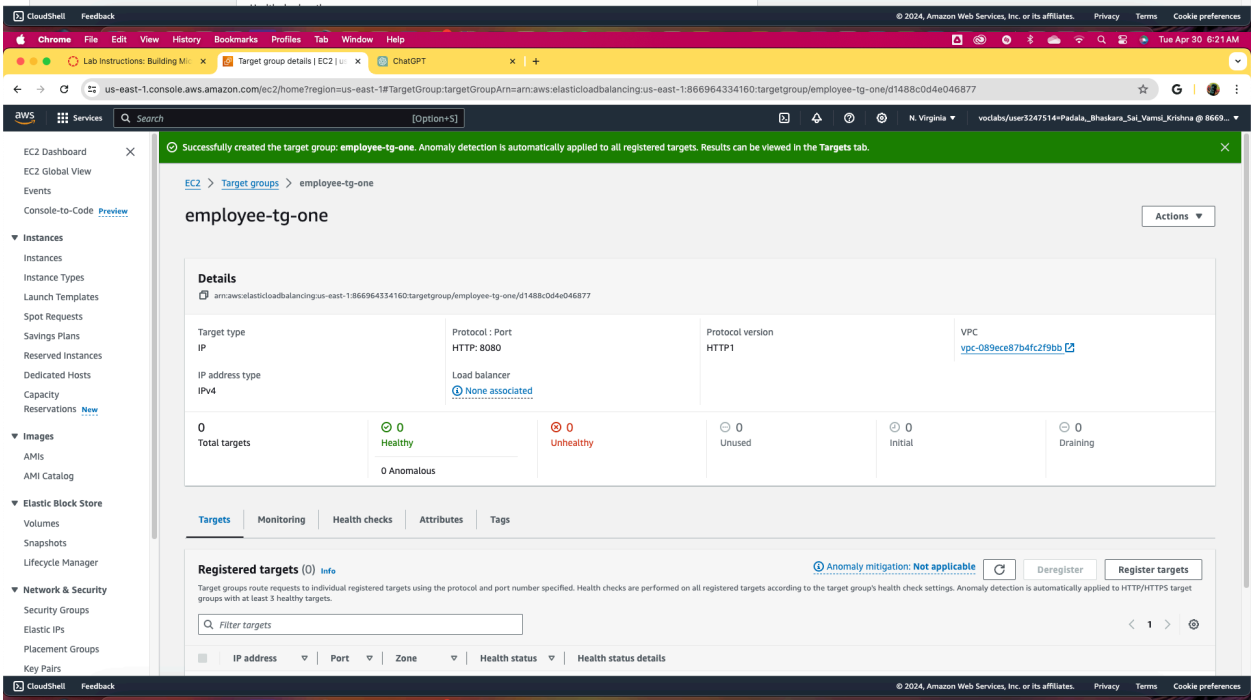


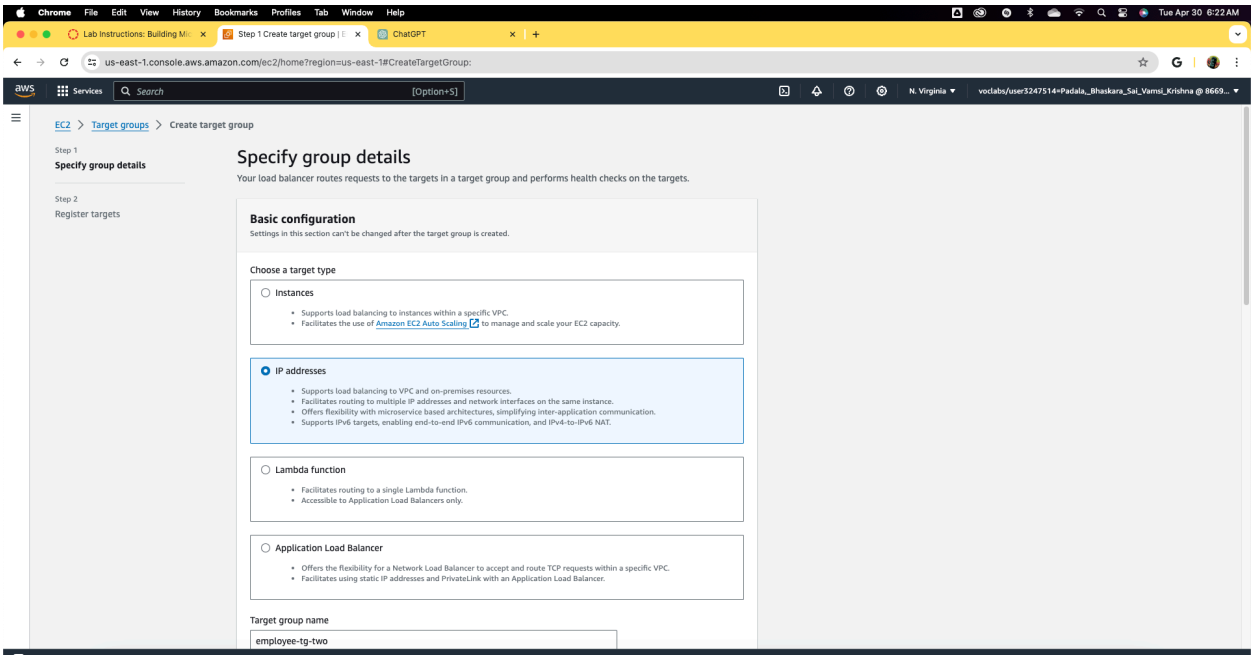


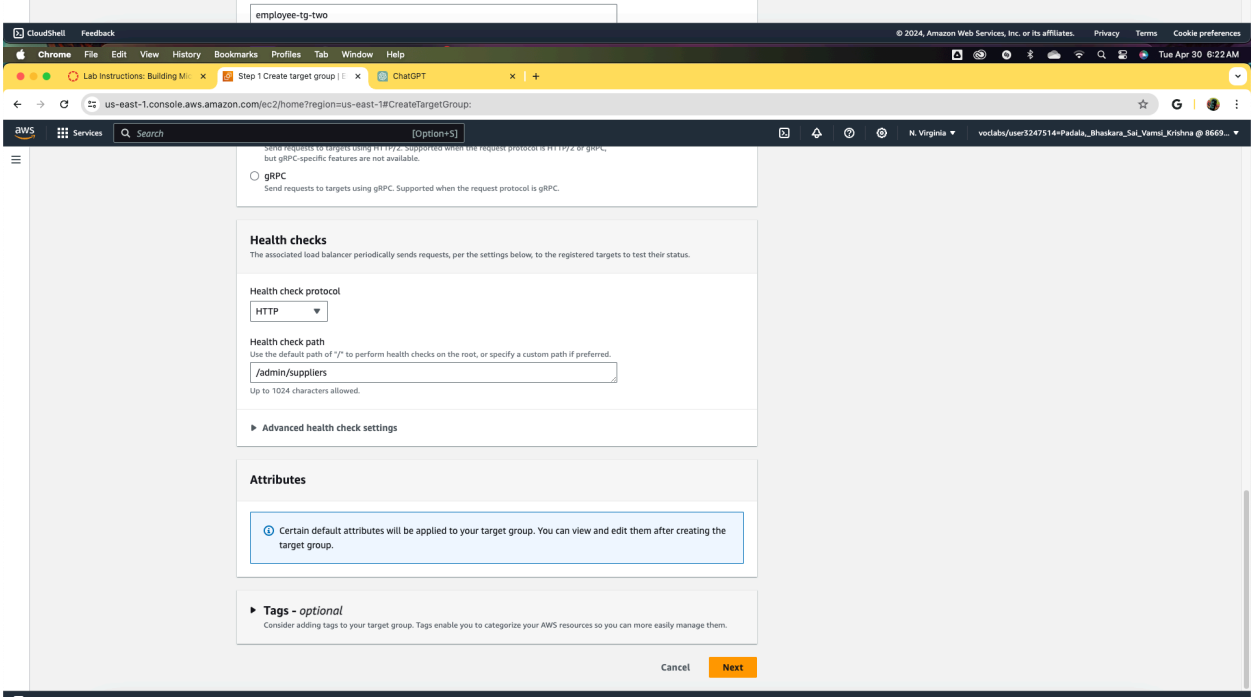
Now we will Create a target group for the employee microservice. We will use the same settings as the other target groups with the following exceptions:

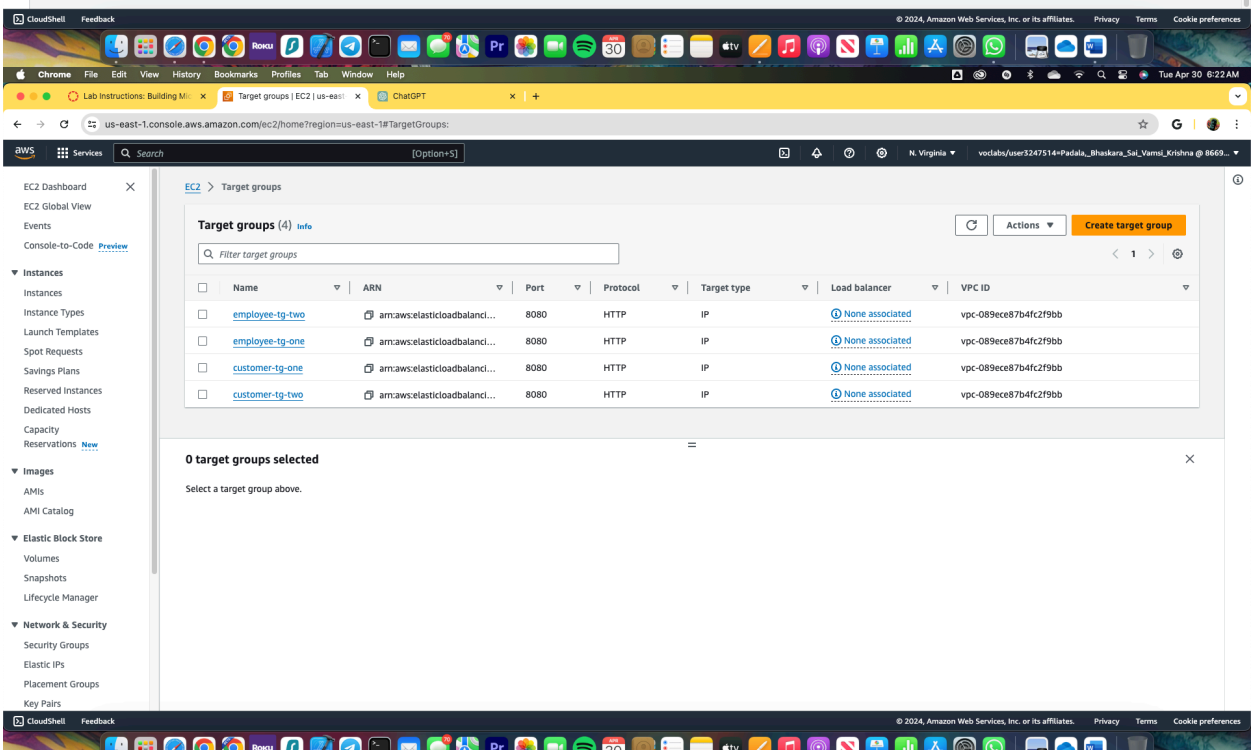

EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Specify group details
Your load balancer routes requests to the targets in a target group and performs health checks on the targets.
Basic configuration
Settings in this section can't be changed after the target group is created.
Choose a target type
Instances
Supports load balancing to instances within a specific VPC.
Facilitates the use of Amazon EC2 Auto Scaling to manage and scale your EC2 capacity.
IP addresses
Supports load balancing to VPC and on-premises resources.
Facilitates routing to multiple IP addresses and network interfaces on the same instance.
Offers flexibility with microservice based architectures, simplifying inter-application communication.
Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
Lambda function
Facilitates routing to a single Lambda function.
Accessible to Application Load Balancers only.
Application Load Balancer
Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.
Target group name
employee-tg-one


Protocol : Port
Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols now include anomaly detection for the targets and you can set mitigation options once your target group is created. This choice cannot be changed after creation
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).
LabVPC
vpc-089ece87b4fc2f9bb
IPv4 VPC CIDR: 10.16.0.0/16
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


Successfully created the target group: employee-tg-one. Anomaly detection is automatically applied to all registered targets. Results can be viewed in the Targets tab.
EC2 > Target groups > employee-tg-one
employee-tg-one
Actions
Details
arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-one/d1488c0d4e046877
Target type
IP
Protocol : Port
HTTP: 8080
Protocol version
HTTP1
VPC
vpc-089ece87b4fc2f9bb
IP address type
IPv4
Load balancer
None associated
0
Total targets
0
Healthy
0 Anomalous
0
Unhealthy
0
Unused
0
Initial
0
Draining
Targets
Monitoring
Health checks
Attributes
Tags
Registered targets (0) Info
Anomaly mitigation: Not applicable
Deregister
Register targets
Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.
Filter targets
IP address
Port
Zone
Health status
Health status details
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs

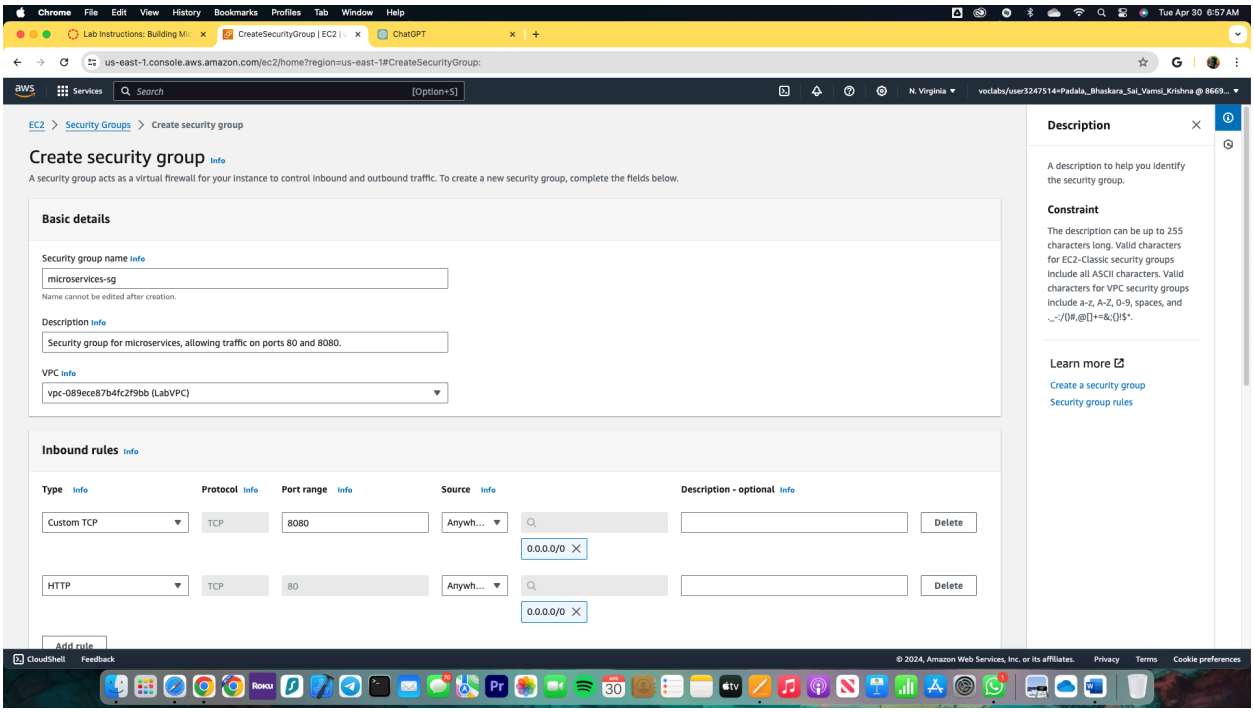

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:22 AM
Lab Instructions: Building Mic
Step 1 Create target group | E
ChatGPT
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:
aws
Services
Search
[Option+S]
N. Virginia
voclabs/user3247514=Padala_Bhaskara_Sai_Vamsi_Krishna @ 8669...
EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Specify group details
Your load balancer routes requests to the targets in a target group and performs health checks on the targets.
Basic configuration
Settings in this section can't be changed after the target group is created.
Choose a target type
Instances
Supports load balancing to instances within a specific VPC.
Facilitates the use of Amazon EC2 Auto Scaling to manage and scale your EC2 capacity.
IP addresses
Supports load balancing to VPC and on-premises resources.
Facilitates routing to multiple IP addresses and network interfaces on the same instance.
Offers flexibility with microservice based architectures, simplifying inter-application communication.
Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
Lambda function
Facilitates routing to a single Lambda function.
Accessible to Application Load Balancers only.
Application Load Balancer
Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.
Target group name
employee-tg-two
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

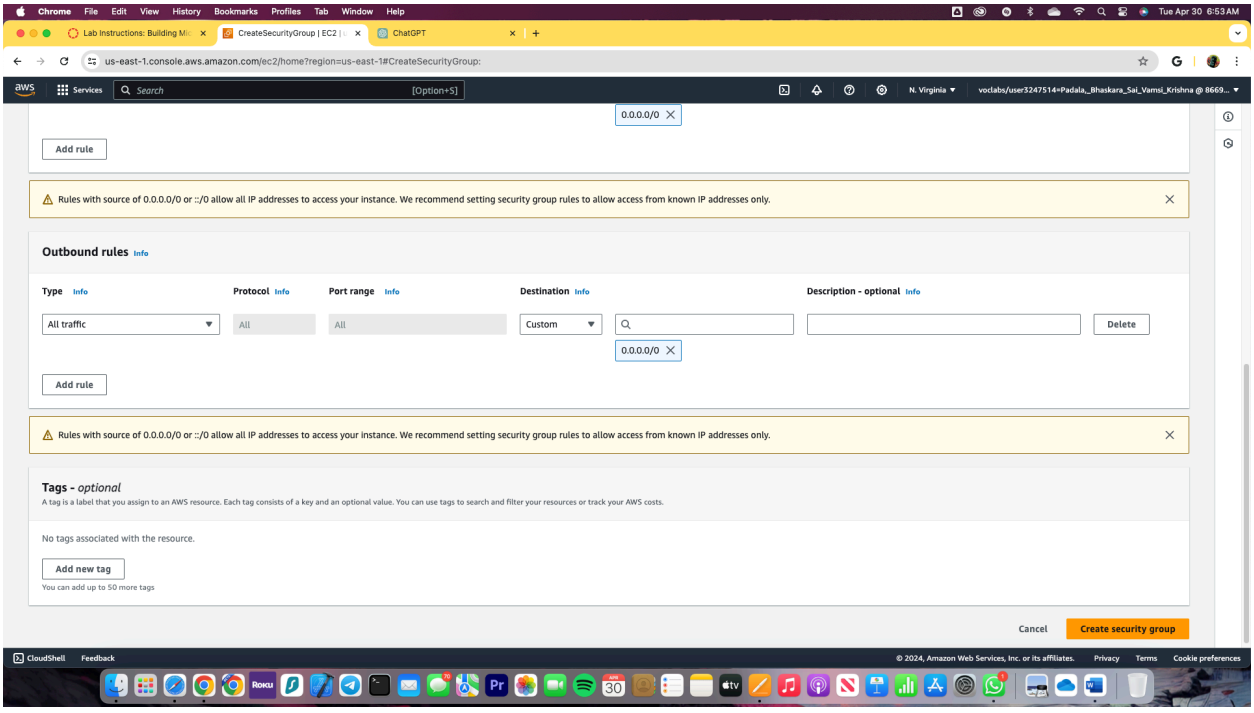

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:22 AM
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:
gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.
Health checks
The associated load balancer periodically 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.
/admin/suppliers
Up to 1024 characters allowed.
Advanced health check settings
Attributes
Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.
Tags - optional
Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.
Cancel
Next
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

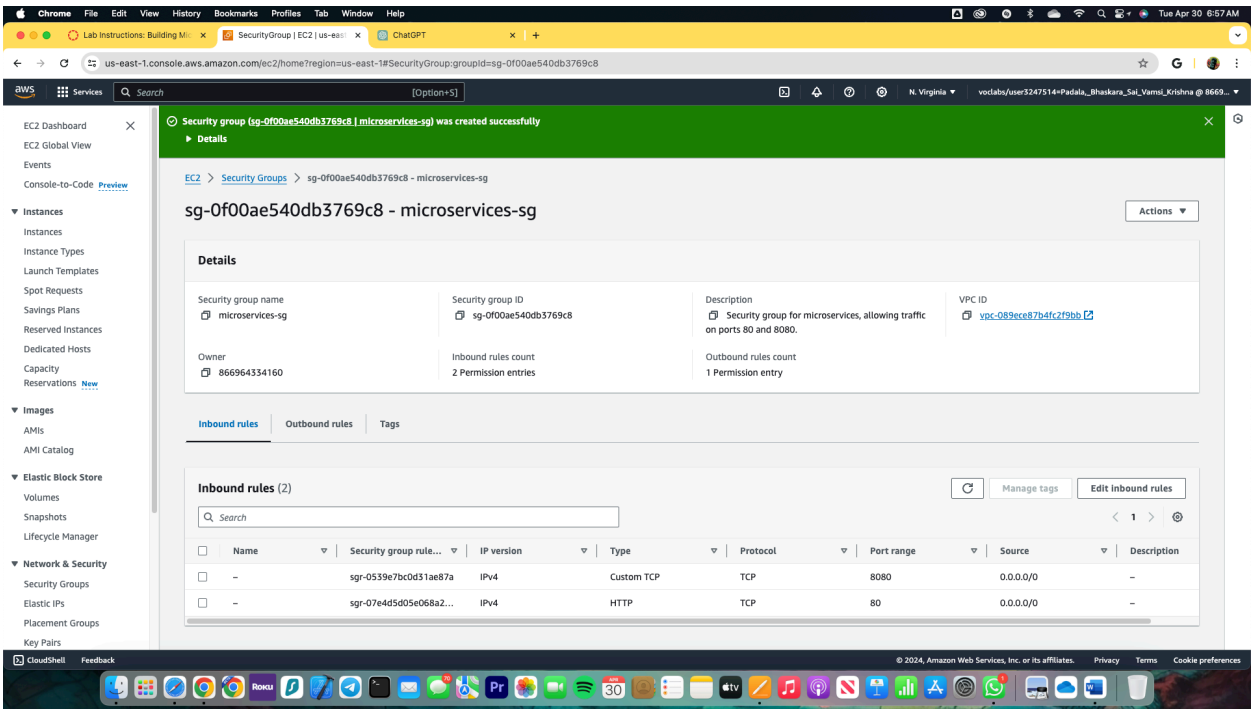

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 6:22 AM
Target groups | EC2 | us-east
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#TargetGroups:
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
EC2 > Target groups
Target groups (4) Info
Actions
Create target group
Filter target groups
Name
ARN
Port
Protocol
Target type
Load balancer
VPC ID
employee-tg-two
arn:aws:elasticloadbalanci...
8080
HTTP
IP
None associated
vpc-089ece87b4fc2f9bb
employee-tg-one
arn:aws:elasticloadbalanci...
8080
HTTP
IP
None associated
vpc-089ece87b4fc2f9bb
customer-tg-one
arn:aws:elasticloadbalanci...
8080
HTTP
IP
None associated
vpc-089ece87b4fc2f9bb
customer-tg-two
arn:aws:elasticloadbalanci...
8080
HTTP
IP
None associated
vpc-089ece87b4fc2f9bb
0 target groups selected
Select a target group above.
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Task 6.2: Create a security group and an Application Load Balancer, and configure rules to route traffic

In this task, we will create an Application Load Balancer. We will also define two listeners for the load balancer: one on port 80 and another on port 8080. For each listener, we will then define path-based routing rules so that traffic is routed to the correct target group depending on the URL that a user attempts to load.



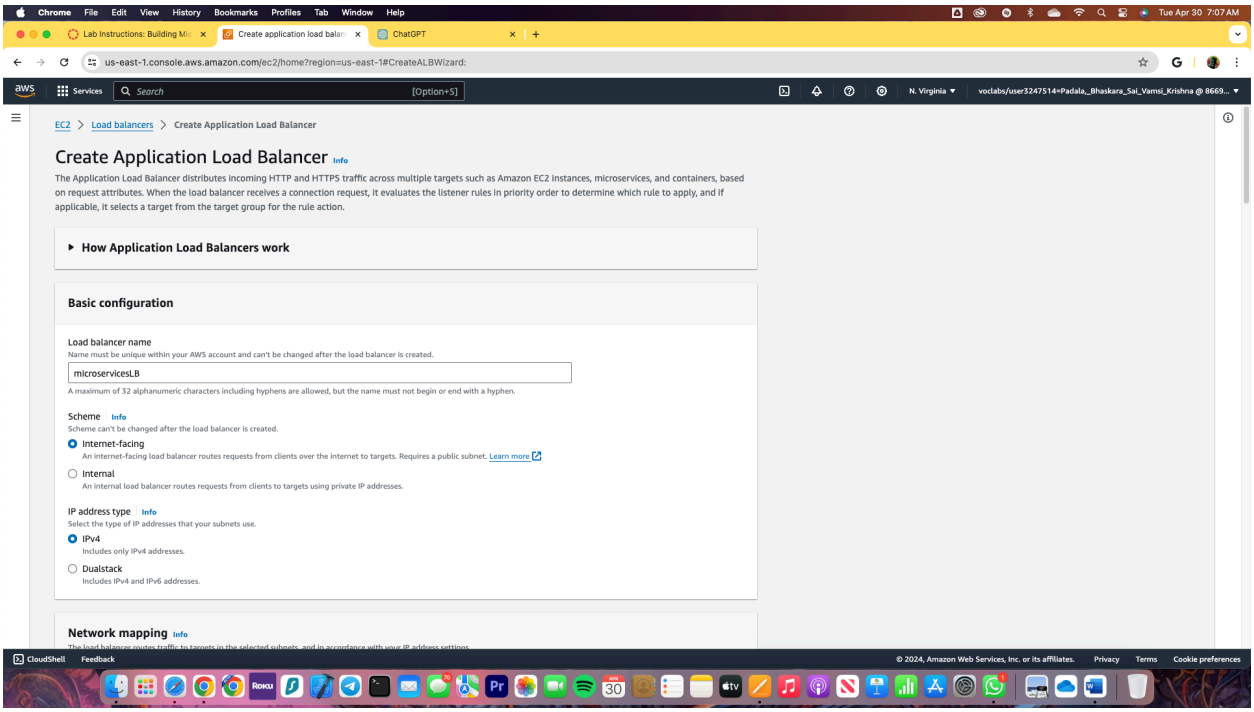


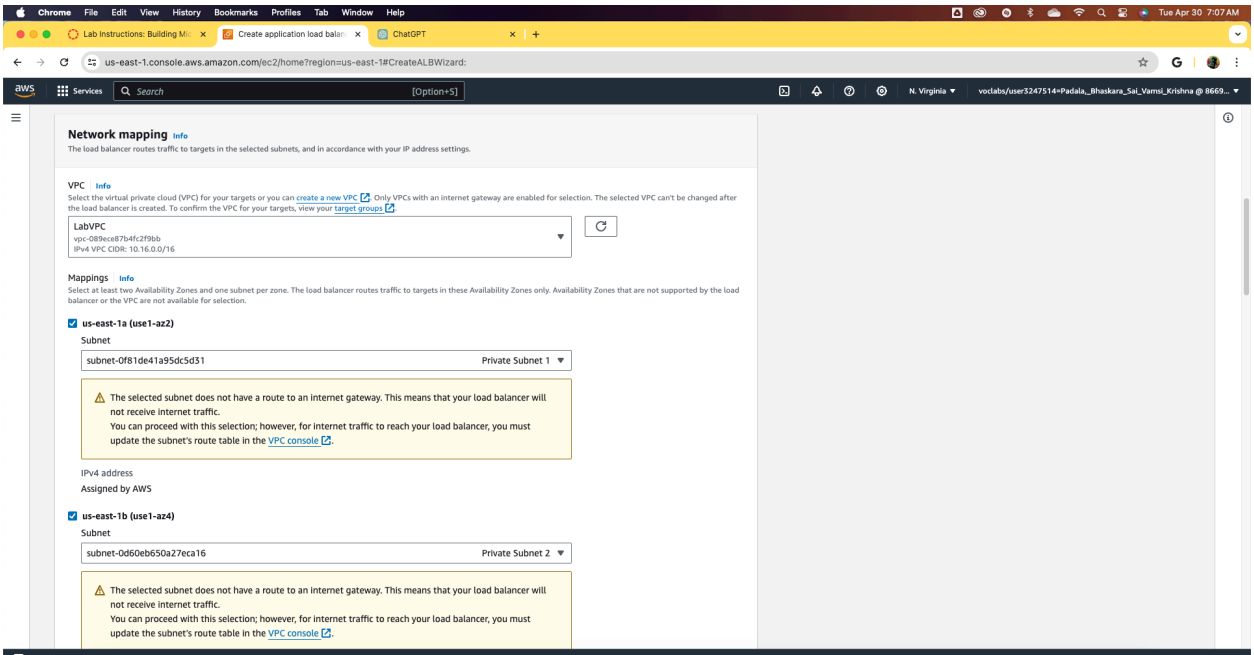


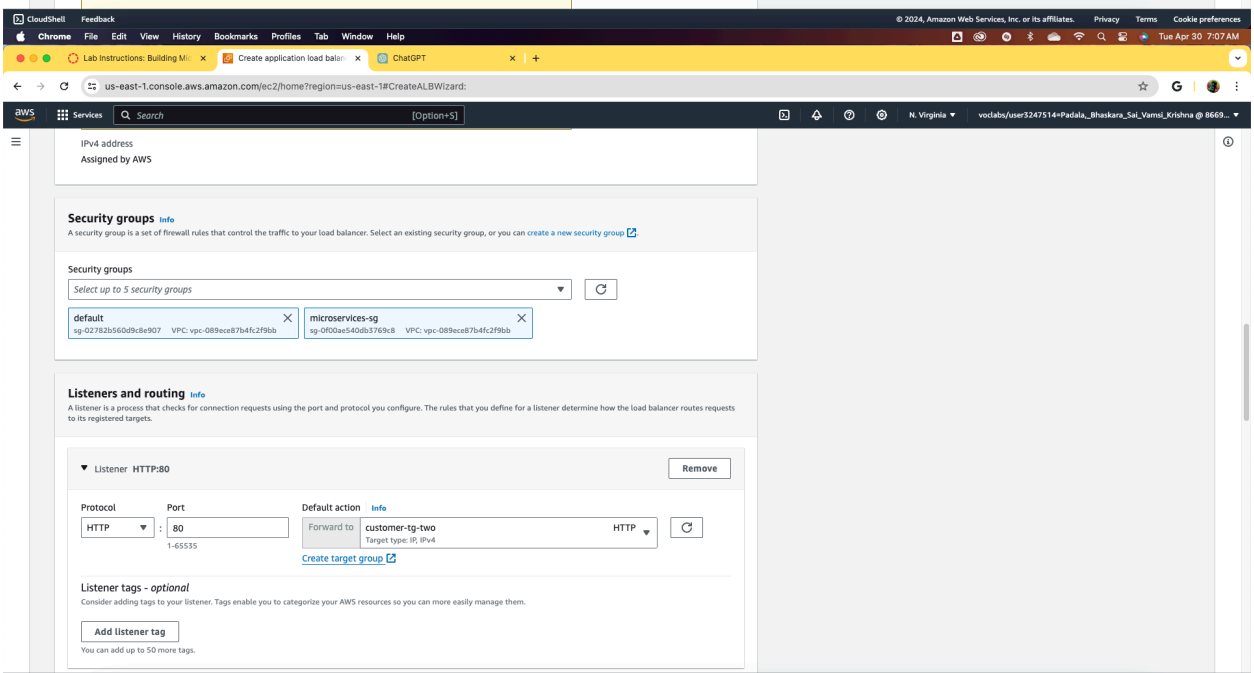
In the Amazon EC2 console, we will create an Application Load Balancer named microservicesLB. We will make it internet facing for IPv4 addresses.

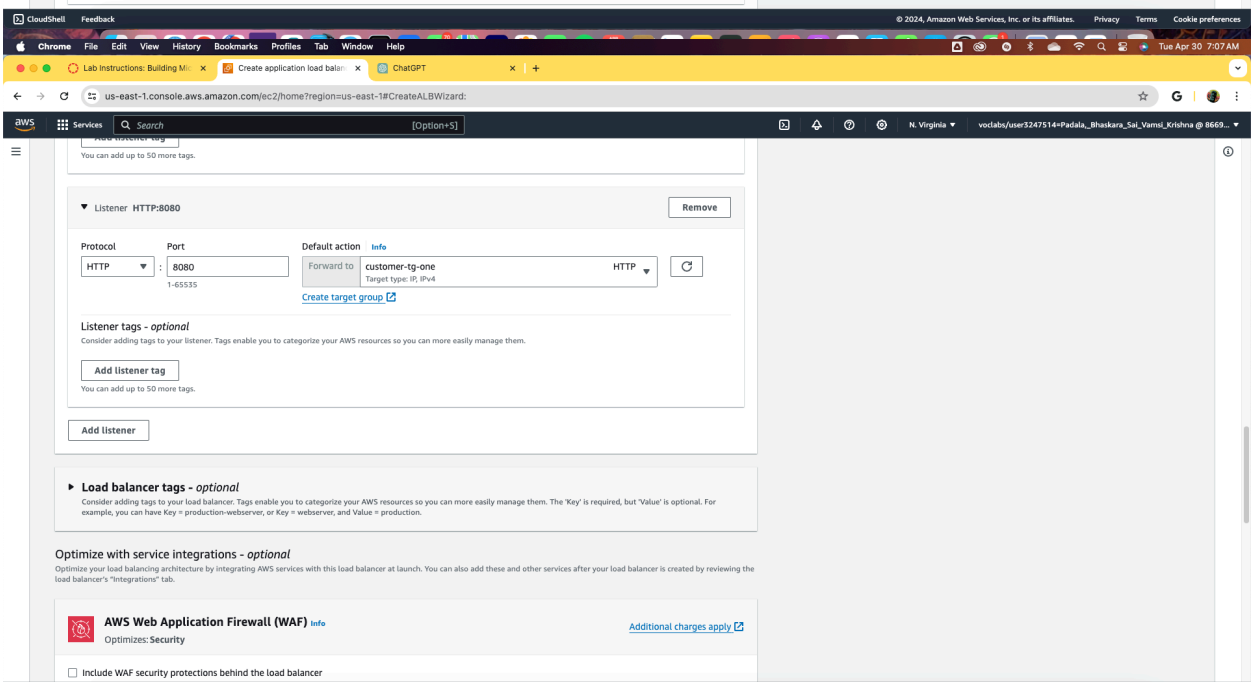
We will use LabVPC, Public Subnet1, Public Subnet2, and the microservices-sg security group.

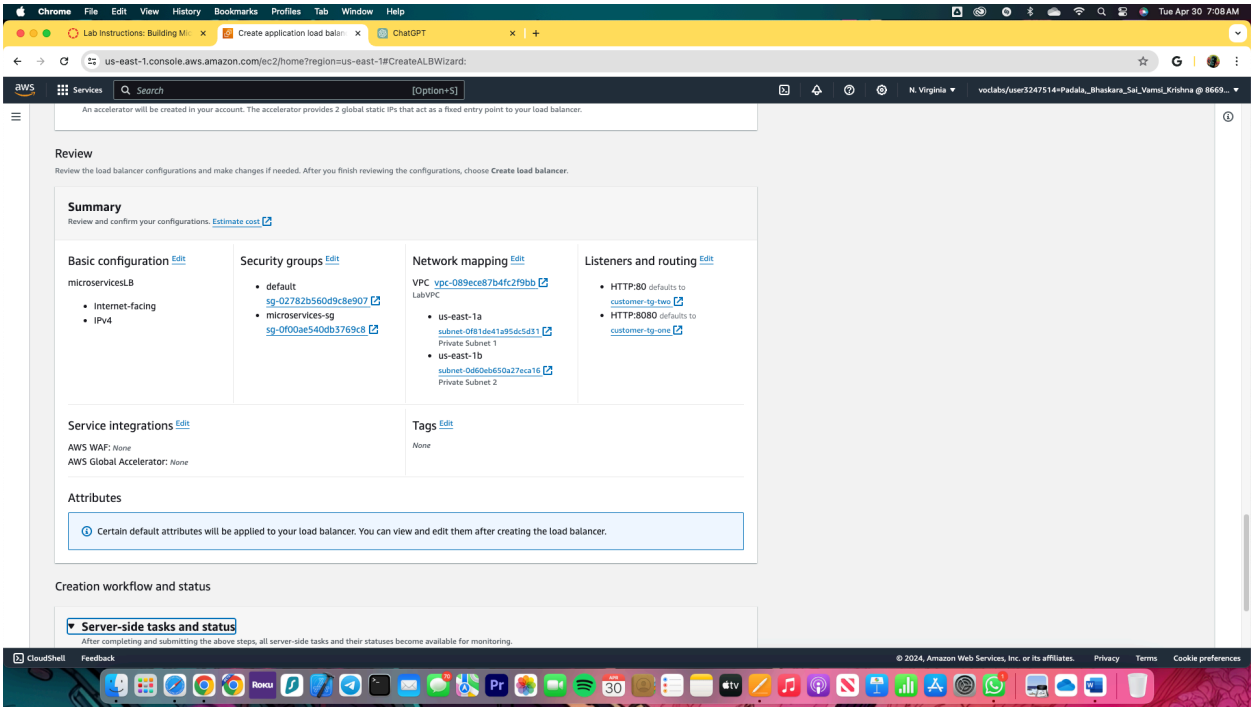
Configure two listeners on it. The first should listen on HTTP:80 and forward traffic to customer-tg-two by default. The second should listen on HTTP:8080 and forward traffic to customer-tg-one by default.

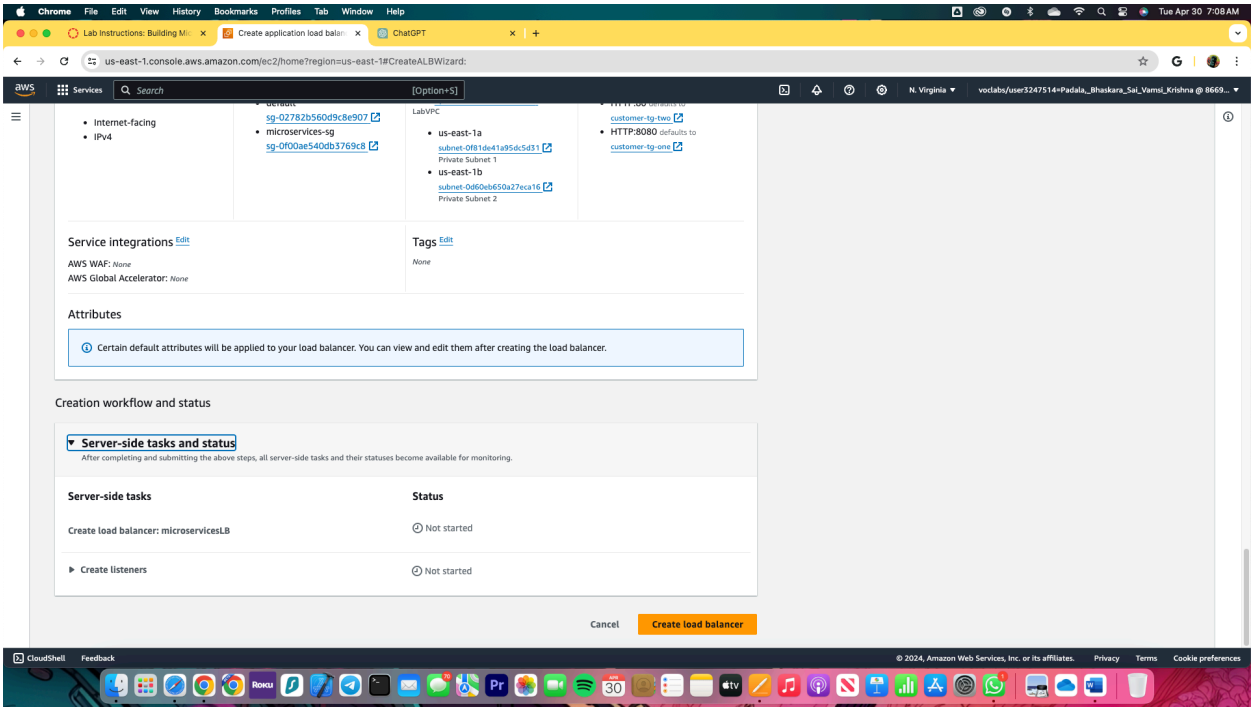



Network mapping Info
The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.
VPC Info
Select the virtual private cloud (VPC) for your targets or you can create a new VPC. Only VPCs with an internet gateway are enabled for selection. The selected VPC can't be changed after the load balancer is created. To confirm the VPC for your targets, view your target groups
LabVPC
vpc-089ece87b4fc2f9bb
IPv4 VPC CIDR: 10.16.0.0/16
Mappings Info
Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.
us-east-1a (use1-az2)
Subnet
subnet-0f81de41a95dc5d31
Private Subnet 1
The selected subnet does not have a route to an internet gateway. This means that your load balancer will not receive internet traffic.
You can proceed with this selection; however, for internet traffic to reach your load balancer, you must update the subnet's route table in the VPC console.
IPv4 address
Assigned by AWS
us-east-1b (use1-az4)
Subnet
subnet-0d60eb650a27eca16
Private Subnet 2
The selected subnet does not have a route to an internet gateway. This means that your load balancer will not receive internet traffic.
You can proceed with this selection; however, for internet traffic to reach your load balancer, you must update the subnet's route table in the VPC console.


IPv4 address
Assigned by AWS
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
default
sg-02782b560d9c8e907 VPC: vpc-089ece87b4fc2f9bb
microservices-sg
sg-0f00ae540db3769c8 VPC: vpc-089ece87b4fc2f9bb
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 load balancer routes requests to its registered targets.
Listener HTTP:80
Remove
Protocol
HTTP
Port
80
1-65535
Default action Info
Forward to
customer-tg-two
Target type: IP, IPv4
HTTP
Create target group
Listener tags - optional
Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.
Add listener tag
You can add up to 50 more tags.


You can add up to 50 more tags.
Listener HTTP:8080
Remove
Protocol
HTTP
Port
8080
1-65535
Default action Info
Forward to
customer-tg-one
Target type: IP, IPv4
HTTP
Create target group
Listener tags - optional
Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.
Add listener tag
You can add up to 50 more tags.
Add listener
Load balancer tags - optional
Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them. The 'Key' is required, but 'Value' is optional. For example, you can have Key = production-webserver, or Key = webserver, and Value = production.
Optimize with service integrations - optional
Optimize your load balancing architecture by integrating AWS services with this load balancer at launch. You can also add these and other services after your load balancer is created by reviewing the load balancer's "Integrations" tab.
AWS Web Application Firewall (WAF) Info
Optimizes: Security
Additional charges apply
Include WAF security protections behind the load balancer

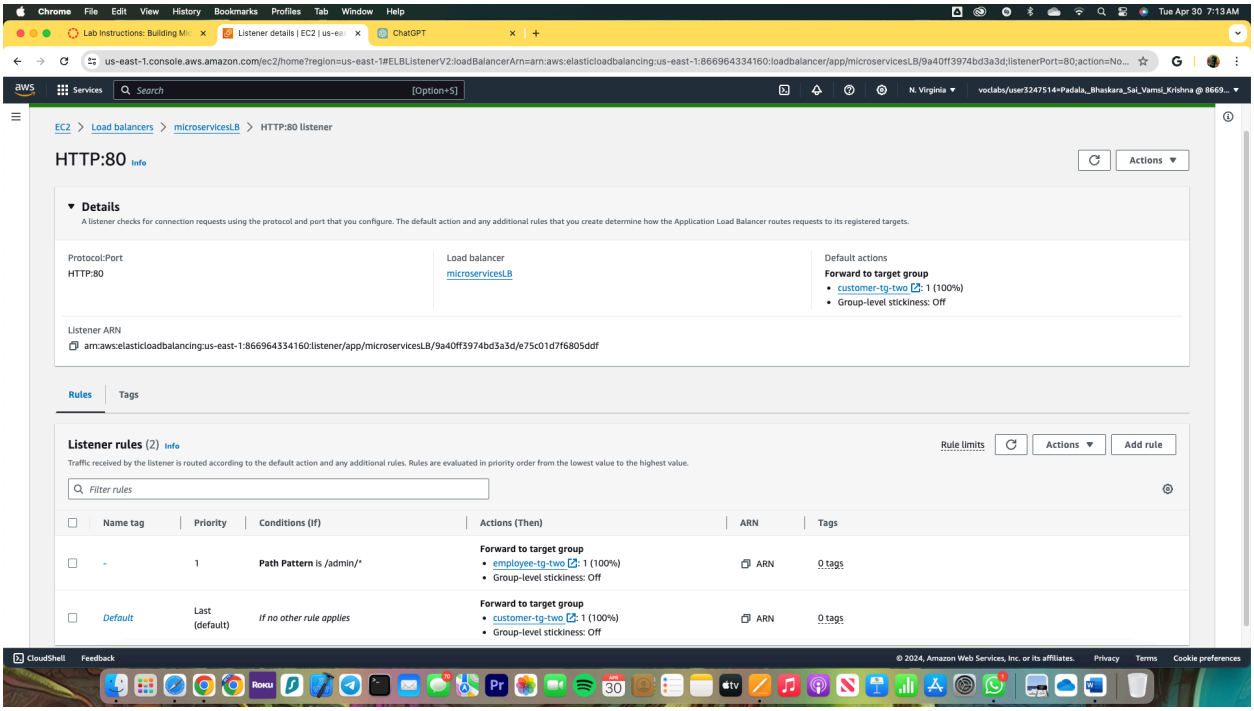




Add a second rule for the HTTP:80 listener. Define the following logic for this new rule:

IF Path is /admin/*

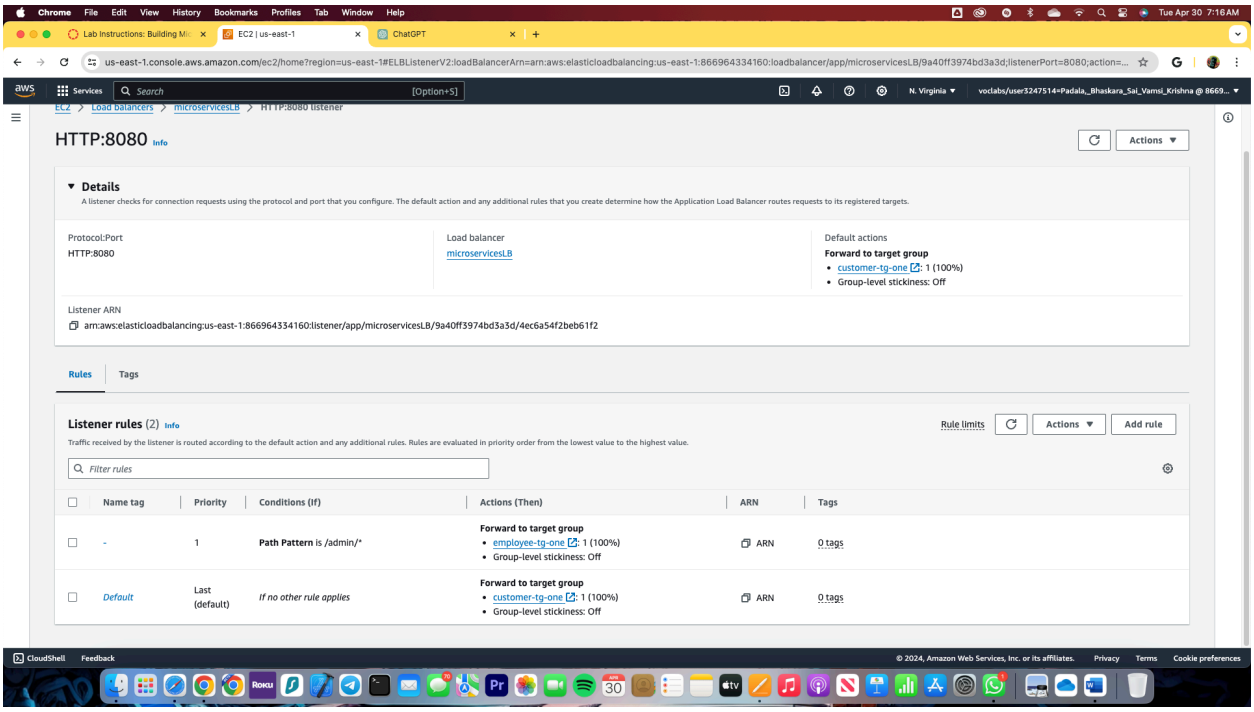
THEN Forward to... the employee-tg-two target group.



Add a second rule for the HTTP:8080 listener. Define the following logic for this new rule:

IF Path is /admin/*

THEN Forward to the employee-tg-one target group.

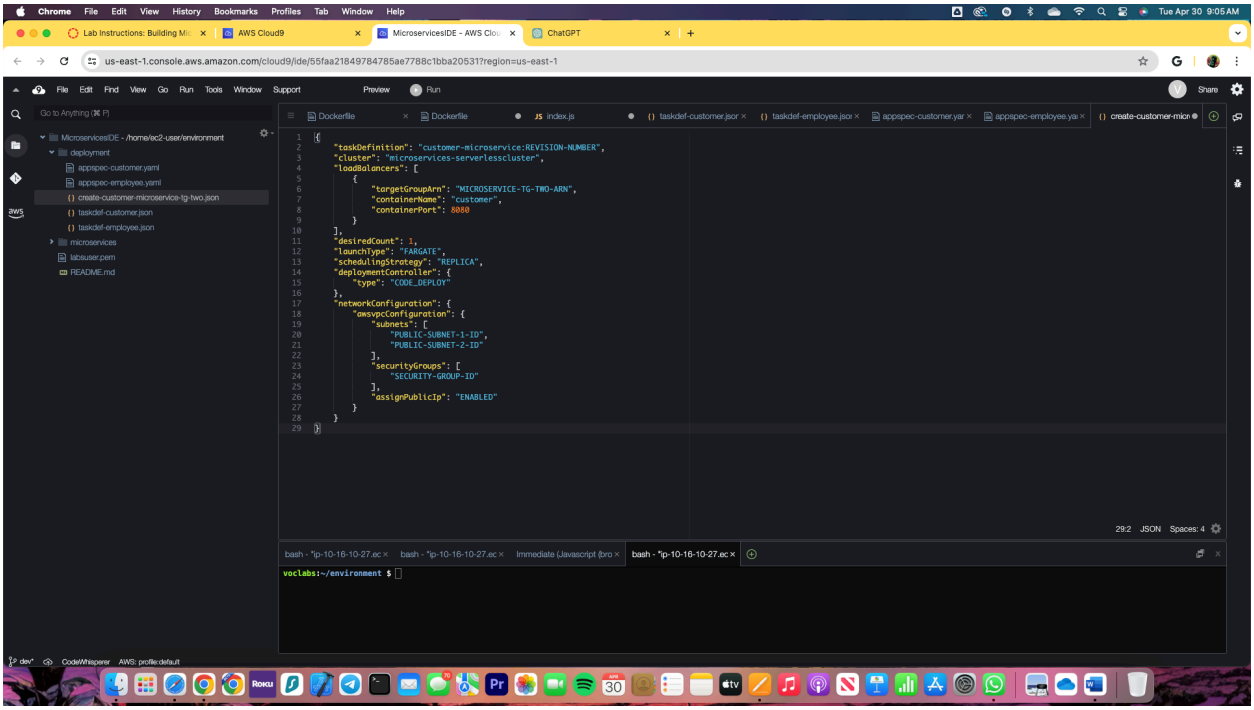


Phase 7: Creating two Amazon ECS services

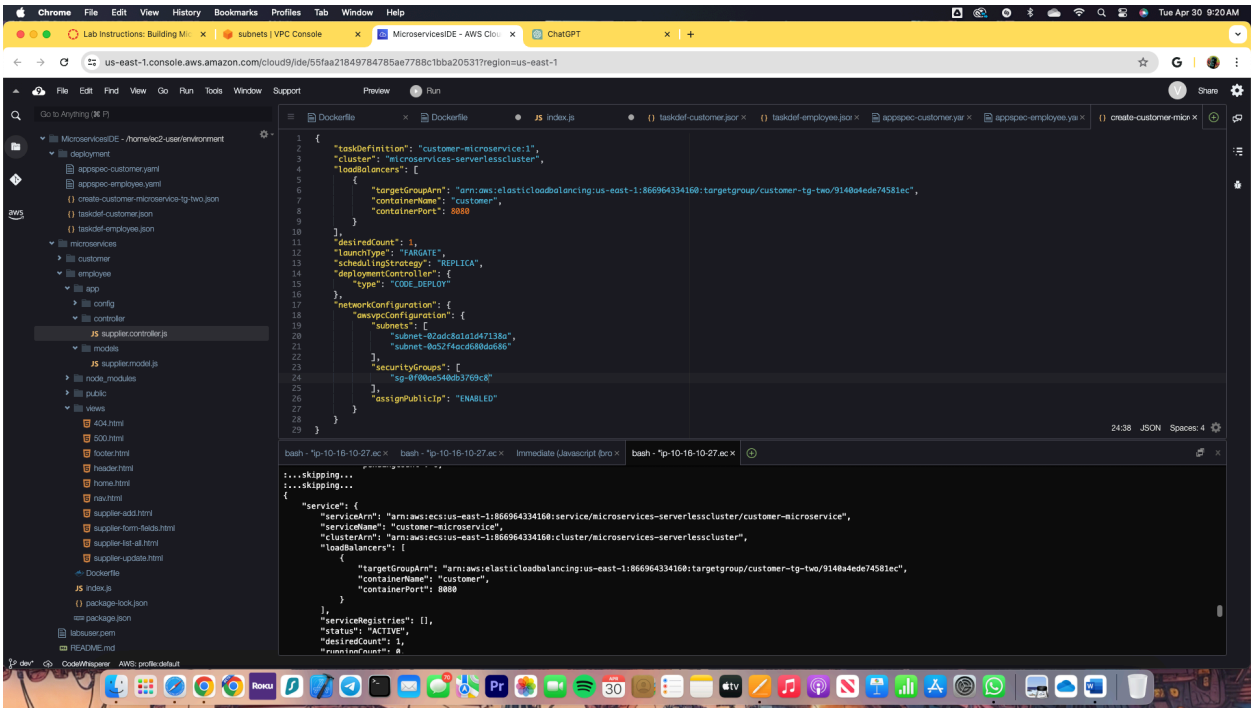
In this phase, we will create a service in Amazon ECS for each microservice. Although we could deploy both microservices to a single ECS service, for this project, it will be easier to manage the microservices independently if each is deployed to its own ECS service.

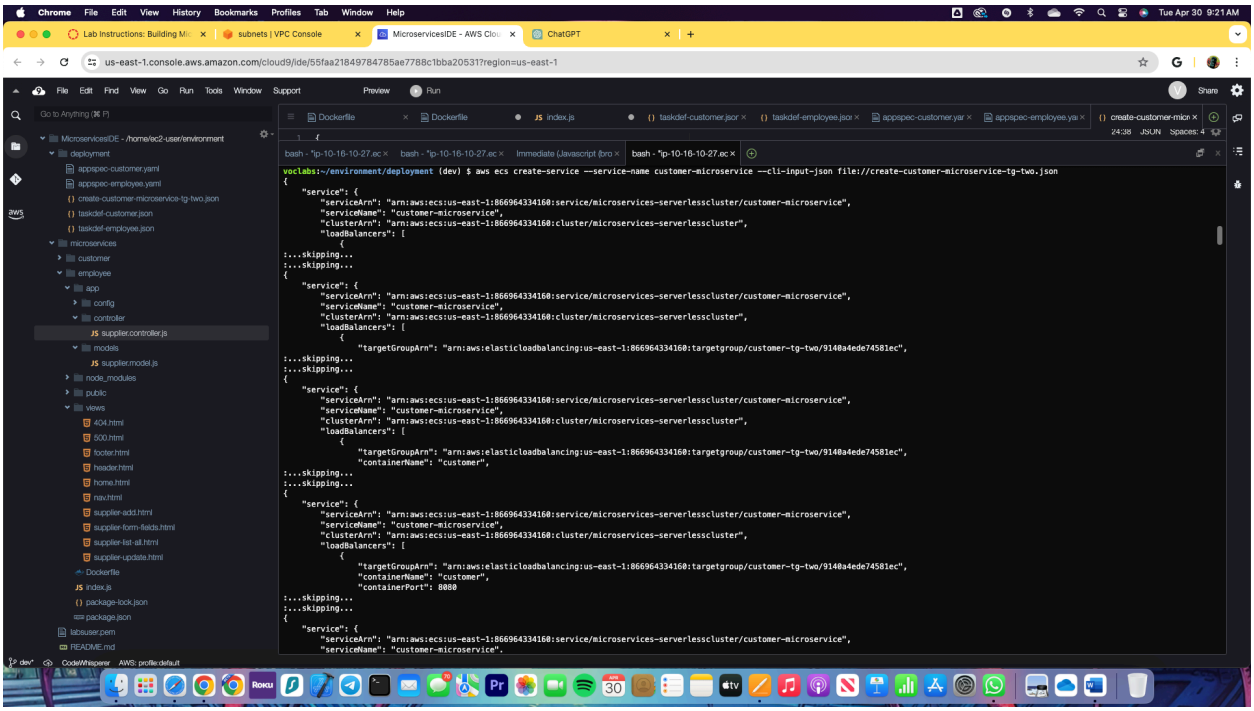
Task 7.1: Create the ECS service for the customer microservice

In AWS Cloud9, create a new file named create-customer-microservice-tg-two.json in the deployment directory. Paste the following JSON code into the file:



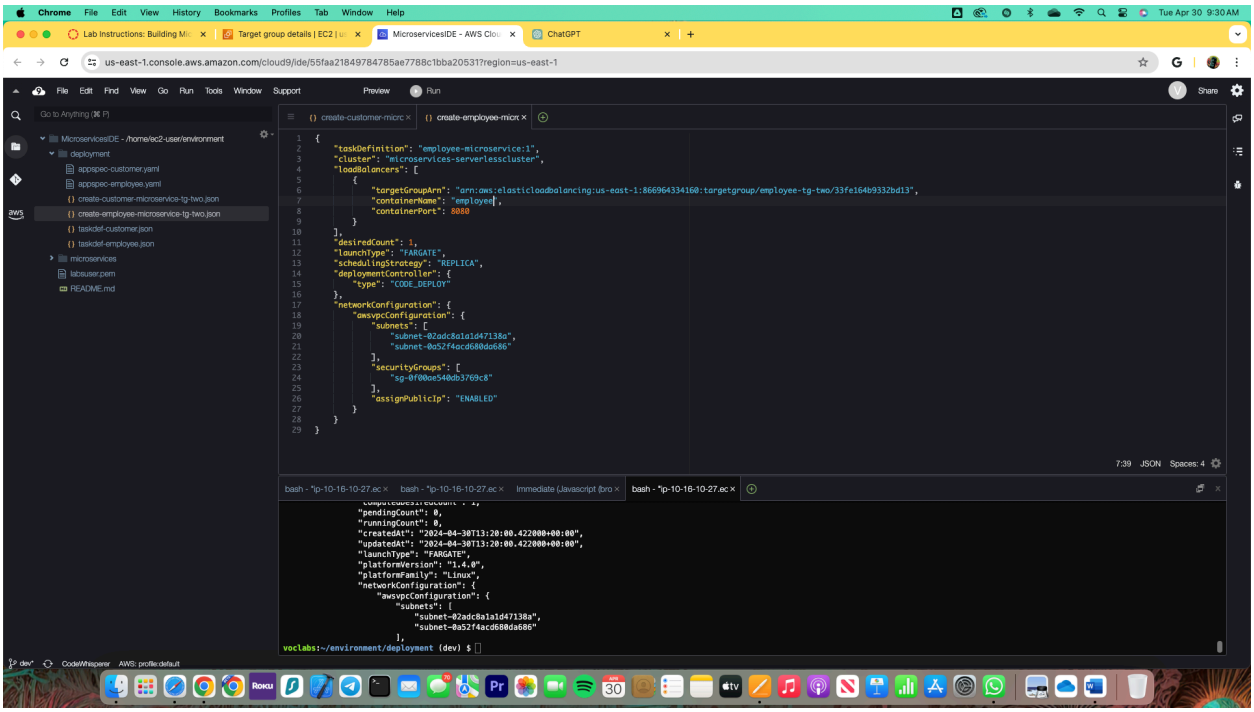
Edit the create-customer-microservice-tg-two.json file:

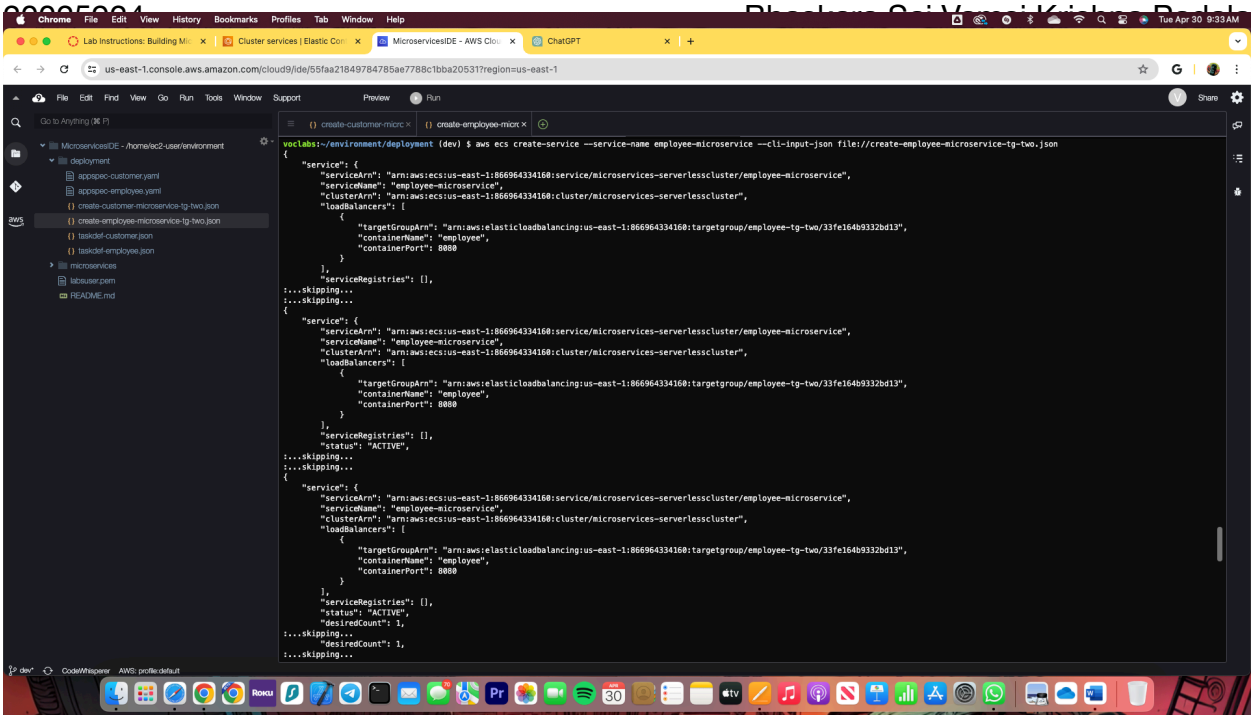


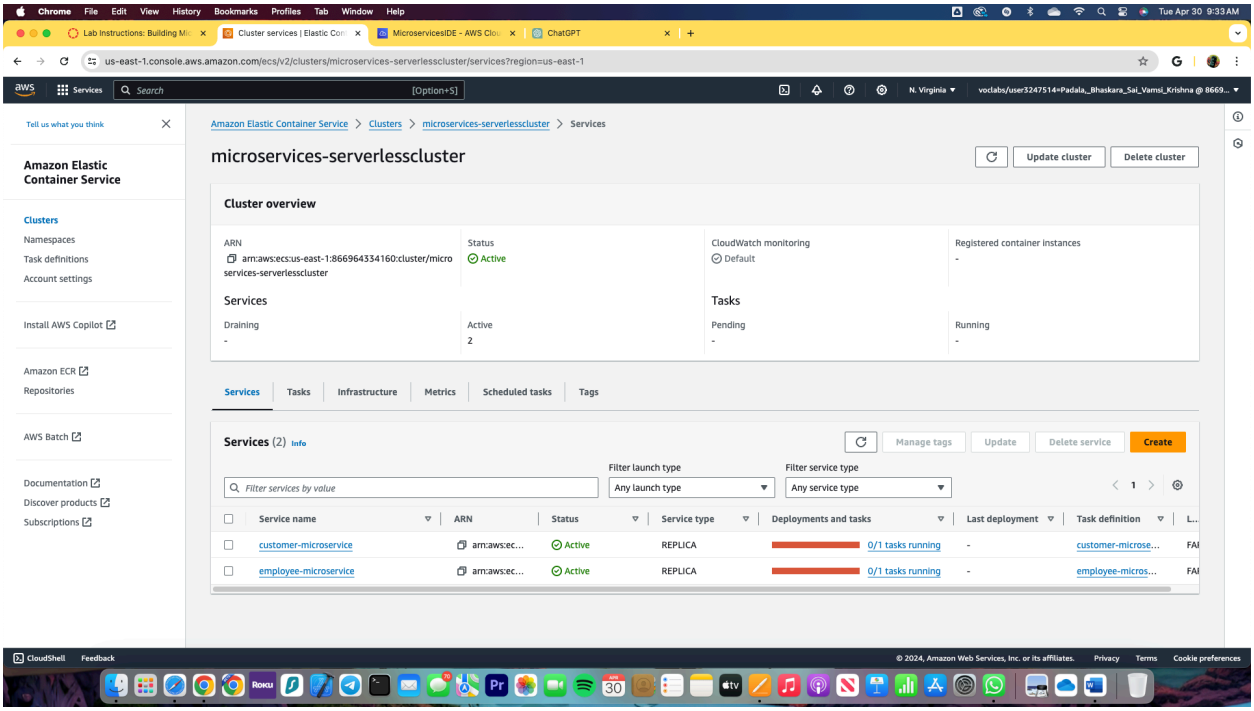


Task 7.2: Create the Amazon ECS service for the employee microservice

Firstly, we will Create an Amazon ECS service for the employee microservice.
Then we need to Open the File: Locate the create-employee-microservice-tg-two.json file in your file system. Open the file using a text editor or an IDE. Change Service Name and Revision Number: Navigate to line 2 in the file. Update customer-microservice to employee-microservice. Also, update the revision number to reflect the changes made. Enter the ARN of the Target Group: Find the ARN of the employee-tg-two target group.
Navigate to line 6 in the file. Replace the existing placeholder with the actual ARN of the employee-tg-two target group.
Update Resource Tag: Navigate to line 7 in the file. Change customer to employee to ensure consistency with the service name.







Save the Changes: After making the necessary modifications, save the file. When you check the services in the Amazon ECS console, it's normal to observe a status like "0/1 Task running" at this point because you haven't initiated task sets yet. This simply means that you haven't started any tasks for these services yet, and it's part of the expected setup process.

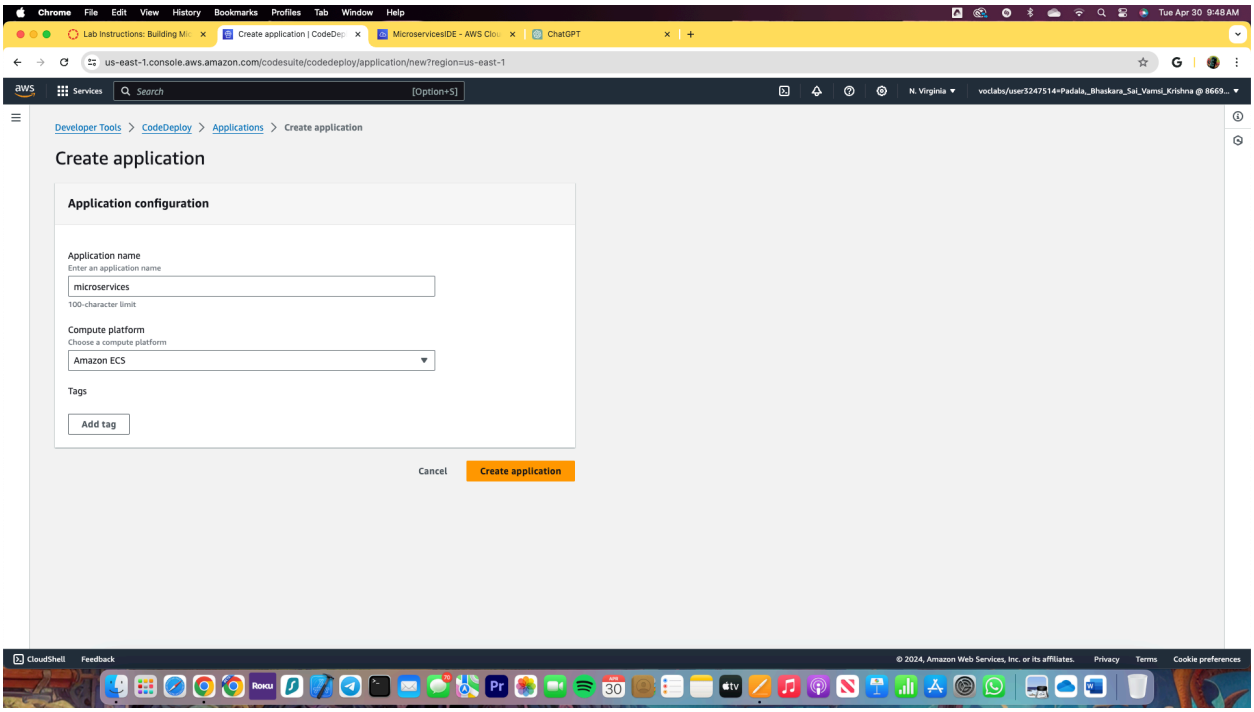
In the ECS context, it's akin to laying out the groundwork before actual tasks start running. This phase involves setting up task definitions, creating task sets, and eventually launching tasks. Once tasks are initiated, you can monitor their status and manage them through the ECS console.

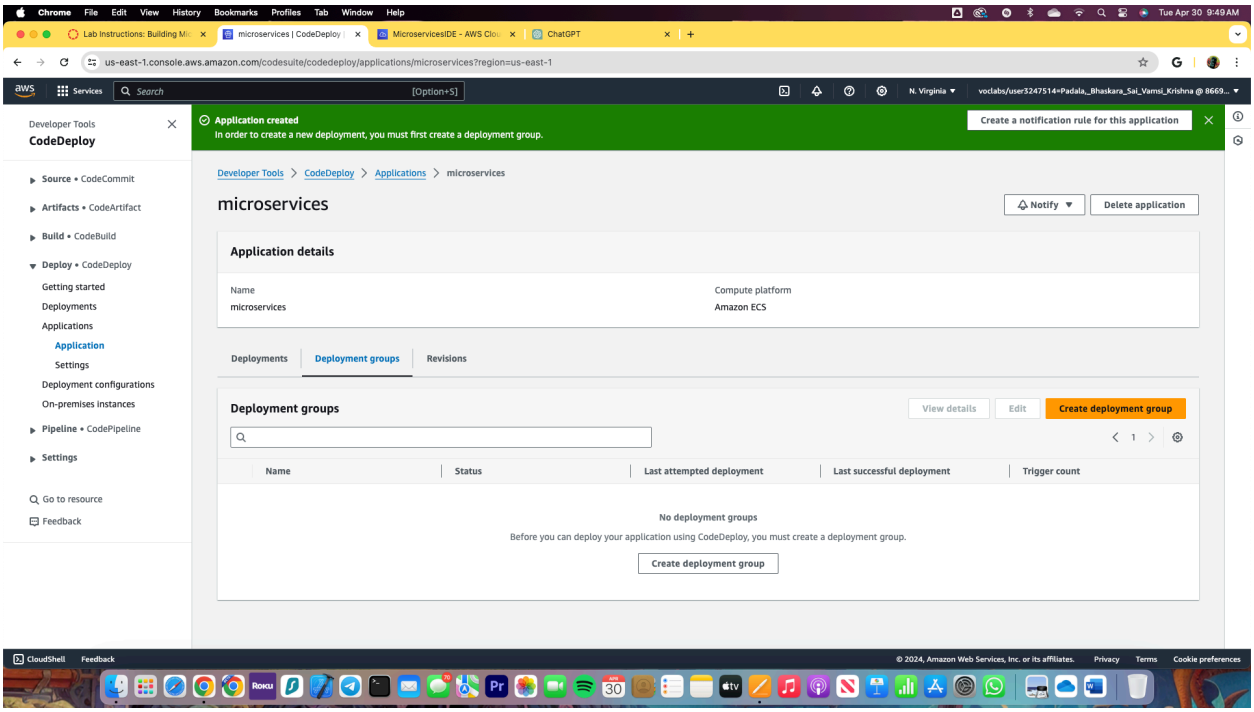
Phase 8: Configuring CodeDeploy and CodePipeline

Now that you've laid out the Application Load Balancer, target groups, and the Amazon ECS services that form the infrastructure for deploying your microservices, the subsequent phase involves setting up the CI/CD pipeline to facilitate application deployment.

The diagram below illustrates the pivotal role of the pipeline within the solution you're constructing.

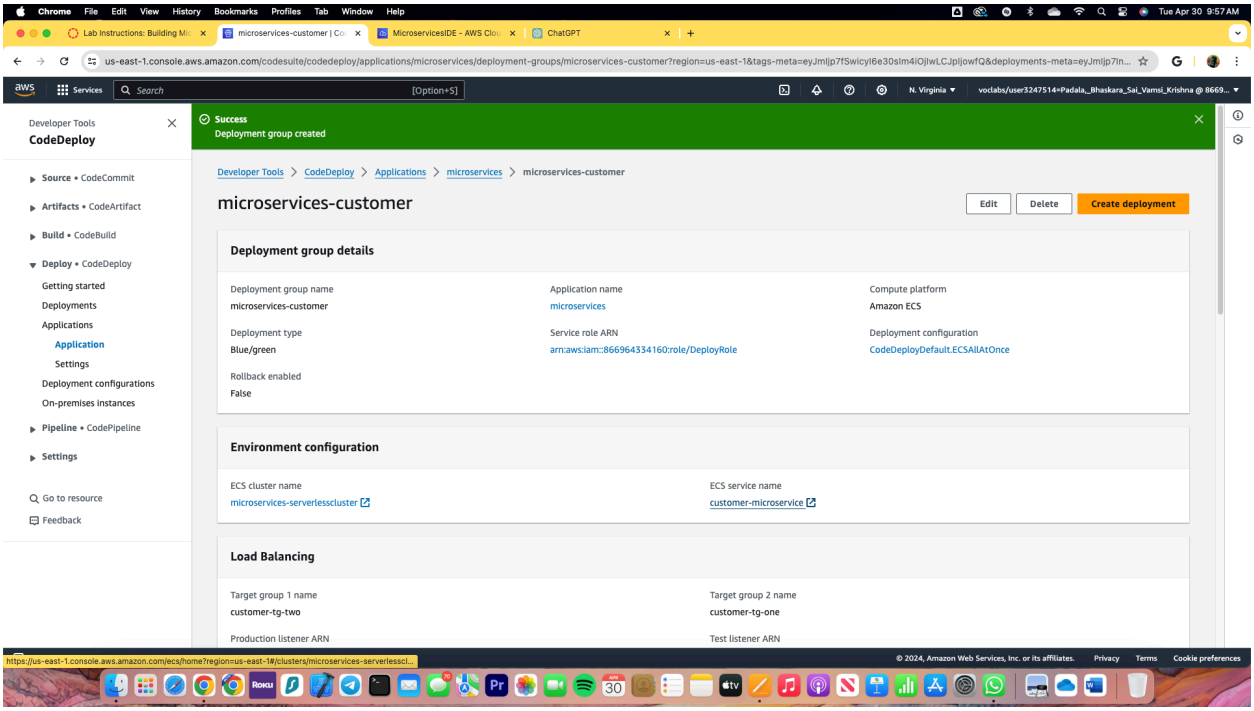
Task 8.1: Create a CodeDeploy application and deployment groups



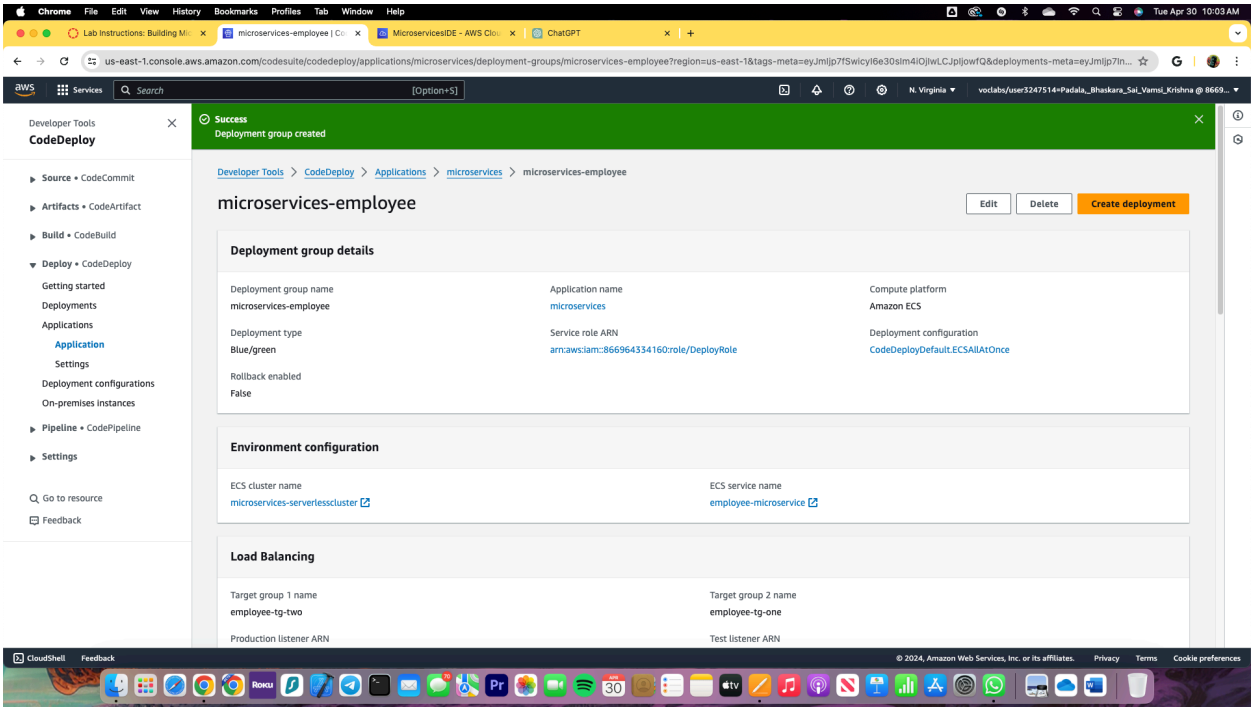


To create a CodeDeploy deployment group for the customer microservice, follow these steps:

Go to the microservices application detail page in the AWS Management Console.
Choose the "Deployment groups" tab.
Click on the "Create deployment group" button.
Provide a name for the deployment group, such as "customer-microservice-deployment-group".



In the "Service role" dropdown, select the IAM service role that you created for CodeDeploy. Under "Deployment type", choose the appropriate deployment type based on your requirements (e.g., in-place or blue/green). For "Environment configuration", select "Amazon ECS" as the compute platform. Select the appropriate ECS cluster and service for the

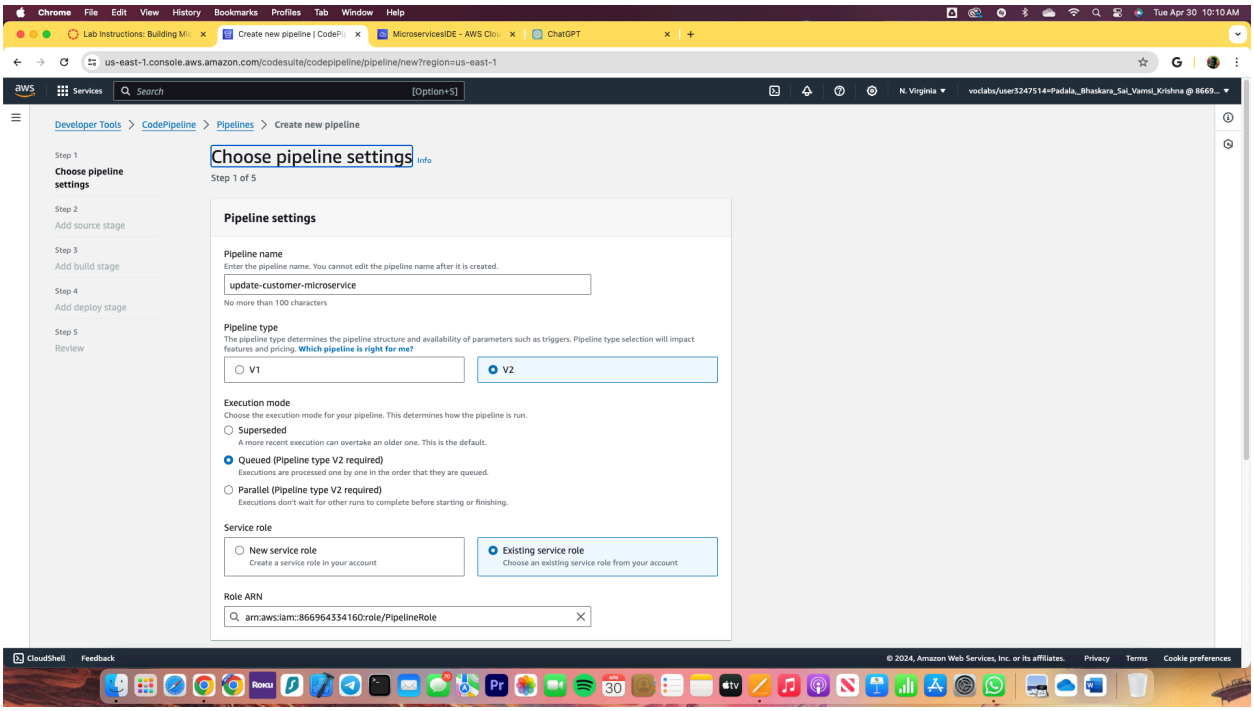


customer microservice. Configure the load balancer settings as needed, specifying the Application Load Balancer and target group associated with the customer microservice.

Task 8.2: Create a pipeline for the customer microservice

In this task, the objective is to create a pipeline to update the customer microservice. Initially, the pipeline will be configured with CodeCommit as the source and CodeDeploy as the deployment service. Later, the pipeline will be edited to incorporate Amazon ECR as a second source.

For an Amazon ECS blue/green deployment, which will be specified in this task, a new set of containers is provisioned. CodeDeploy then installs the latest version of the application on these containers. Subsequently, load balancer traffic is rerouted from the existing set of containers (which host the previous version of the application) to the new set of containers (which host the latest version). Following the rerouting of traffic to the new containers, the existing containers can be terminated. This blue/green deployment approach allows for testing of the new application version before directing production traffic to it.

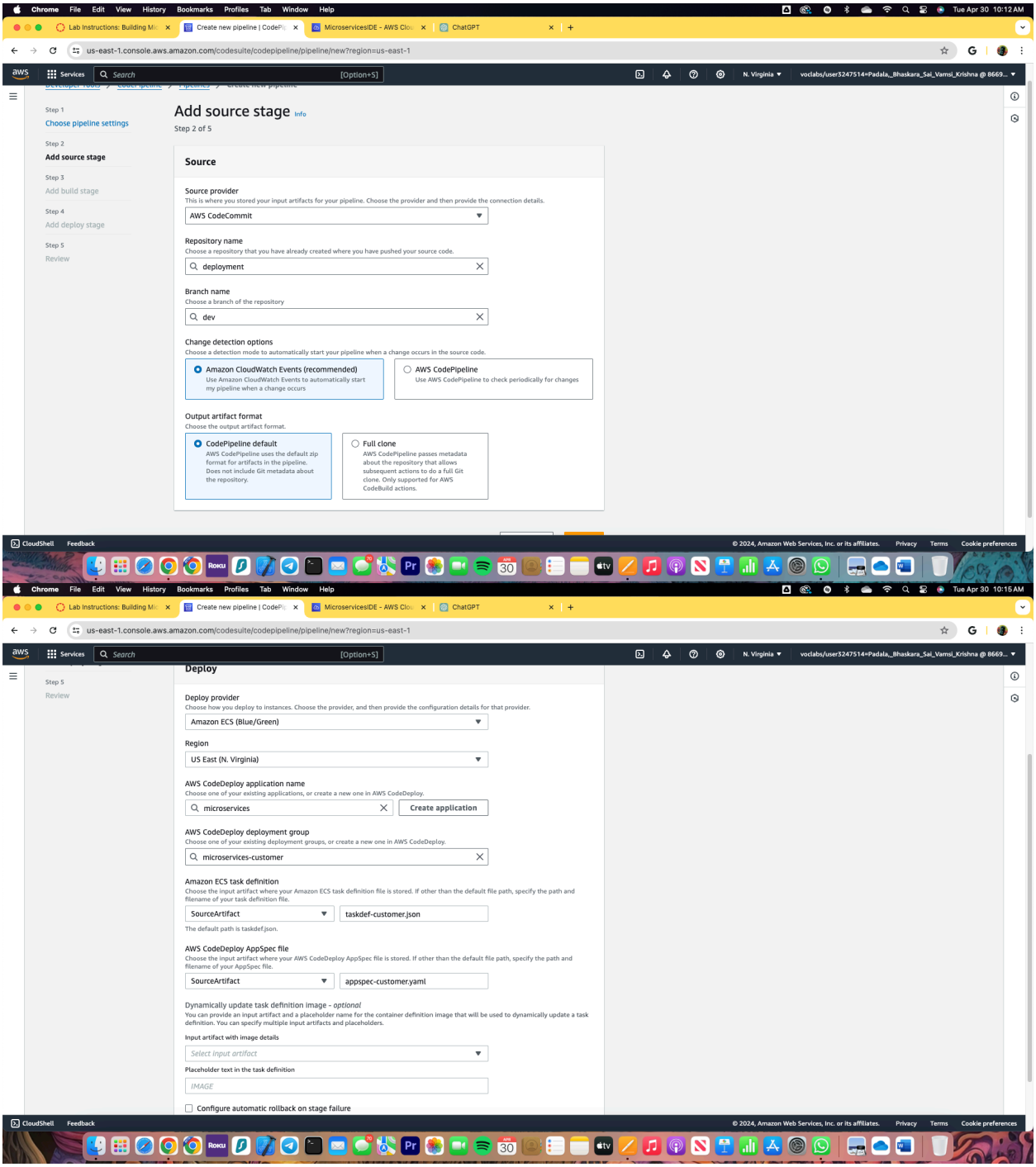


Pipeline name: Enter update-customer-microservice

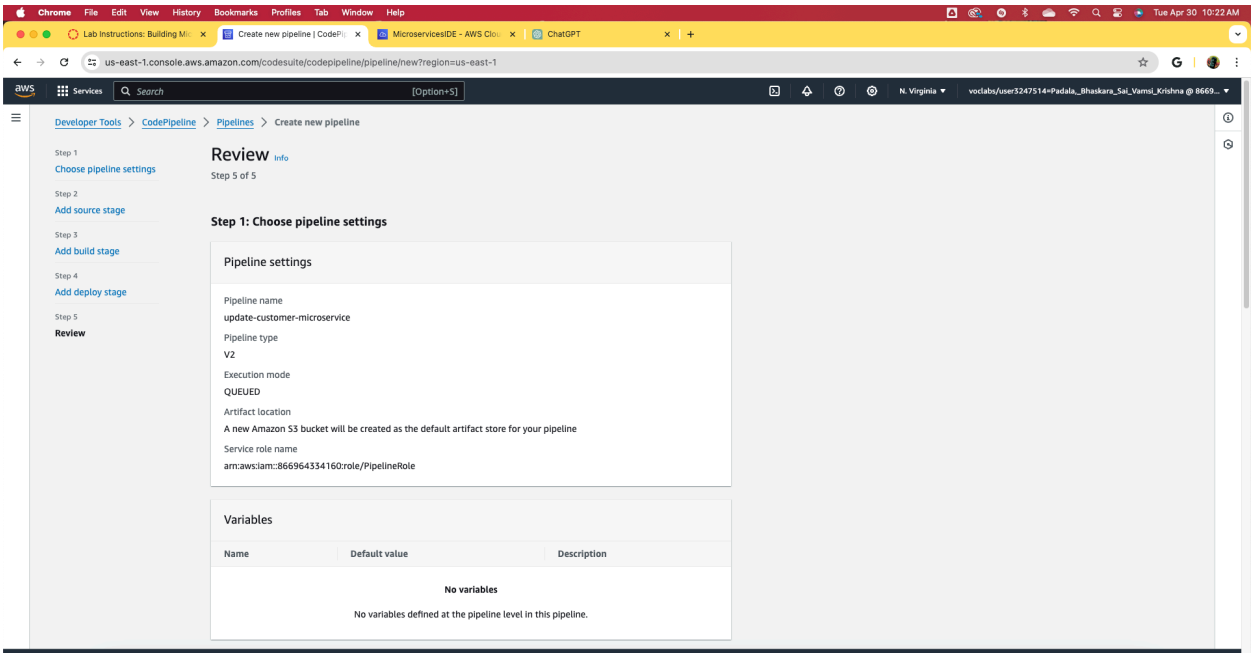
Service role: Choose the ARN for PipelineRole.

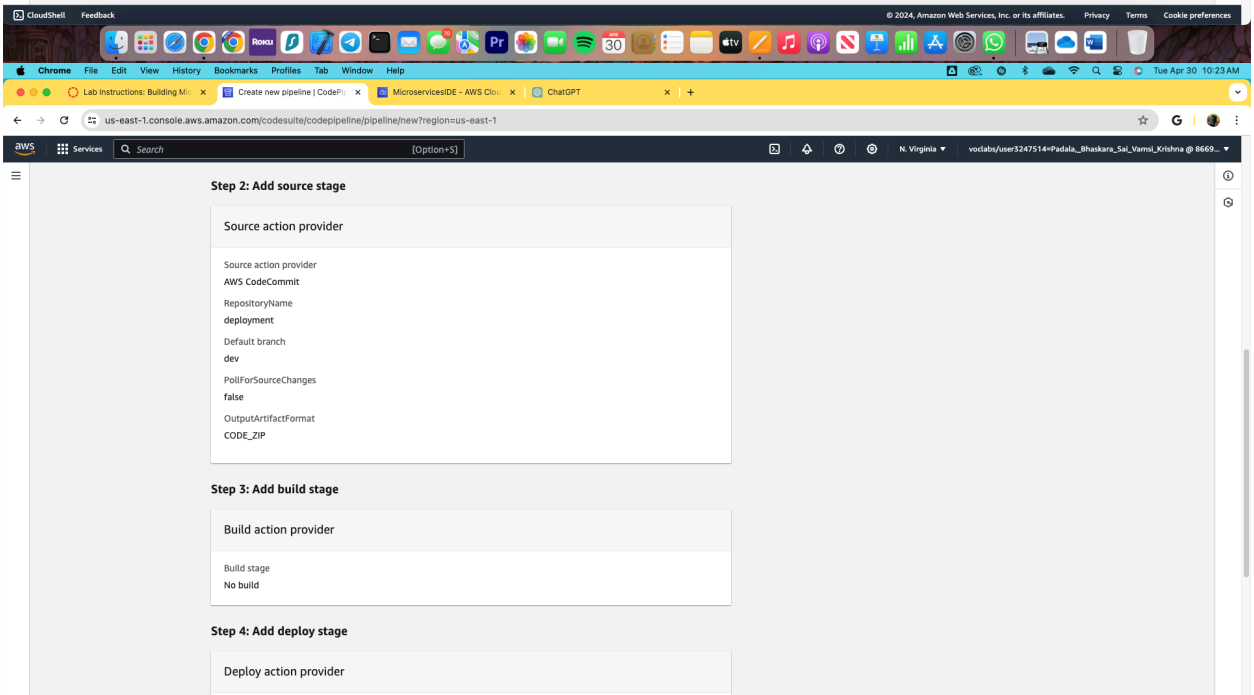
Source provider: Choose AWS CodeCommit.

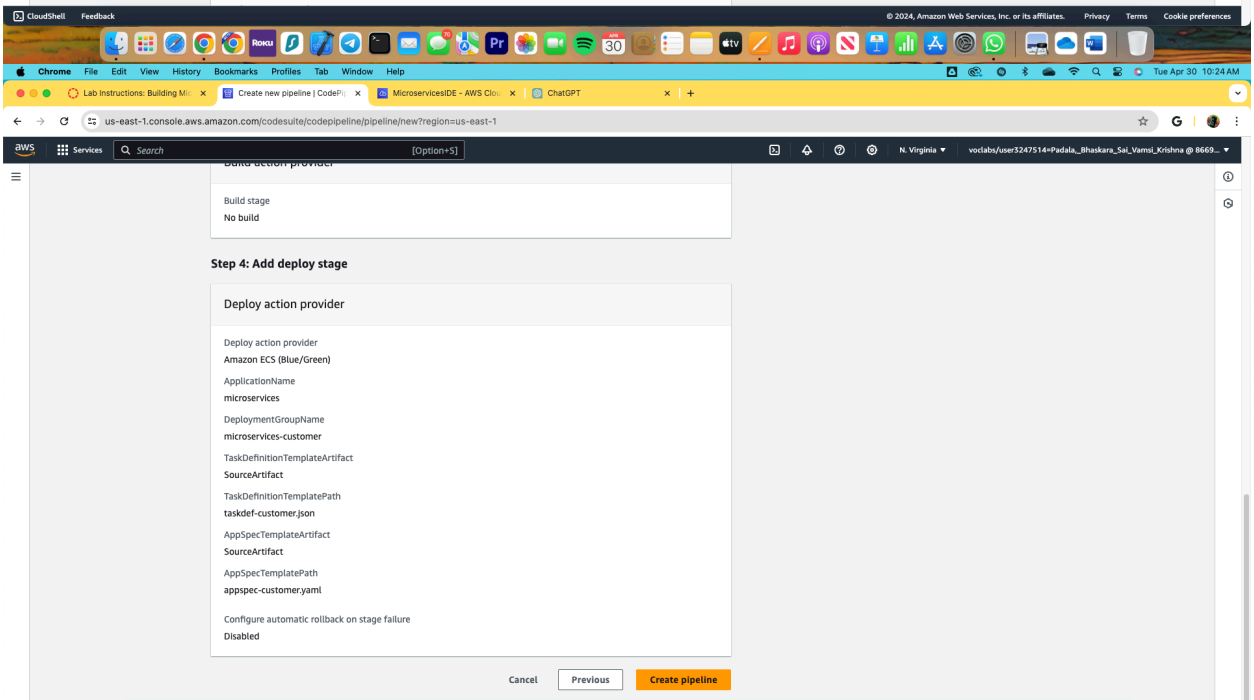
Repository name: Choose deployment.



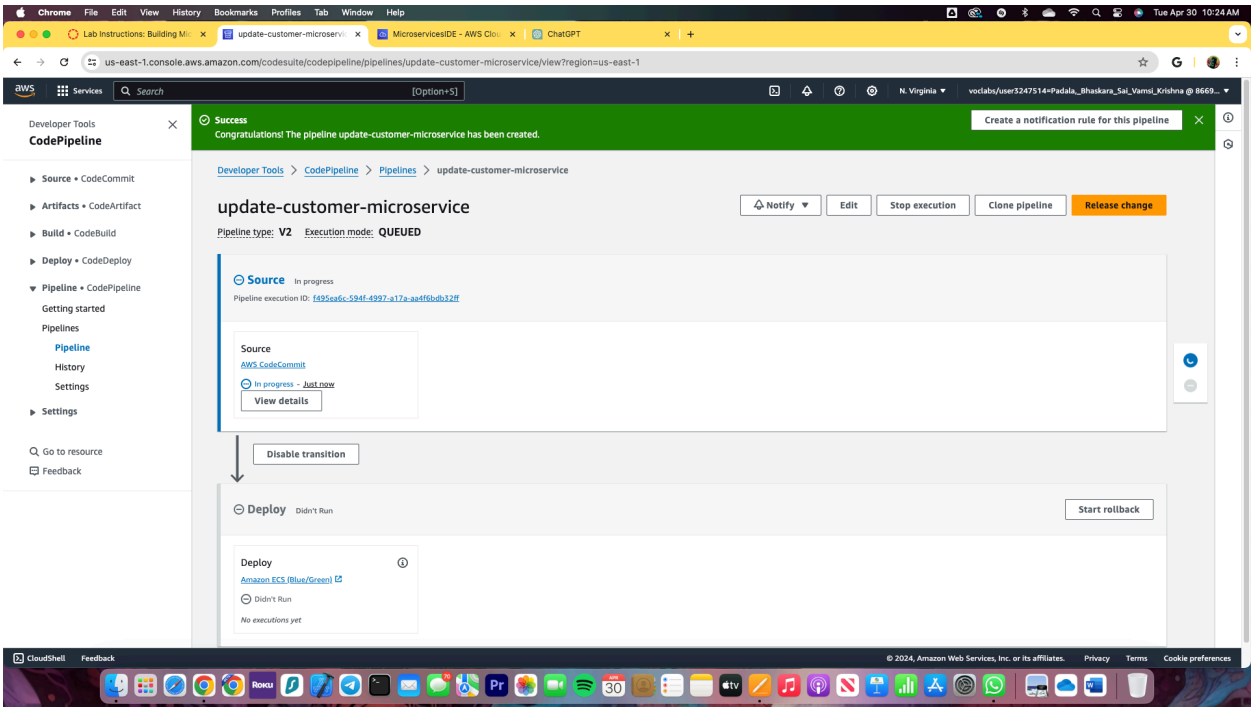
Now we will verify the configurations which we figured out again so that we won't end up making mistakes.

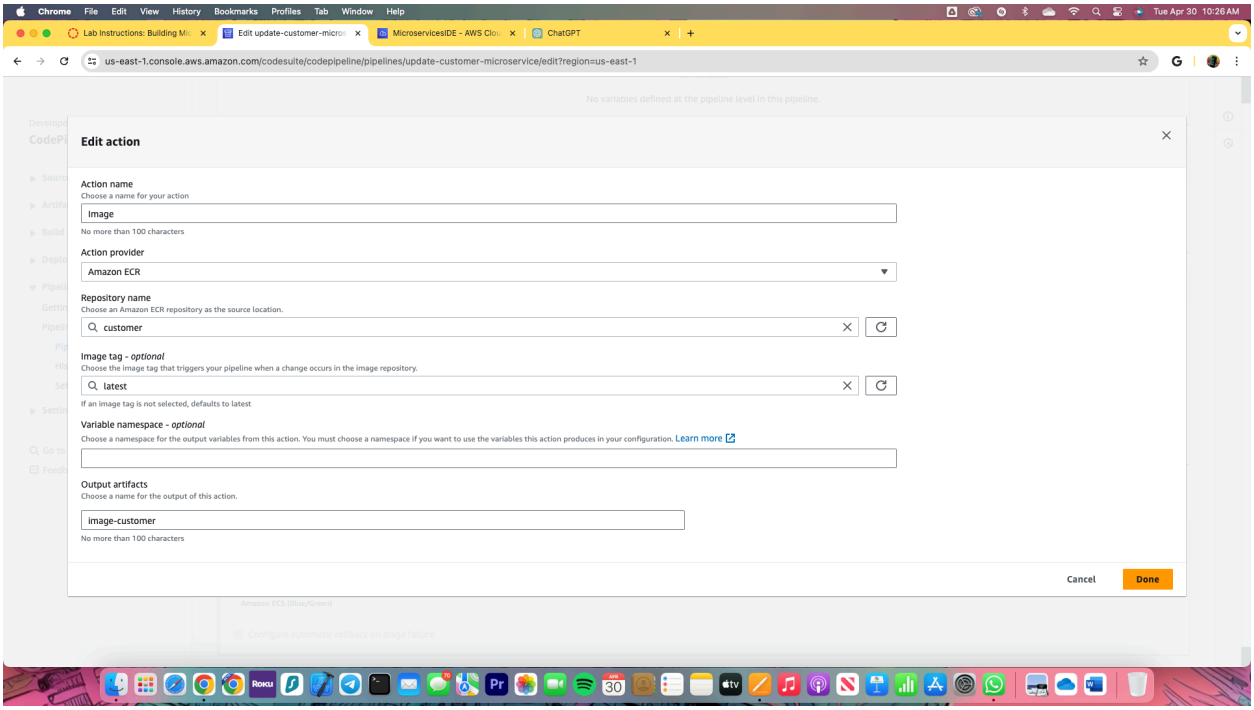

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 10:22 AM
Create new pipeline | CodePip
MicroservicesIDE - AWS Clou
ChatGPT
us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Option+S]
N. Virginia
voclabs/user3247514=Padala,_Bhaskara_Sai_Vamsi_Krishna @ 8669...
Developer Tools > CodePipeline > Pipelines > Create new pipeline
Step 1
Choose pipeline settings
Step 2
Add source stage
Step 3
Add build stage
Step 4
Add deploy stage
Step 5
Review
Review Info
Step 5 of 5
Step 1: Choose pipeline settings
Pipeline settings
Pipeline name
update-customer-microservice
Pipeline type
V2
Execution mode
QUEUED
Artifact location
A new Amazon S3 bucket will be created as the default artifact store for your pipeline
Service role name
arn:aws:iam::866964334160:role/PipelineRole
Variables
Name
Default value
Description
No variables
No variables defined at the pipeline level in this pipeline.


CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 10:23 AM
us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Step 2: Add source stage
Source action provider
Source action provider
AWS CodeCommit
RepositoryName
deployment
Default branch
dev
PollForSourceChanges
false
OutputArtifactFormat
CODE_ZIP
Step 3: Add build stage
Build action provider
Build stage
No build
Step 4: Add deploy stage
Deploy action provider

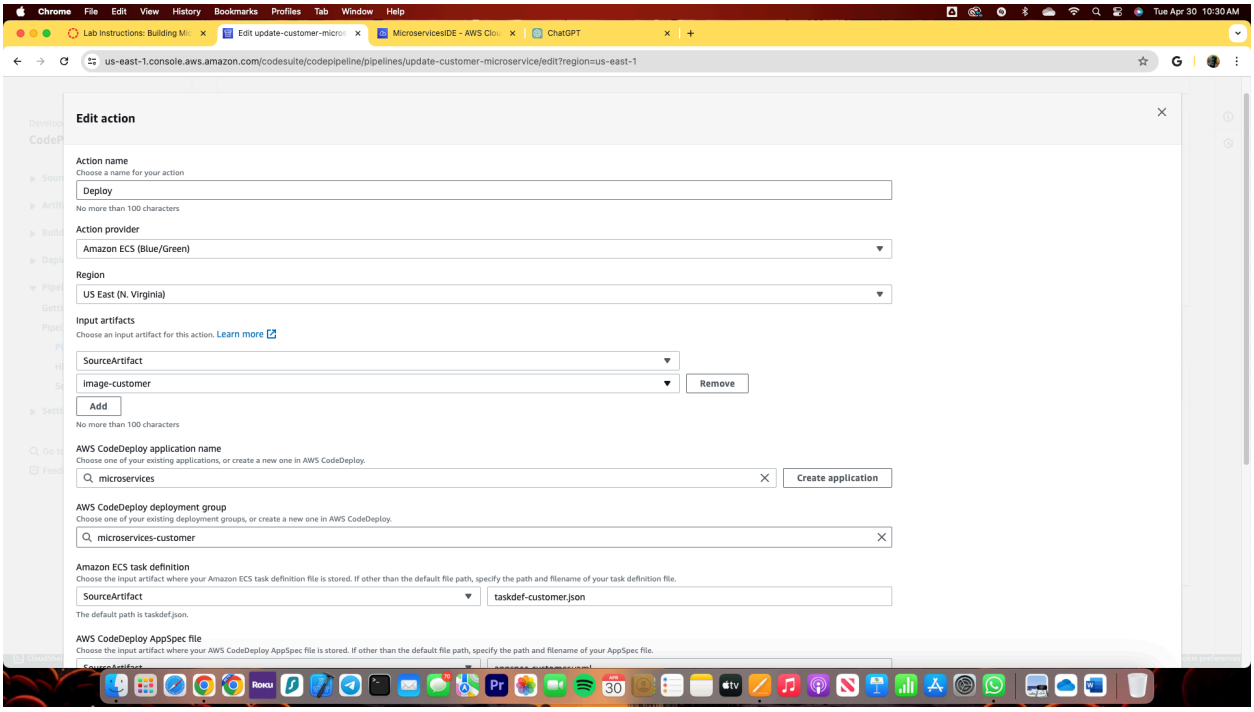

CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Apr 30 10:24 AM
us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Build stage
No build
Step 4: Add deploy stage
Deploy action provider
Deploy action provider
Amazon ECS (Blue/Green)
ApplicationName
microservices
DeploymentGroupName
microservices-customer
TaskDefinitionTemplateArtifact
SourceArtifact
TaskDefinitionTemplatePath
taskdef-customer.json
AppSpecTemplateArtifact
SourceArtifact
AppSpecTemplatePath
appspec-customer.yaml
Configure automatic rollback on stage failure
Disabled
Cancel
Previous
Create pipeline

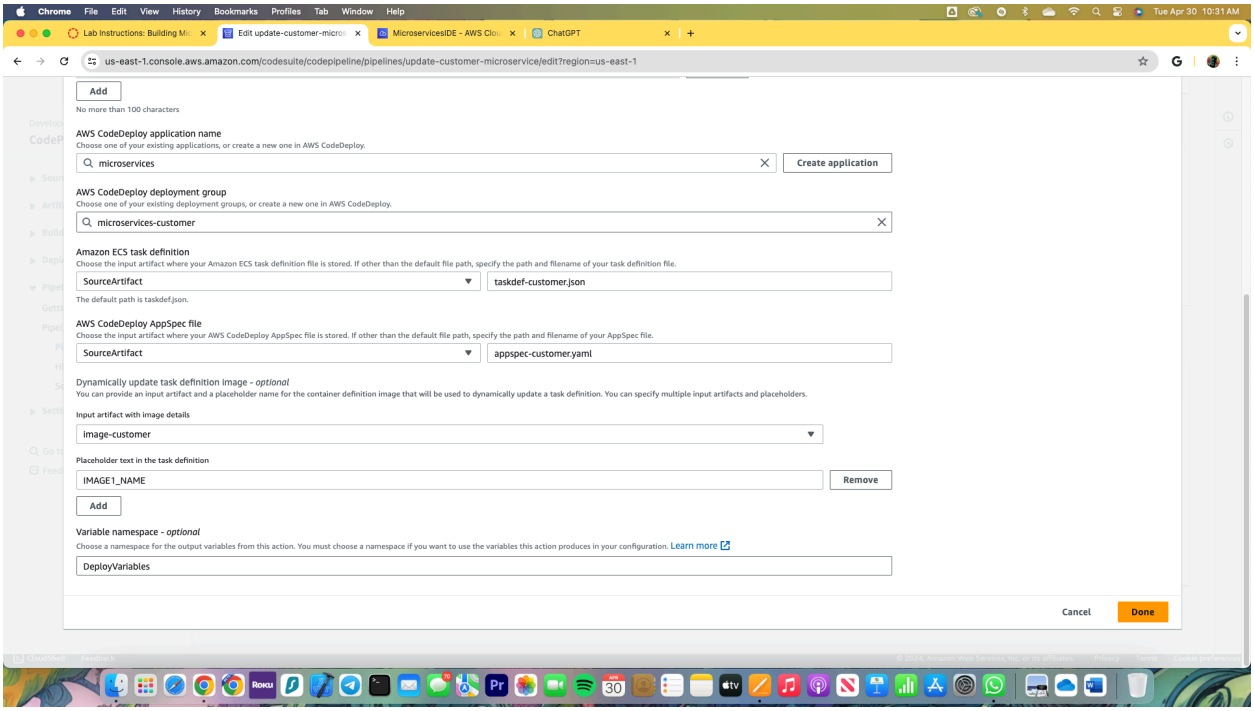
We successfully created it.





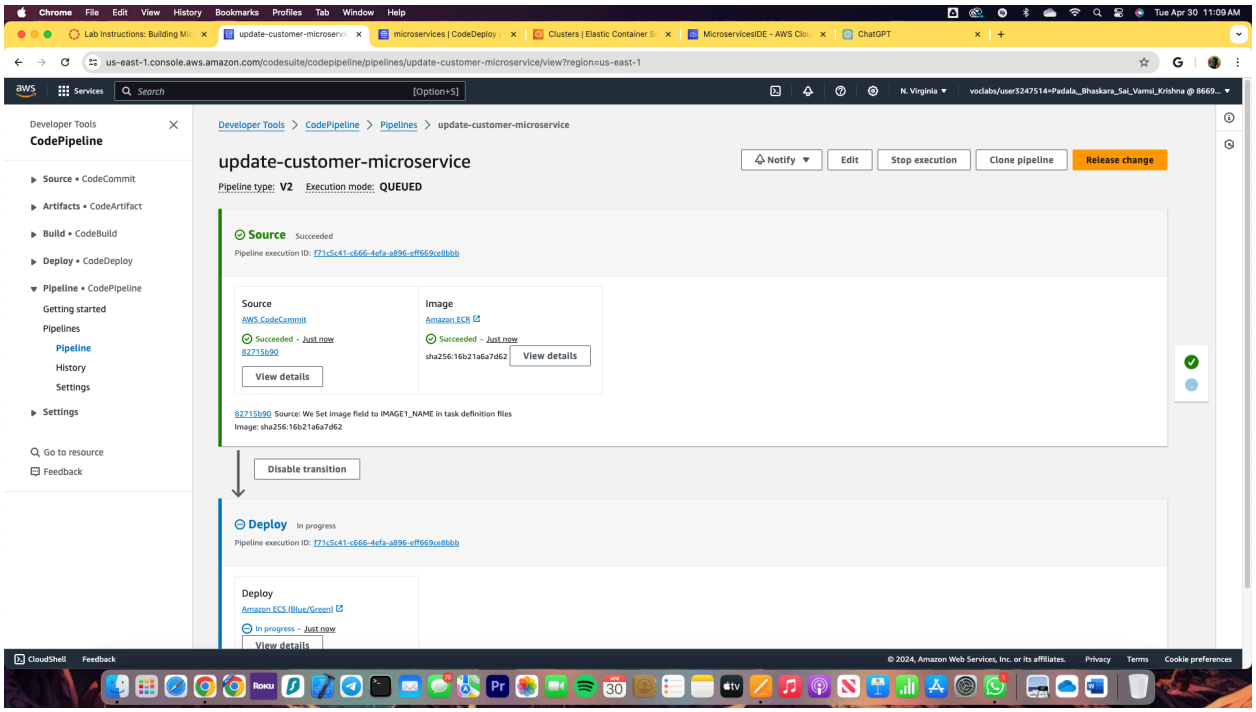
To edit the deploy action of the "update-customer-microservice" pipeline, you would typically navigate to the pipeline configuration in the AWS CodePipeline console and make the necessary changes.





Task 8.3: Test the CI/CD pipeline for the customer microservice

In this task, we will test that the CI/CD pipeline for the customer microservice functions as intended.
Navigate to the CodePipeline Console.
Access the Pipelines page and select the pipeline named "update-customer-microservice".
Trigger a test by clicking on the "Release change" link.
Confirm the release to start the test of the current pipeline settings.



Monitor the pipeline execution in the CodePipeline console to observe its behavior with the current settings.Initiate a deployment of the customer microservice on Amazon ECS using Fargate.

Access the CodePipeline console. Select the pipeline named "update-customer-microservice" from the Pipelines page.

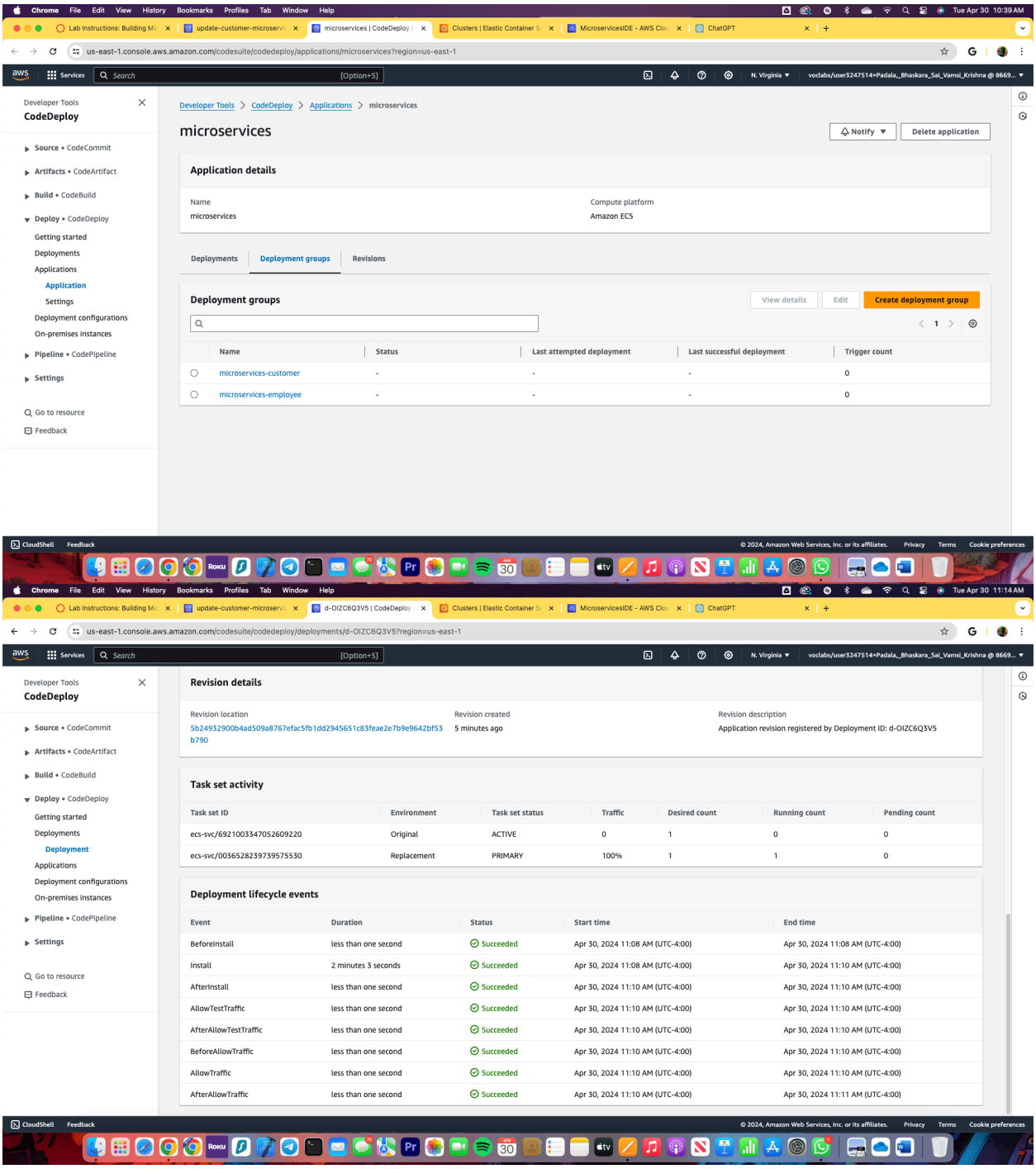
To validate the current pipeline settings, trigger a release by selecting "Release change" and then confirming the release.

Note: By triggering the pipeline, a new revision of the task definition is generated.

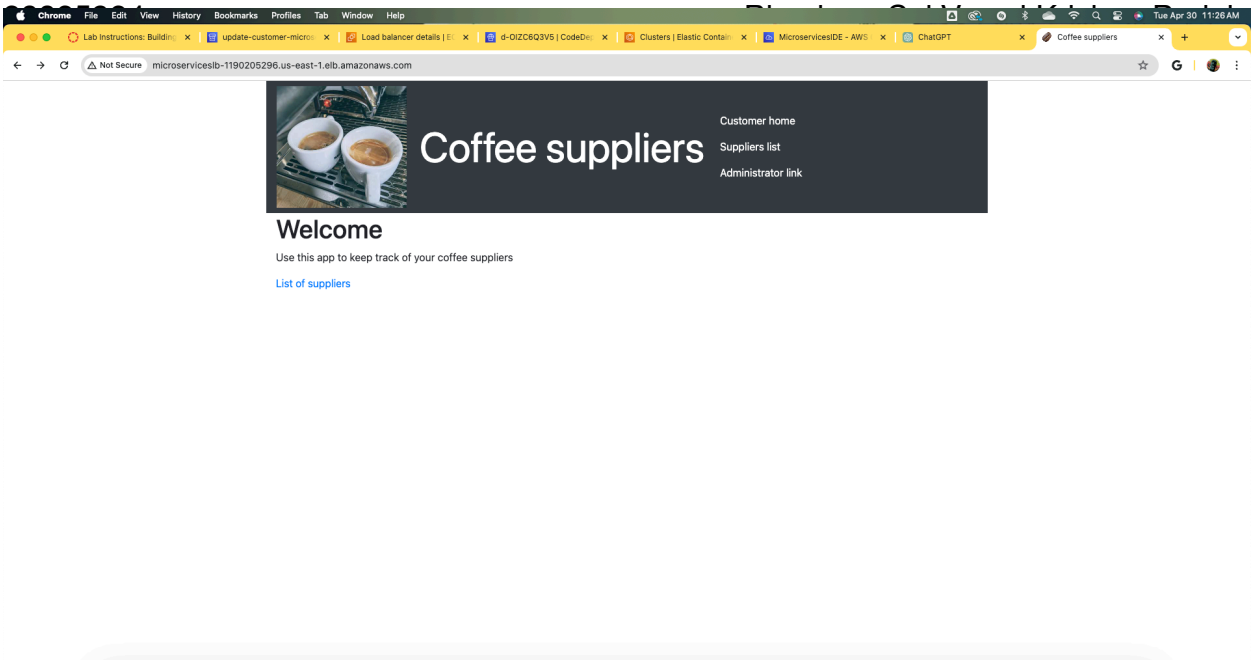
Wait until the two Source tasks indicate a status of "Succeeded - just now".

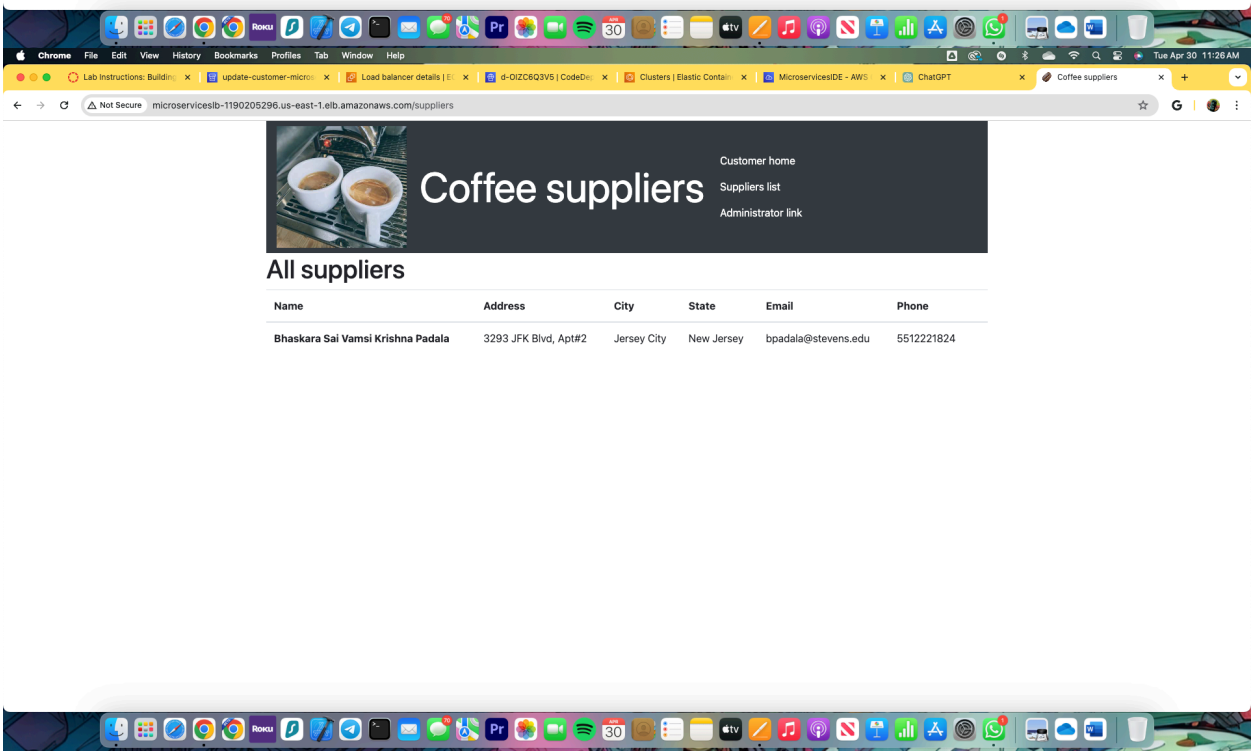
In the Deploy section, wait for the appearance of a "Details" link, and then click on it.

This action will open a CodeDeploy page in a new browser tab. Now save the changes which you performed and it will run as follows.



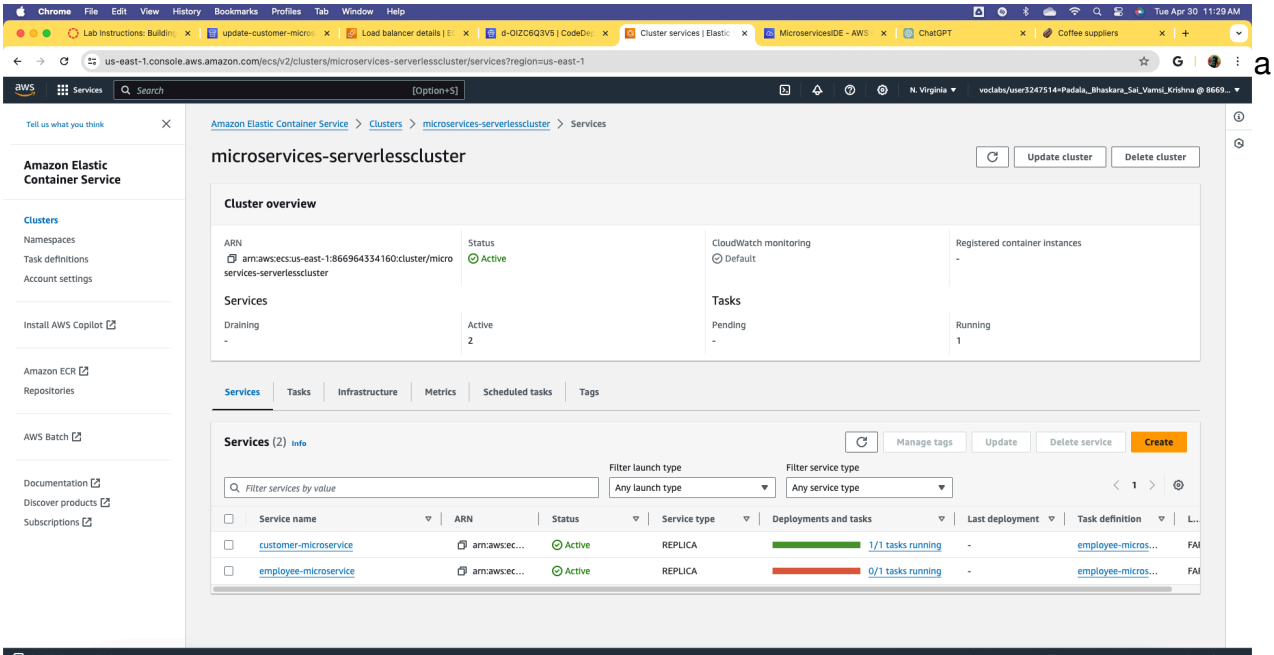
Test the customer microservice by loading it in a browser tab. Retrieve the DNS name value of the microservicesLB load balancer and paste it into a new browser tab. If the customer microservice fails to load, append ":8080" to the end of the URL and retry. Consider that the microservicesLB load balancer is configured with listeners on both port 80 and port 8080. Initially, the replacement task set operates on port 8080 for the first 5 minutes. Therefore, accessing the URL with port 80 during this time may not load the customer microservice page, but it should be accessible at port 8080. After 5 minutes, the microservice should be available on both ports. In the café web application, select either "List of suppliers" or "Suppliers list". Confirm that the suppliers page loads without displaying edit or add supplier buttons, as it is intended for customer use.

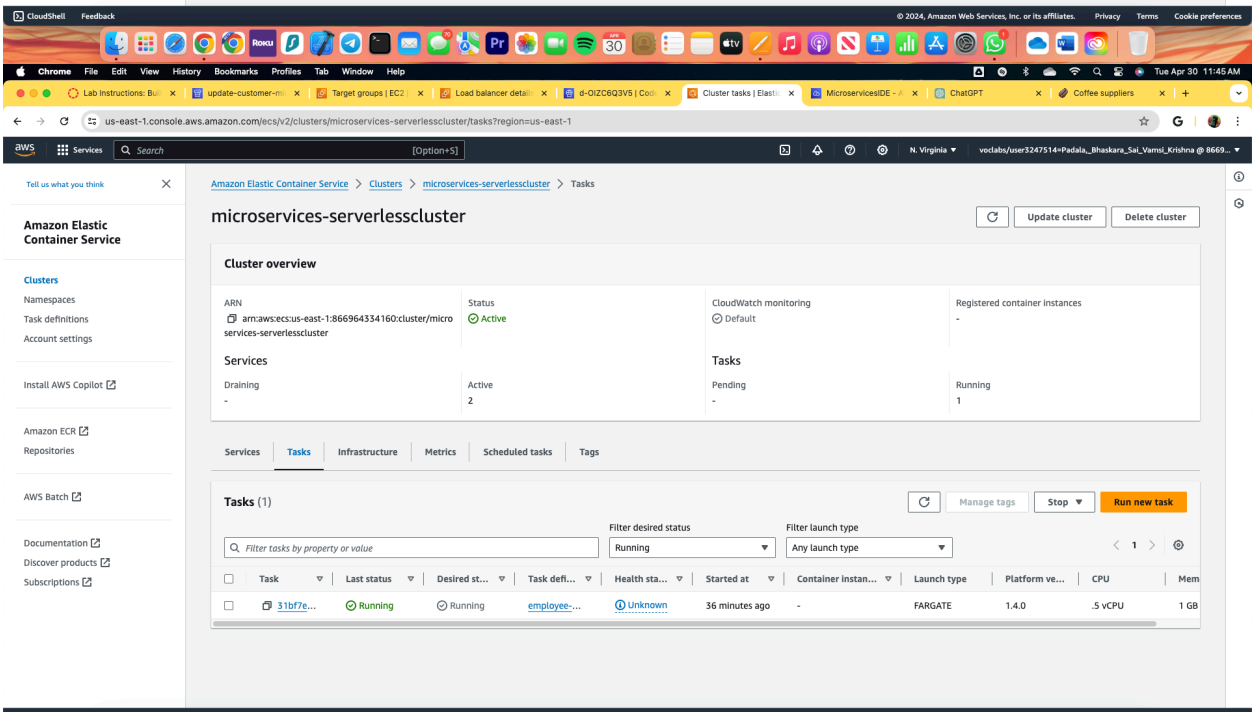


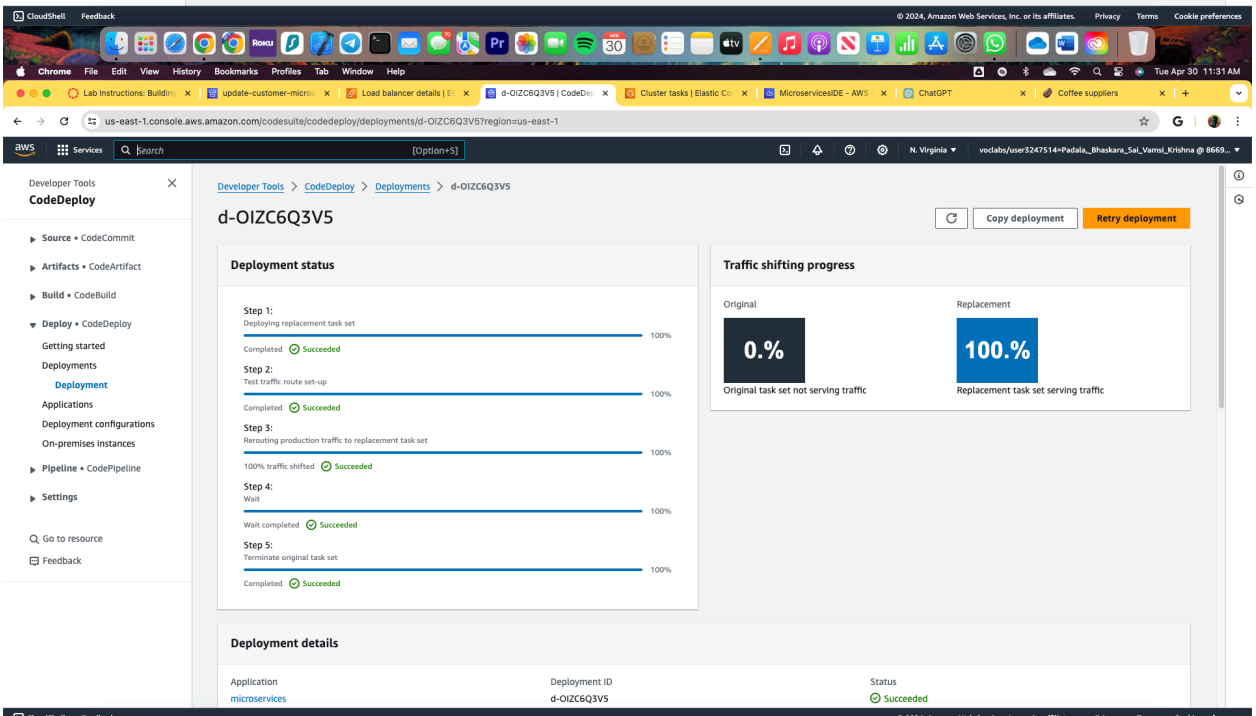


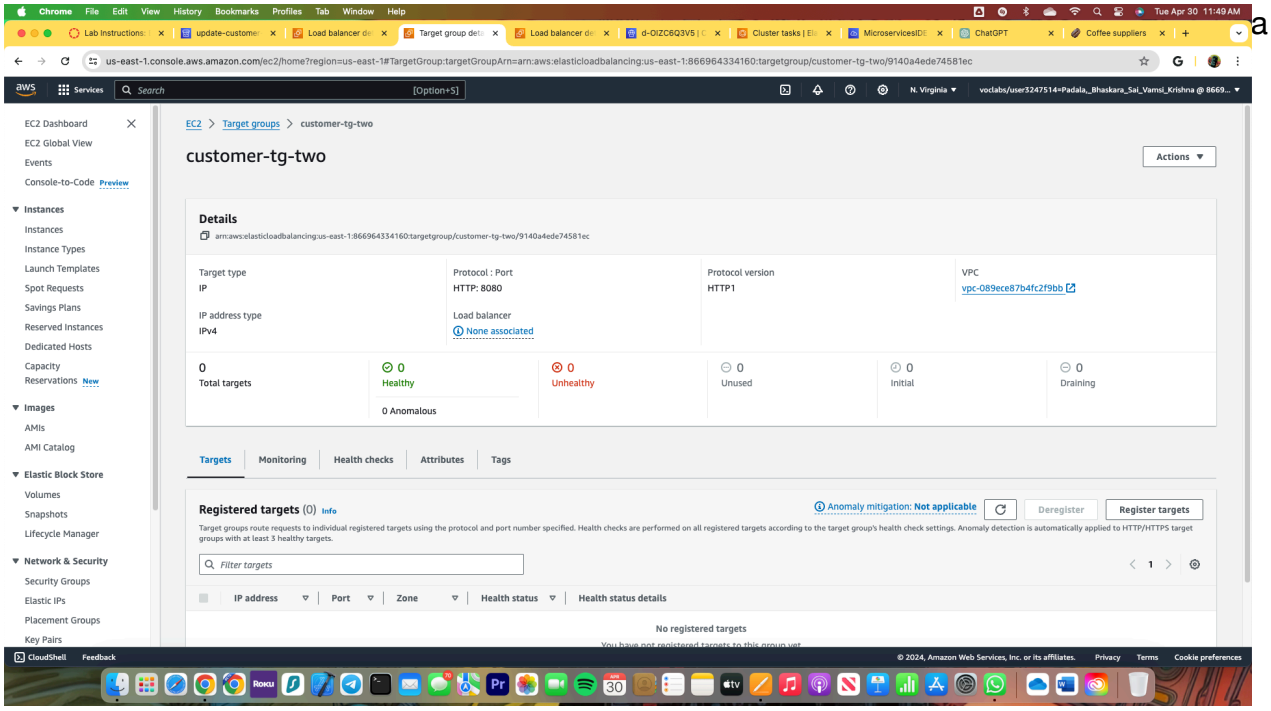
Now we will Observe the running tasks in the Amazon ECS console
Return to the CodeDeploy page that is currently open in another browser tab.

You should observe that all five steps of the deployment have succeeded, and the replacement task set is now actively serving traffic.

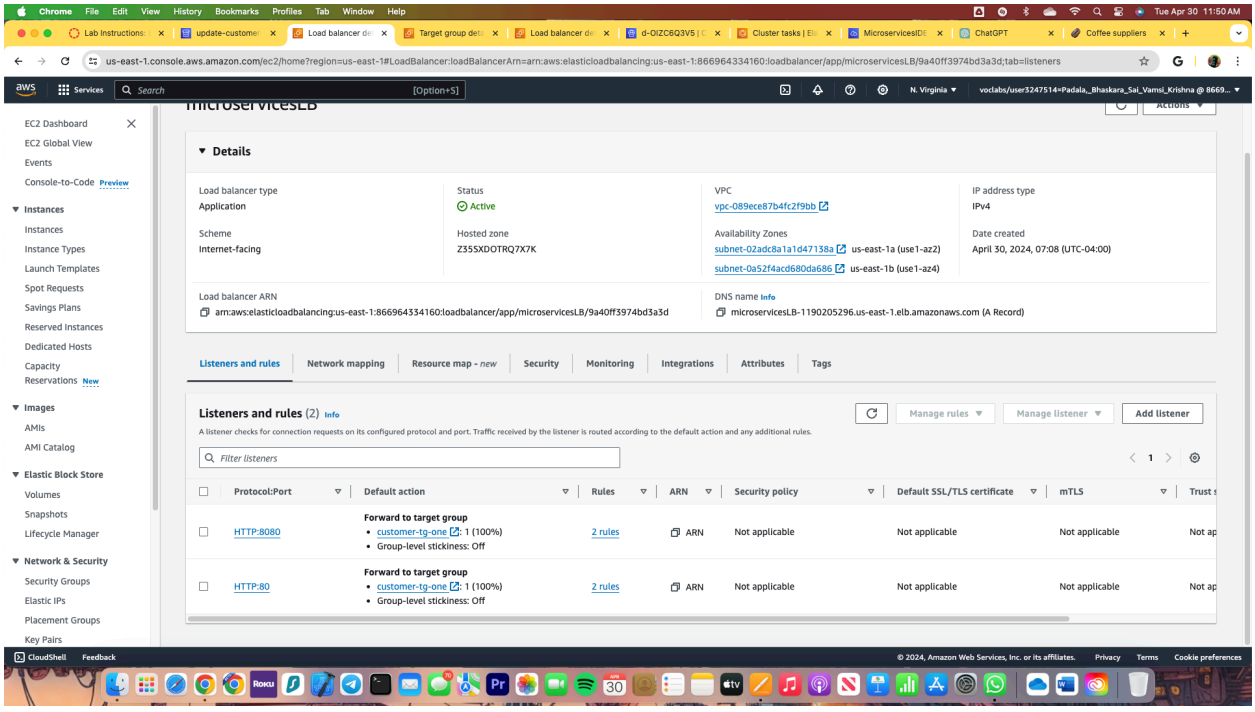

Amazon Elastic Container Service > Clusters > microservices-serverlesscluster > Services
microservices-serverlesscluster
Update cluster
Delete cluster
Cluster overview
ARN
arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster
Status
Active
CloudWatch monitoring
Default
Registered container instances
-
Services
Draining
-
Active
2
Tasks
Pending
-
Running
1
Services
Tasks
Infrastructure
Metrics
Scheduled tasks
Tags
Services (2) Info
Manage tags
Update
Delete service
Create
Filter services by value
Filter launch type
Any launch type
Filter service type
Any service type
Service name
ARN
Status
Service type
Deployments and tasks
Last deployment
Task definition
customer-microservice
Active
REPLICA
1/1 tasks running
employee-microservice
Active
REPLICA
0/1 tasks running


Amazon Elastic Container Service > Clusters > microservices-serverlesscluster > Tasks
microservices-serverlesscluster
Update cluster
Delete cluster
Cluster overview
ARN
arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster
Status
Active
CloudWatch monitoring
Default
Registered container instances
-
Services
Draining
-
Active
2
Tasks
Pending
-
Running
1
Services
Tasks
Infrastructure
Metrics
Scheduled tasks
Tags
Tasks (1)
Manage tags
Stop
Run new task
Filter tasks by property or value
Filter desired status
Running
Filter launch type
Any launch type
Task
Last status
Desired st...
Task defi...
Health sta...
Started at
Container instan...
Launch type
Platform ve...
CPU
Mem
31bf7e...
Running
Running
employee-...
Unknown
36 minutes ago
-
FARGATE
1.4.0
.5 vCPU
1 GB


Developer Tools > CodeDeploy > Deployments > d-OIZC6Q3V5
d-OIZC6Q3V5
Copy deployment
Retry deployment
Deployment status
Step 1:
Deploying replacement task set
100%
Completed
Succeeded
Step 2:
Test traffic route set-up
100%
Completed
Succeeded
Step 3:
Rerouting production traffic to replacement task set
100%
100% traffic shifted
Succeeded
Step 4:
Wait
100%
Wait completed
Succeeded
Step 5:
Terminate original task set
100%
Completed
Succeeded
Traffic shifting progress
Original
0.%
Original task set not serving traffic
Replacement
100.%
Replacement task set serving traffic
Deployment details
Application
microservices
Deployment ID
d-OIZC6Q3V5
Status
Succeeded

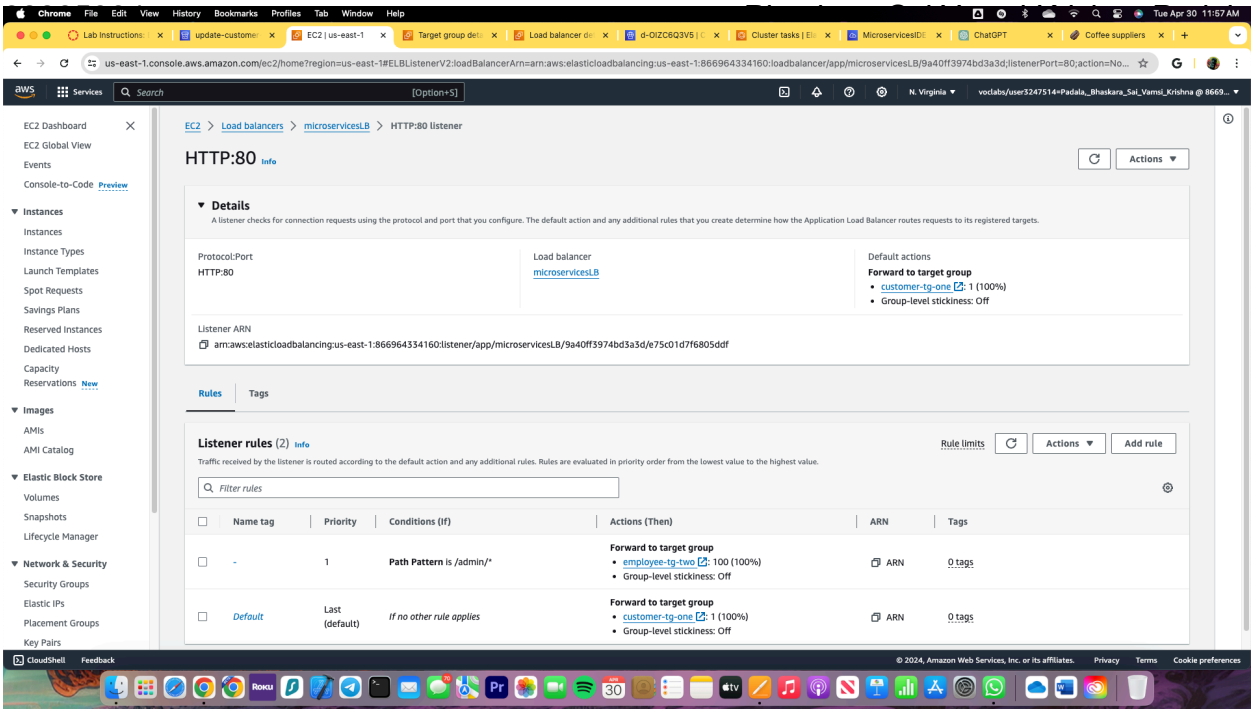


You may observe that the customer-tg-two target group is no longer linked with the load balancer. This occurs because CodeDeploy is overseeing the management of load balancer listener rules and may have concluded that certain target groups are no longer required.

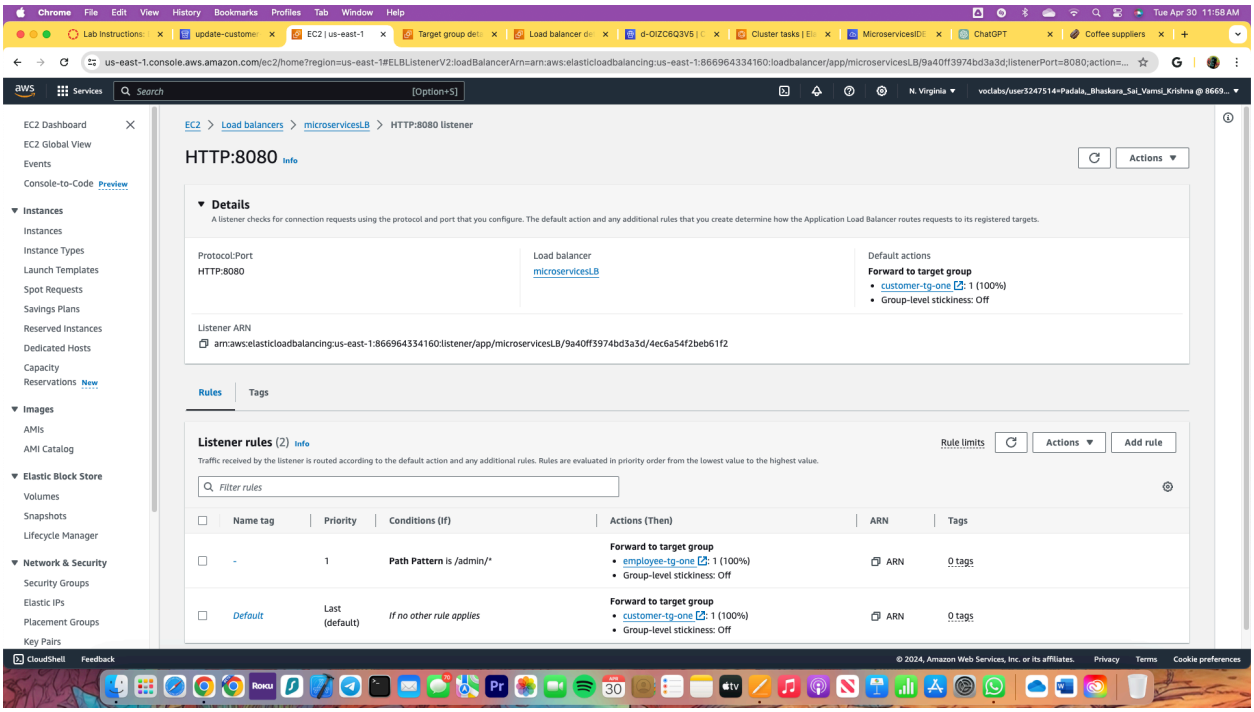


Take note of the HTTP:80 listener rules.

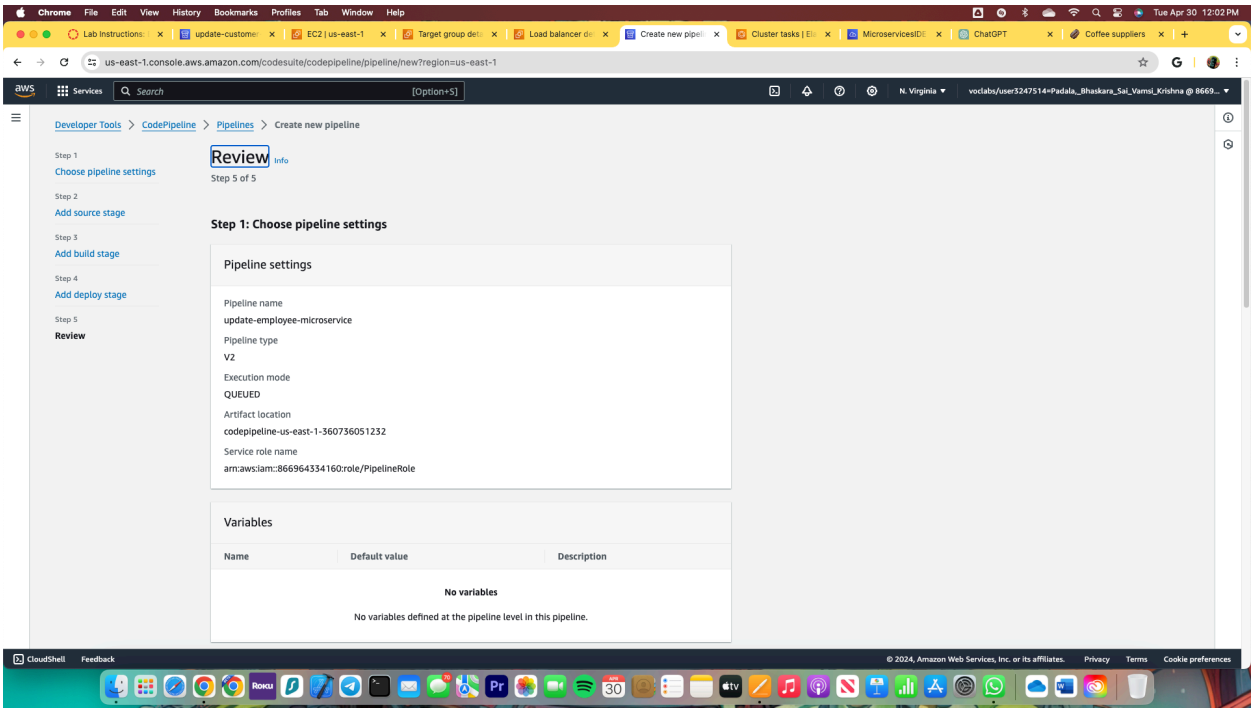
The default rule has been altered. Previously, the default rule, which applies if no other rule matches, directed traffic to customer-tg-two. However, it now directs traffic to customer-tg-one. This change was made by CodeDeploy as it actively manages your Application Load Balancer. Check the HTTP:8080 listener rules.

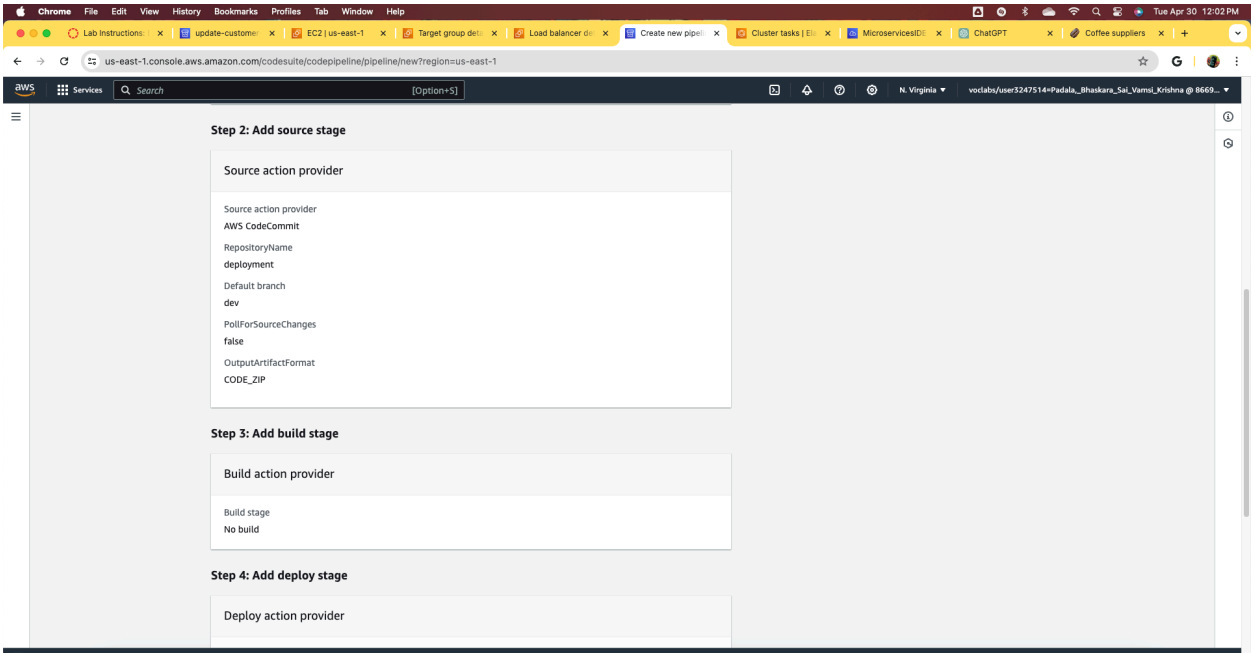


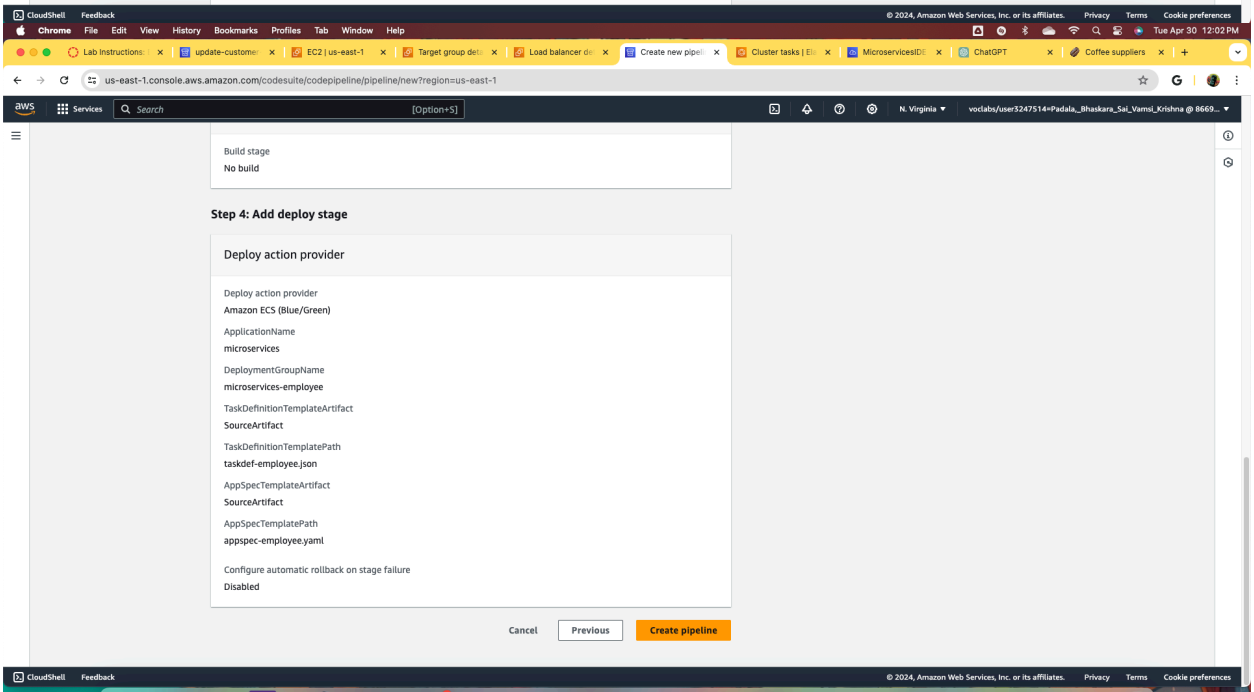
Both rules continue to route traffic to the "one" target groups.

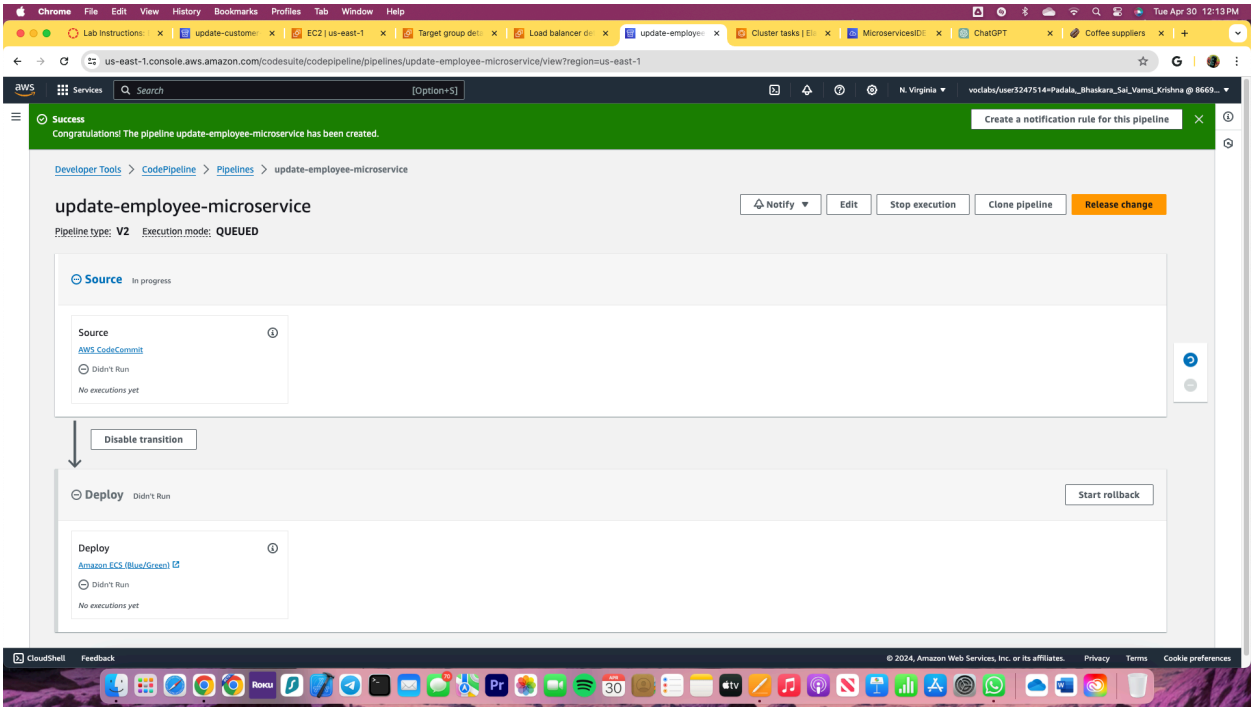


Task 8.4: Create a pipeline for the employee microservice
In this task, we will create the pipeline for the employee microservice.

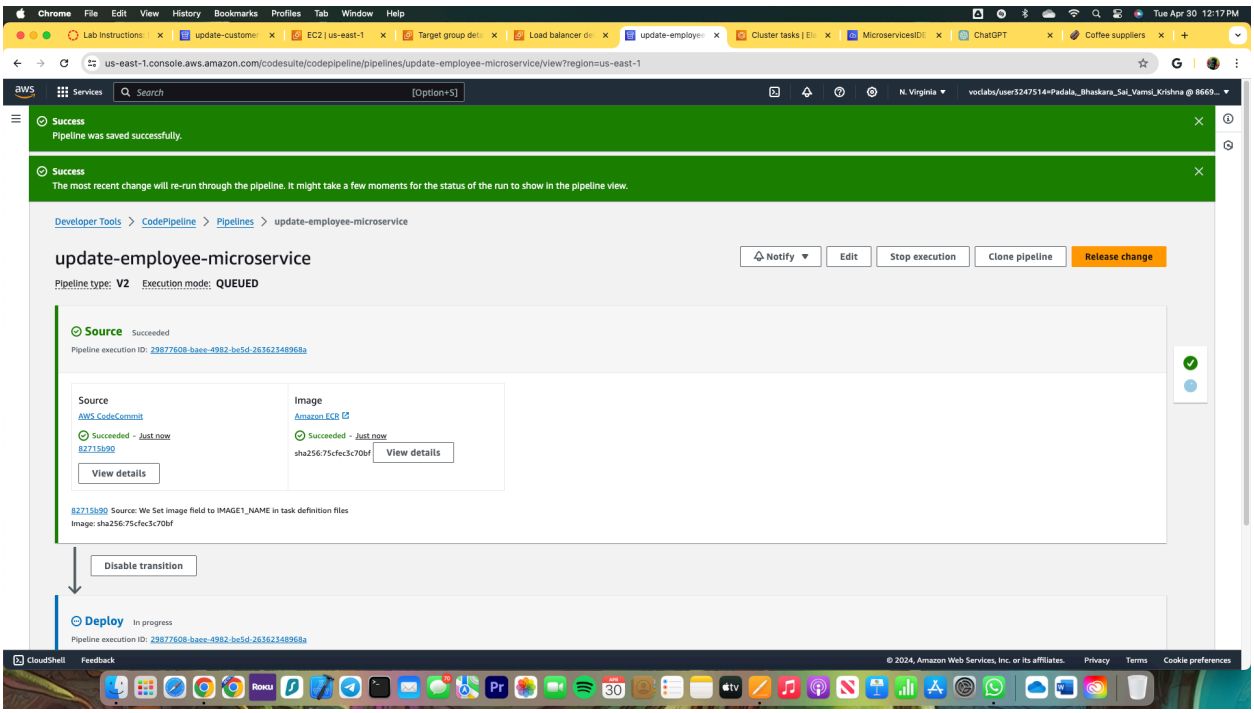

Developer Tools > CodePipeline > Pipelines > Create new pipeline
Step 1
Choose pipeline settings
Step 2
Add source stage
Step 3
Add build stage
Step 4
Add deploy stage
Step 5
Review
Review Info
Step 5 of 5
Step 1: Choose pipeline settings
Pipeline settings
Pipeline name
update-employee-microservice
Pipeline type
V2
Execution mode
QUEUED
Artifact location
codepipeline-us-east-1-360736051232
Service role name
arn:aws:iam::866964334160:role/PipelineRole
Variables
Name
Default value
Description
No variables
No variables defined at the pipeline level in this pipeline.


Step 2: Add source stage
Source action provider
Source action provider
AWS CodeCommit
RepositoryName
deployment
Default branch
dev
PollForSourceChanges
false
OutputArtifactFormat
CODE_ZIP
Step 3: Add build stage
Build action provider
Build stage
No build
Step 4: Add deploy stage
Deploy action provider


Build stage
No build
Step 4: Add deploy stage
Deploy action provider
Deploy action provider
Amazon ECS (Blue/Green)
ApplicationName
microservices
DeploymentGroupName
microservices-employee
TaskDefinitionTemplateArtifact
SourceArtifact
TaskDefinitionTemplatePath
taskdef-employee.json
AppSpecTemplateArtifact
SourceArtifact
AppSpecTemplatePath
appspec-employee.yaml
Configure automatic rollback on stage failure
Disabled
Cancel
Previous
Create pipeline



In this task, the objective is to create a pipeline tailored for the employee microservice, adhering to specific configurations. The pipeline, named "update-employee-microservice," is set to utilize AWS CodeCommit as the source provider, with the repository named "deployment" and the "dev" branch specified. For deployment, Amazon ECS (Blue/Green) is



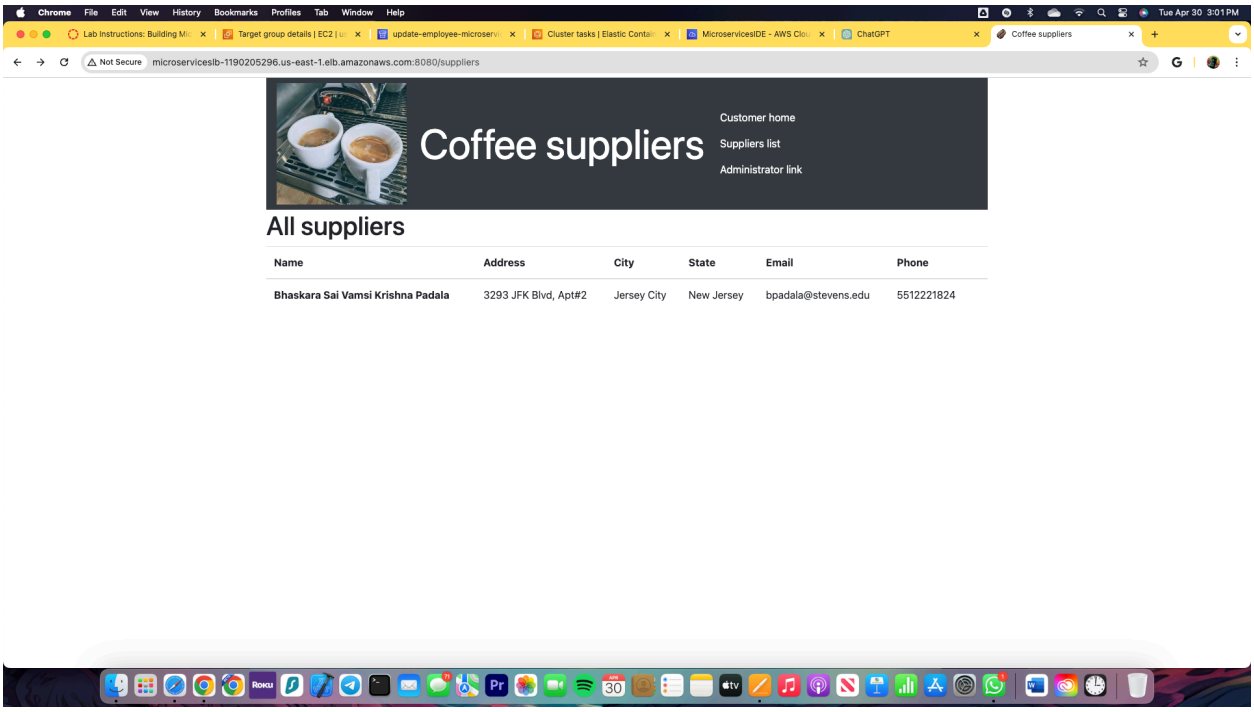
selected as the deploy provider, where AWS CodeDeploy orchestrates the deployment process. The task definition and AppSpec file are sourced from the specified artifacts, "taskdef-employee.json" and "appspec-employee.yaml," respectively. Moreover, an additional source is integrated into the pipeline, leveraging Amazon ECR to fetch the latest image tagged

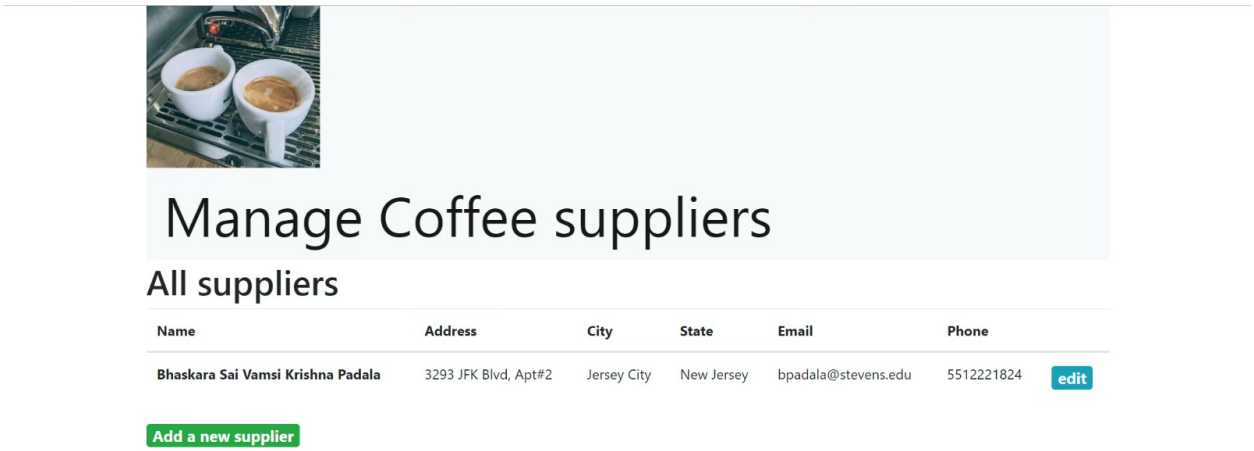
as "latest" from the "employee" repository. Modifications are made to the Amazon ECS (Blue/Green) action within the deploy stage to incorporate the newly added input artifact, "image-employee," and designate "IMAGE1_NAME" as the placeholder text in the task definition to dynamically update the image reference during deployment. Through these configurations, the pipeline is poised to seamlessly manage the deployment process for the employee microservice, ensuring efficient and reliable updates.

Task 8.5: Test the CI/CD pipeline for the employee microservice

In this task, we will test the pipeline that you just defined for the employee microservice.

Here we got The status of deployments and tasks will evolve as the blue/green deployment progresses through its lifecycle events.

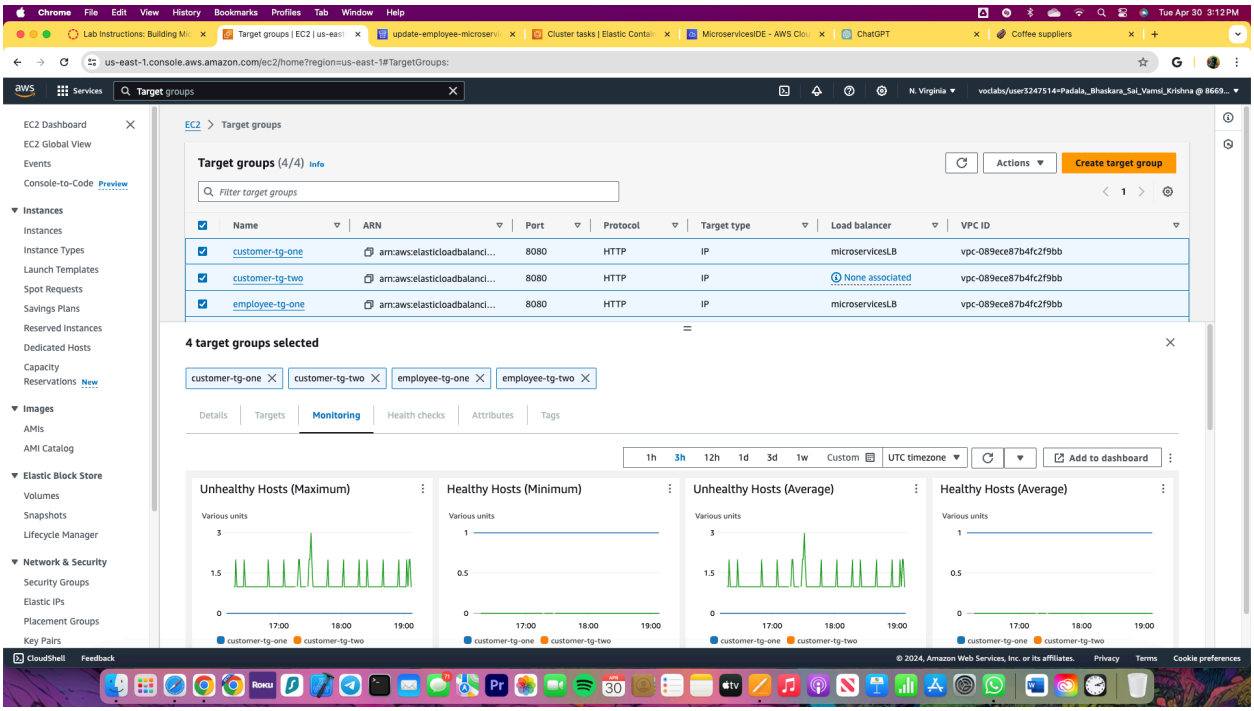




Check the CodeDeploy page to verify that all five deployment steps have completed successfully and the replacement task set is actively serving traffic.

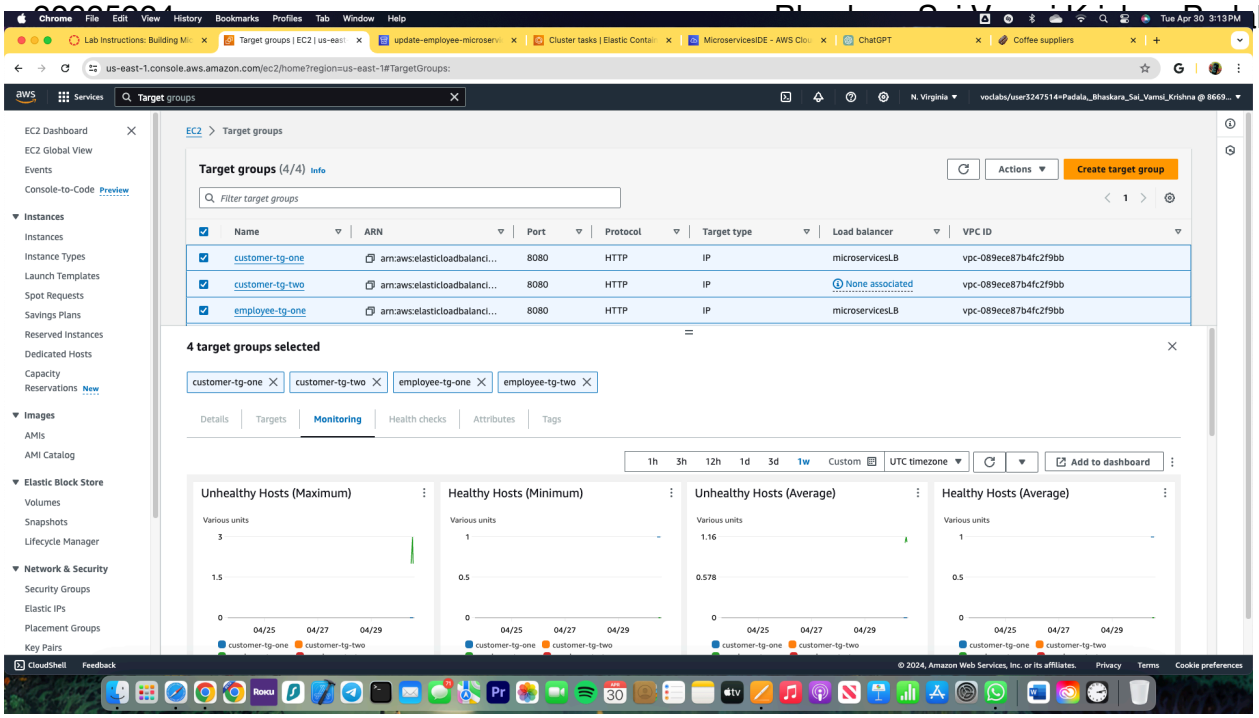
Task 8.6: Observe how CodeDeploy modified the load balancer listener rules

Review the configurations of the load balancer and target groups. Navigate to the Target Groups section in the Amazon EC2 console. Observe that the customer-tg-two target group is no longer linked with the load balancer. This adjustment is due to CodeDeploy's management of the load balancer listener rules. Note: If you repeat this step, the currently attached and unattached target groups may differ. Examine the HTTP:80 listener rules.The default rule has been modified. Previously, the default rule directed traffic to customer-tg-two for the "If no other rule applies" scenario, but it now points to customer-tg-one. Observe the HTTP:8080 listener rules. Both rules continue to route traffic to the "one" target groups.

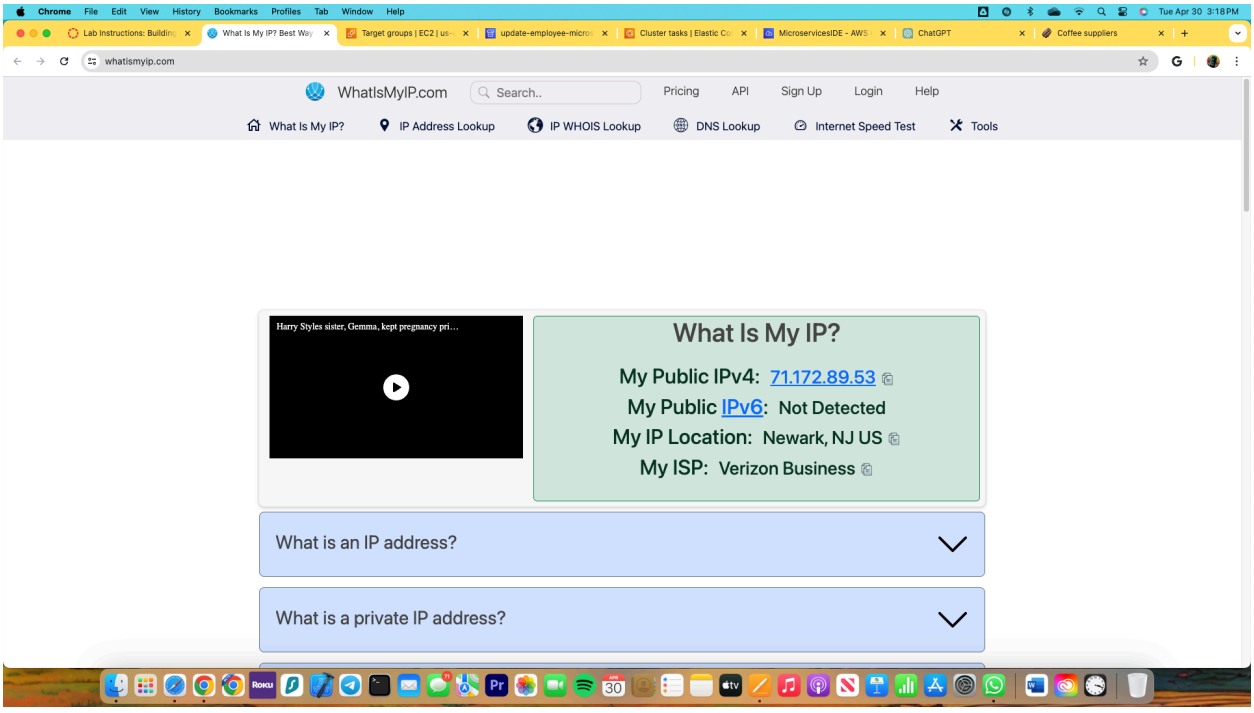


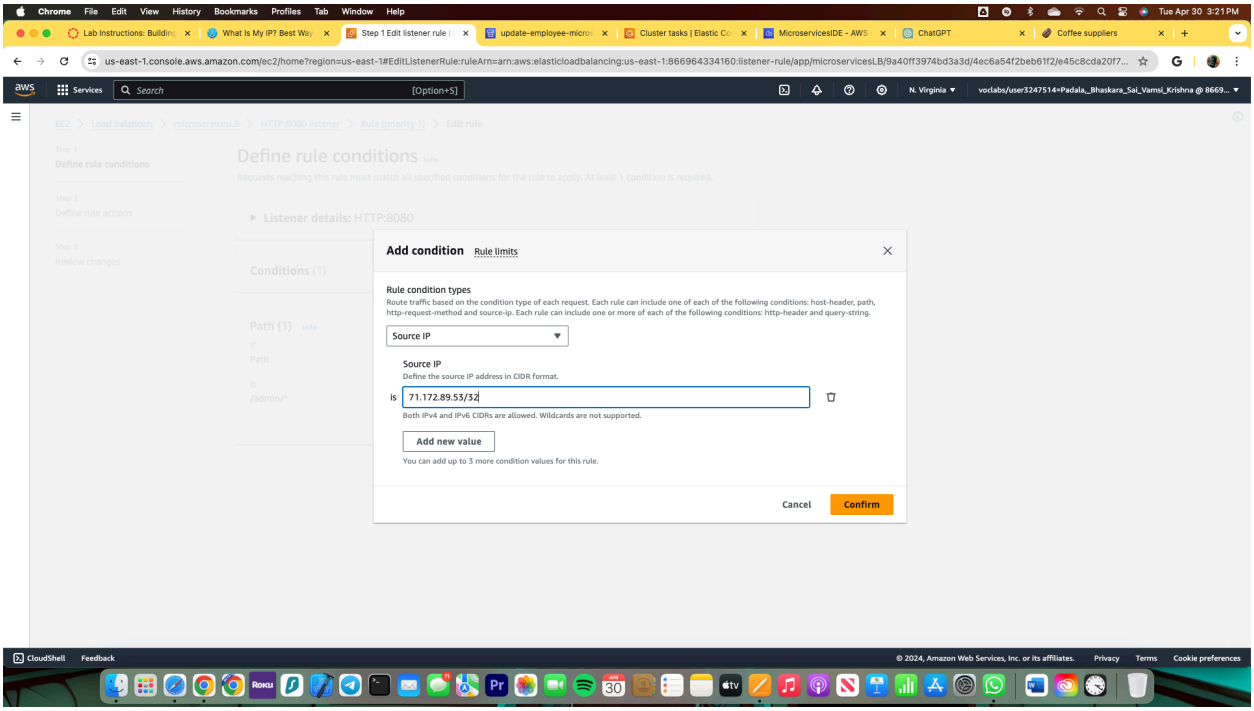
Inspect the load balancer and target group configurations. Navigate to the Target Groups section within the Amazon EC2 console. Please refresh the page if it was already open. Observe that the customer-tg-two target group is no longer linked with the load balancer. This alteration is a result of CodeDeploy managing the load balancer listener rules. Please note that if you repeat this step, the currently attached and unattached target groups may differ. Review the HTTP:80 listener rules. There has been a modification in the default rule. Previously, the default rule directed traffic to customer-tg-two for the scenario "If no other rule applies," but it now directs traffic to customer-tg-one. Examine the HTTP:8080 listener rules. Both rules persist in forwarding traffic to the "one" target groups.

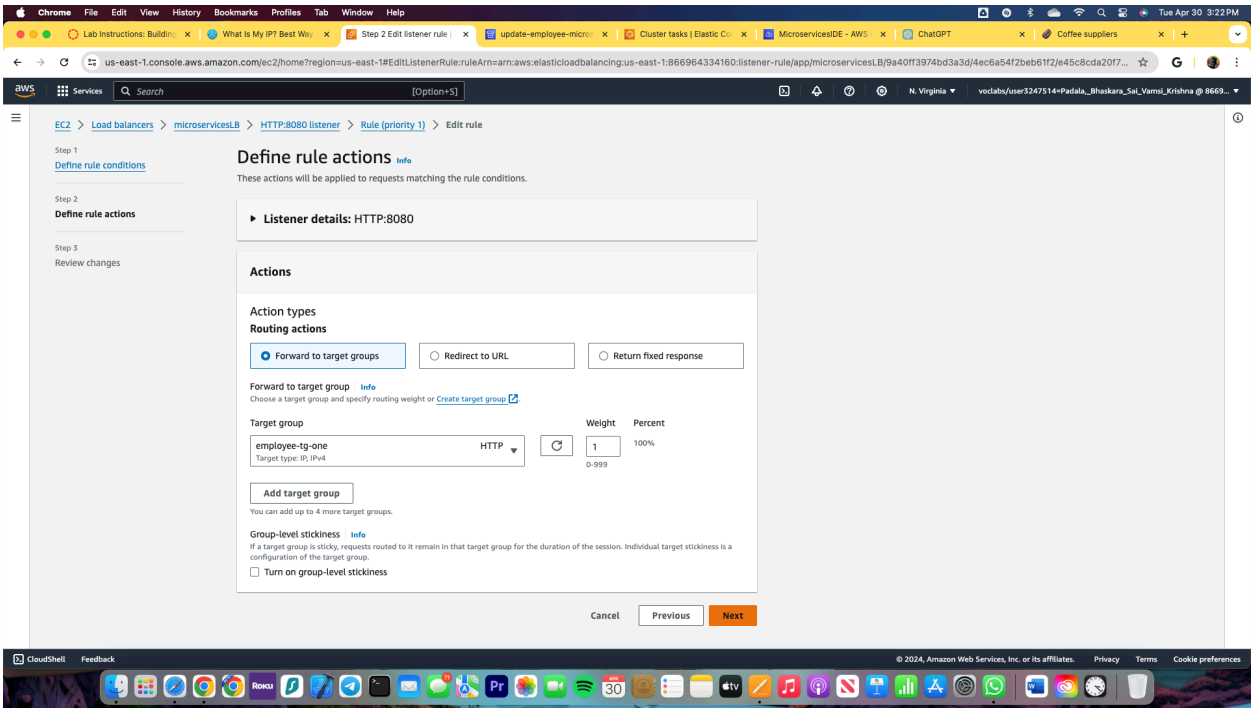
Phase 9: Adjusting the microservice code to cause a pipeline to run again

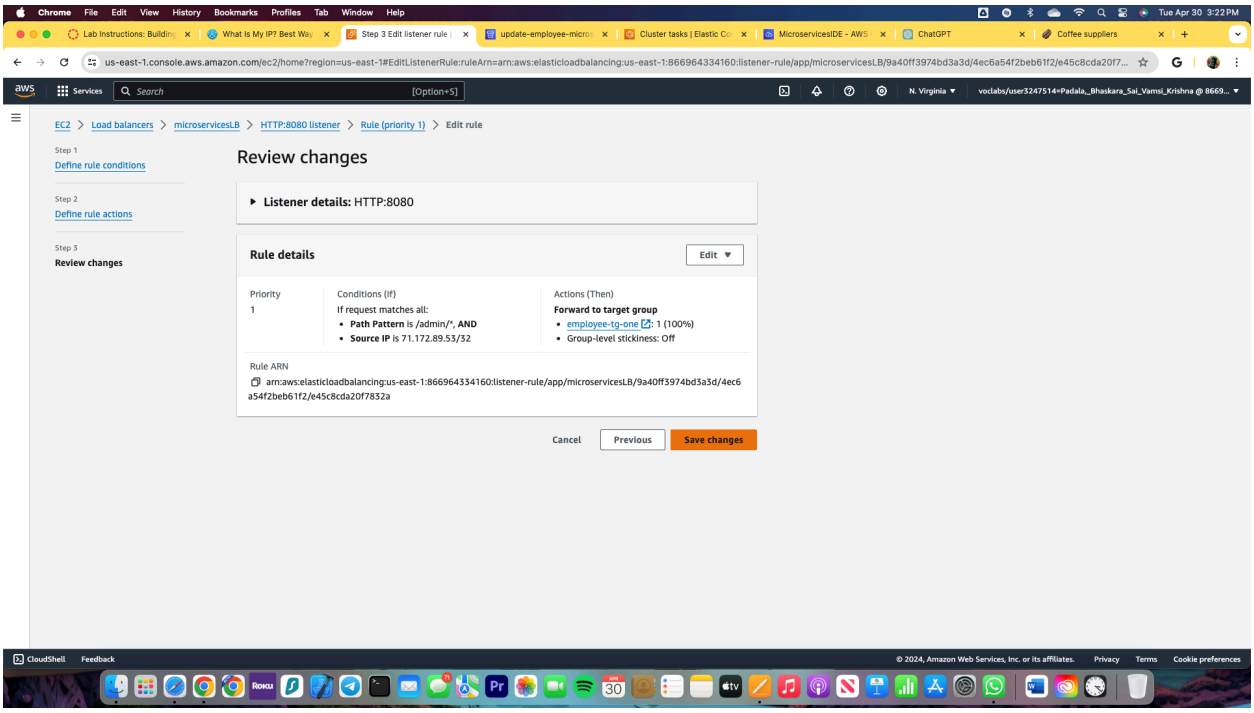


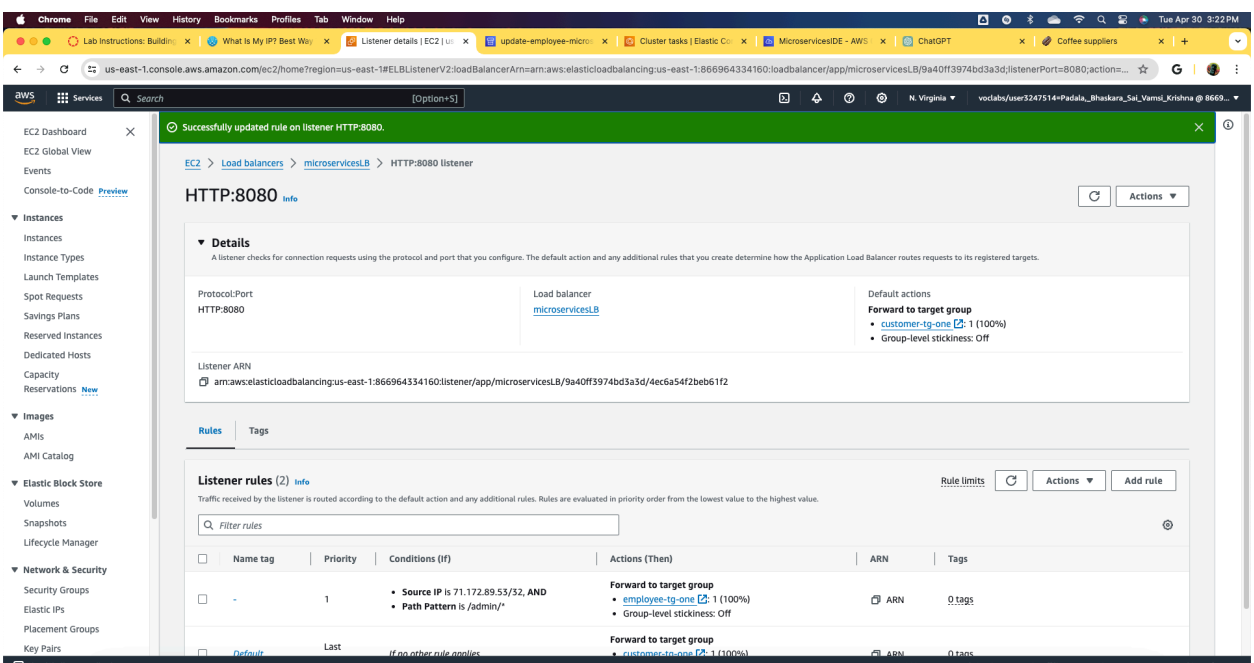
In this task, we will limit access to the employee microservice to only people who try to connect to it from a specific IP address. By limiting the source IP to a specific IP address, only users who access the application from that IP can access the pages, and edit or delete supplier entries.

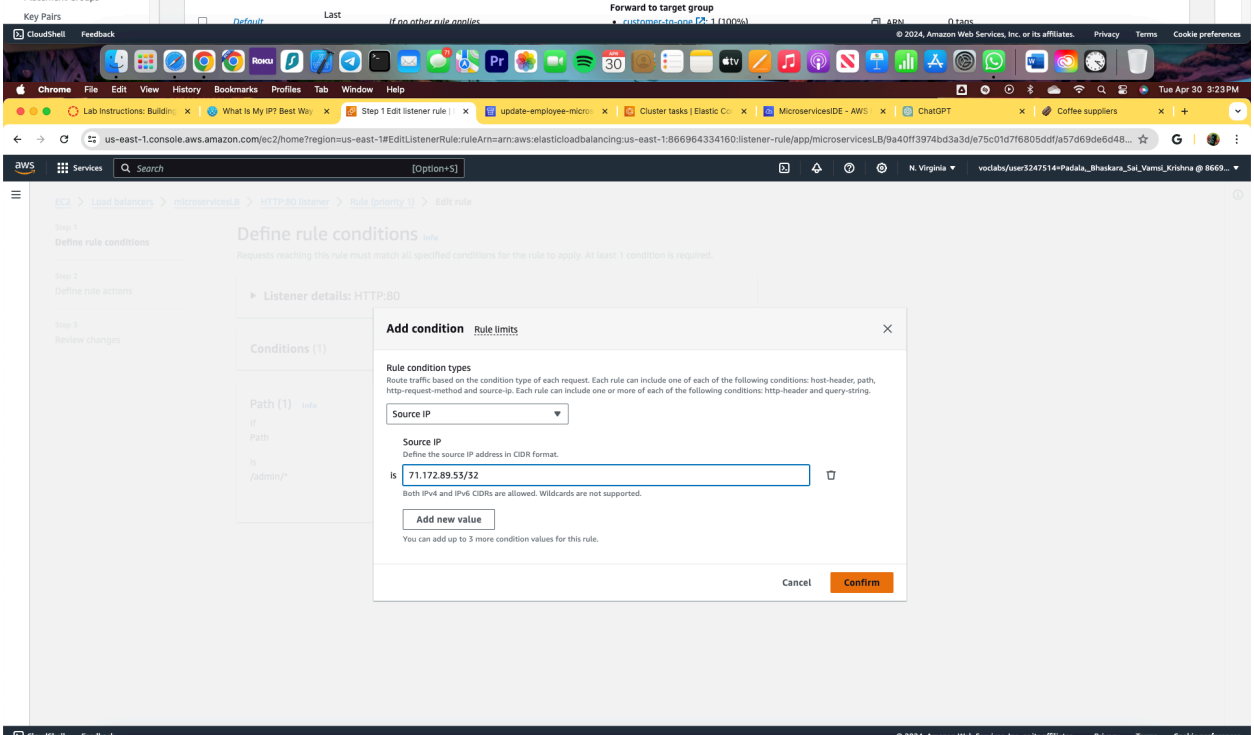


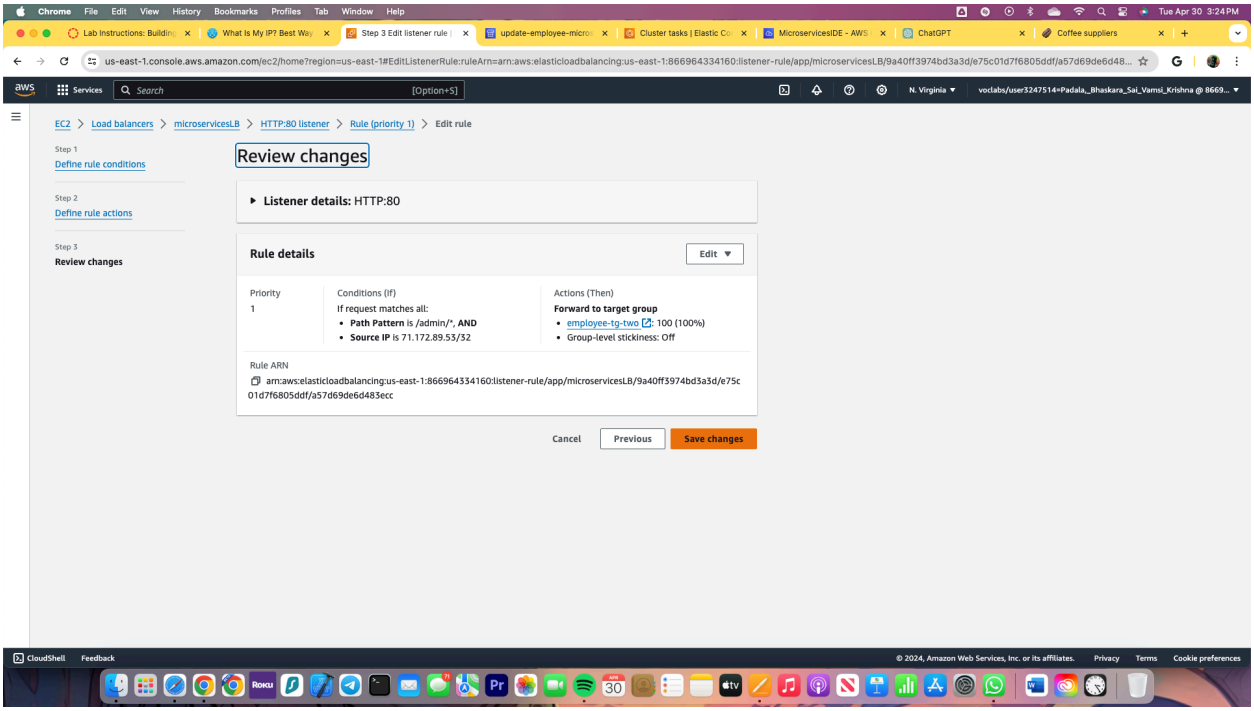

Add condition
Rule limits
Rule condition types
Route traffic based on the condition type of each request. Each rule can include one of each of the following conditions: host-header, path, http-request-method and source-ip. Each rule can include one or more of each of the following conditions: http-header and query-string.
Source IP
Source IP
Define the source IP address in CIDR format.
is
71.172.89.53/32
Both IPv4 and IPv6 CIDRs are allowed. Wildcards are not supported.
Add new value
You can add up to 3 more condition values for this rule.
Cancel
Confirm

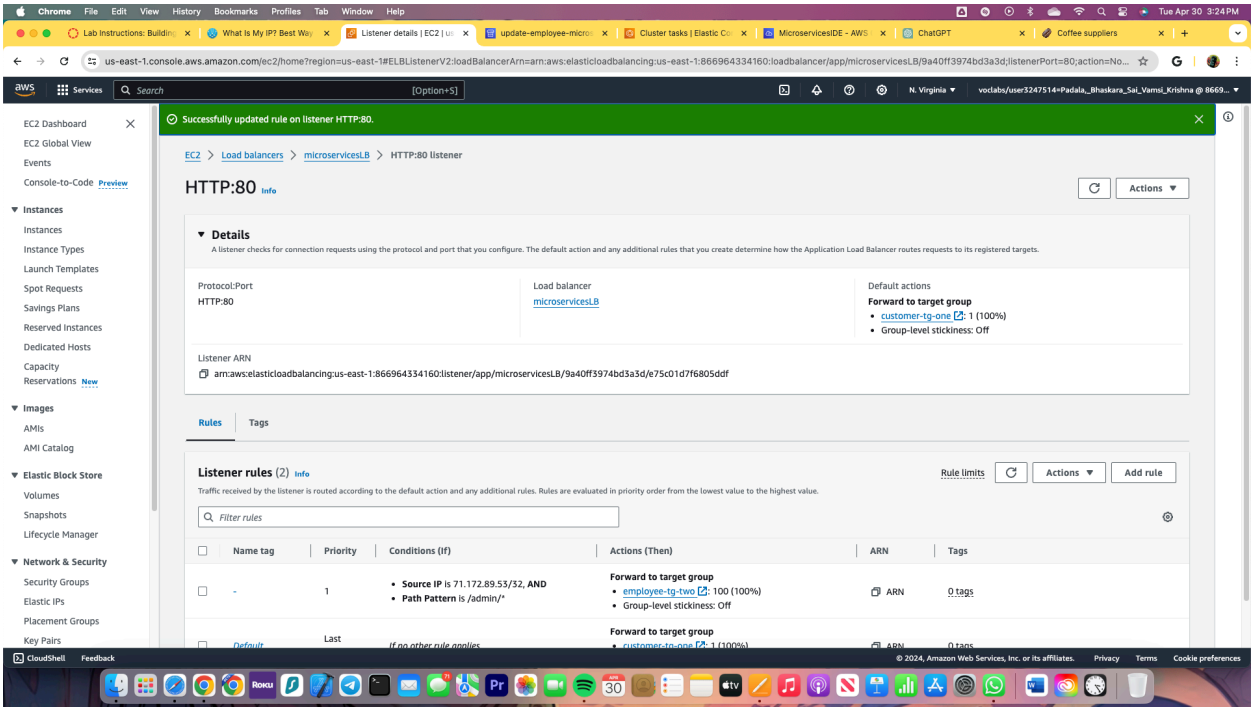

EC2 > Load balancers > microservicesLB > HTTP:8080 listener > Rule (priority 1) > Edit rule
Step 1
Define rule conditions
Step 2
Define rule actions
Step 3
Review changes
Define rule actions
These actions will be applied to requests matching the rule conditions.
Listener details: HTTP:8080
Actions
Action types
Routing actions
Forward to target groups
Redirect to URL
Return fixed response
Forward to target group
Choose a target group and specify routing weight or Create target group
Target group
Weight
Percent
employee-tg-one
Target type: IP, IPv4
HTTP
1
0-999
100%
Add target group
You can add up to 4 more target groups.
Group-level stickiness
If a target group is sticky, requests routed to it remain in that target group for the duration of the session. Individual target stickiness is a configuration of the target group.
Turn on group-level stickiness
Cancel
Previous
Next


Review changes
Listener details: HTTP:8080
Rule details
Edit
Priority
1
Conditions (If)
If request matches all:
Path Pattern is /admin/*, AND
Source IP is 71.172.89.53/32
Actions (Then)
Forward to target group
employee-tg-one: 1 (100%)
Group-level stickiness: Off
Rule ARN
arn:aws:elasticloadbalancing:us-east-1:866964334160:listener-rule/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2/e45c8cda20f7832a
Cancel
Previous
Save changes


Successfully updated rule on listener HTTP:8080.
HTTP:8080 Info
Actions
Details
Protocol:Port
HTTP:8080
Load balancer
microservicesLB
Default actions
Forward to target group
customer-tg-one: 1 (100%)
Group-level stickiness: Off
Listener ARN
arn:aws:elasticloadbalancing:us-east-1:866964334160:listener/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2
Rules
Tags
Listener rules (2) Info
Rule limits
Actions
Add rule
Filter rules
Name tag
Priority
Conditions (If)
Actions (Then)
ARN
Tags
Source IP is 71.172.89.53/32, AND
Path Pattern is /admin/*
Forward to target group
employee-tg-one: 1 (100%)
Group-level stickiness: Off
0 tags


Add condition
Rule limits
Rule condition types
Route traffic based on the condition type of each request. Each rule can include one of each of the following conditions: host-header, path, http-request-method and source-ip. Each rule can include one or more of each of the following conditions: http-header and query-string.
Source IP
Source IP
Define the source IP address in CIDR format.
is
71.172.89.53/32
Both IPv4 and IPv6 CIDRs are allowed. Wildcards are not supported.
Add new value
You can add up to 3 more condition values for this rule.
Cancel
Confirm



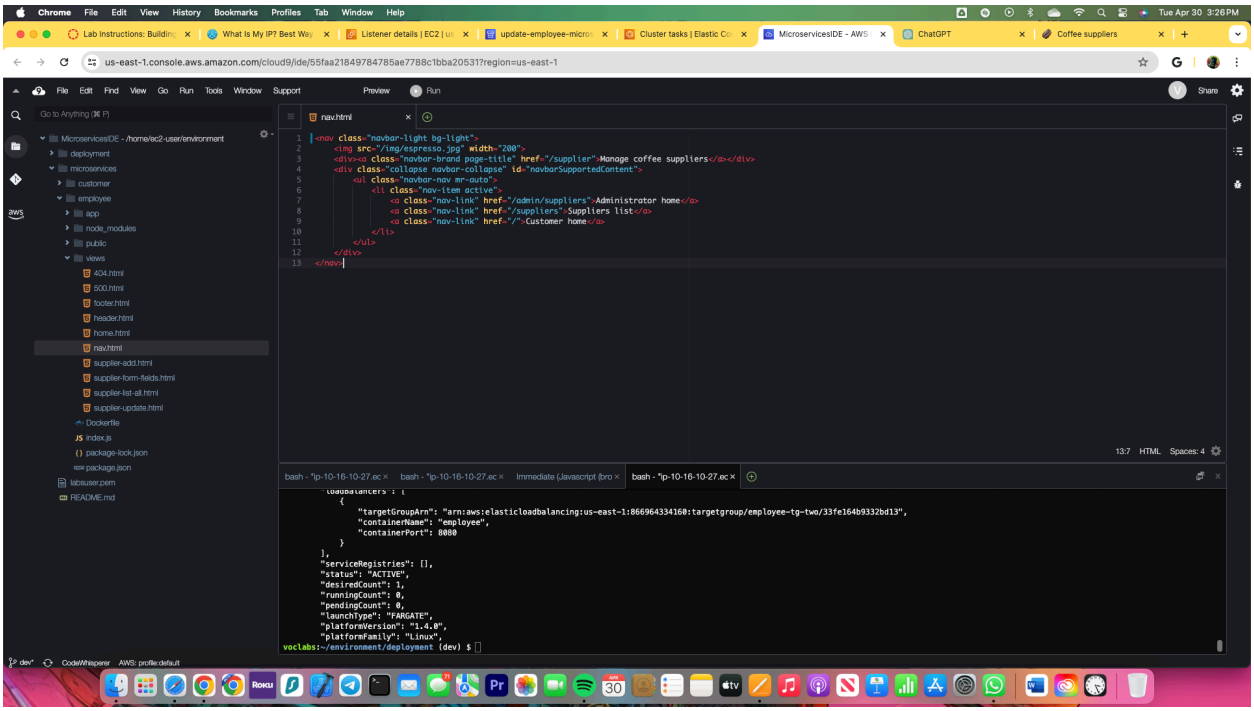


Task 9.2: Adjust the UI for the employee microservice and push the updated image to Amazon ECR

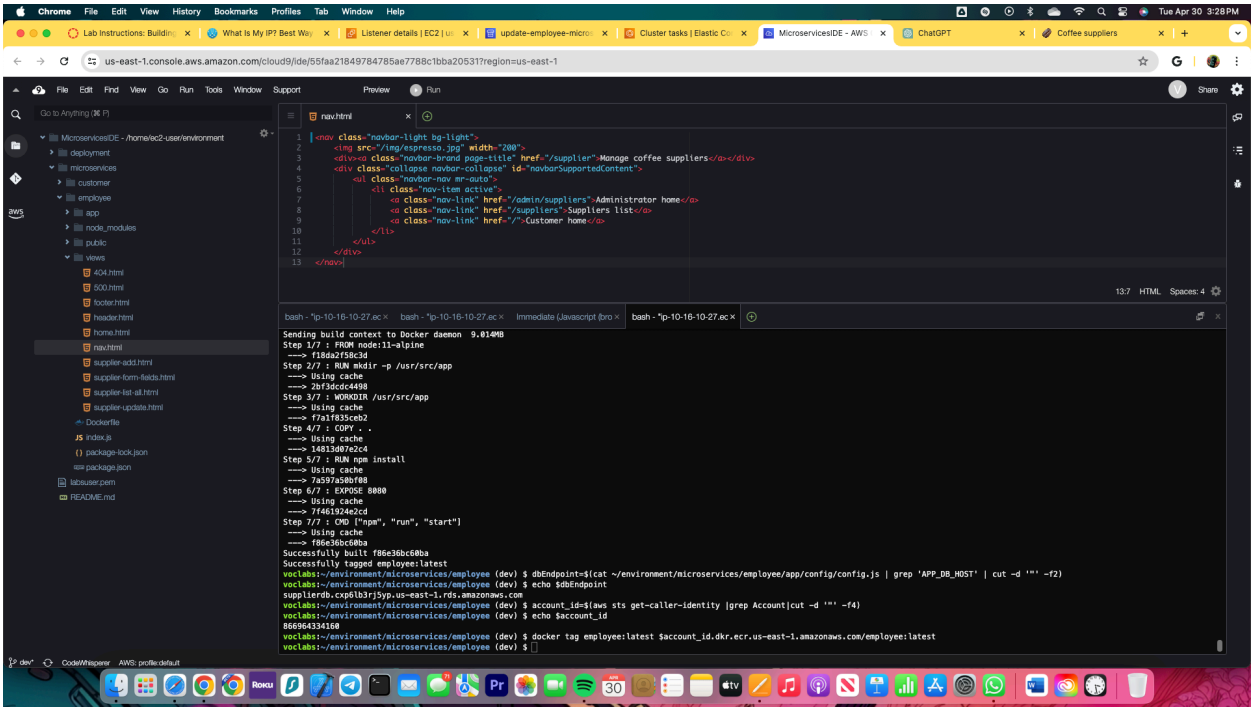
In this task, we will make modifications to the deployed microservices.

Open the file employee/views/nav.html.

Replace navbar-dark bg-dark with navbar-light bg-light on line 1.



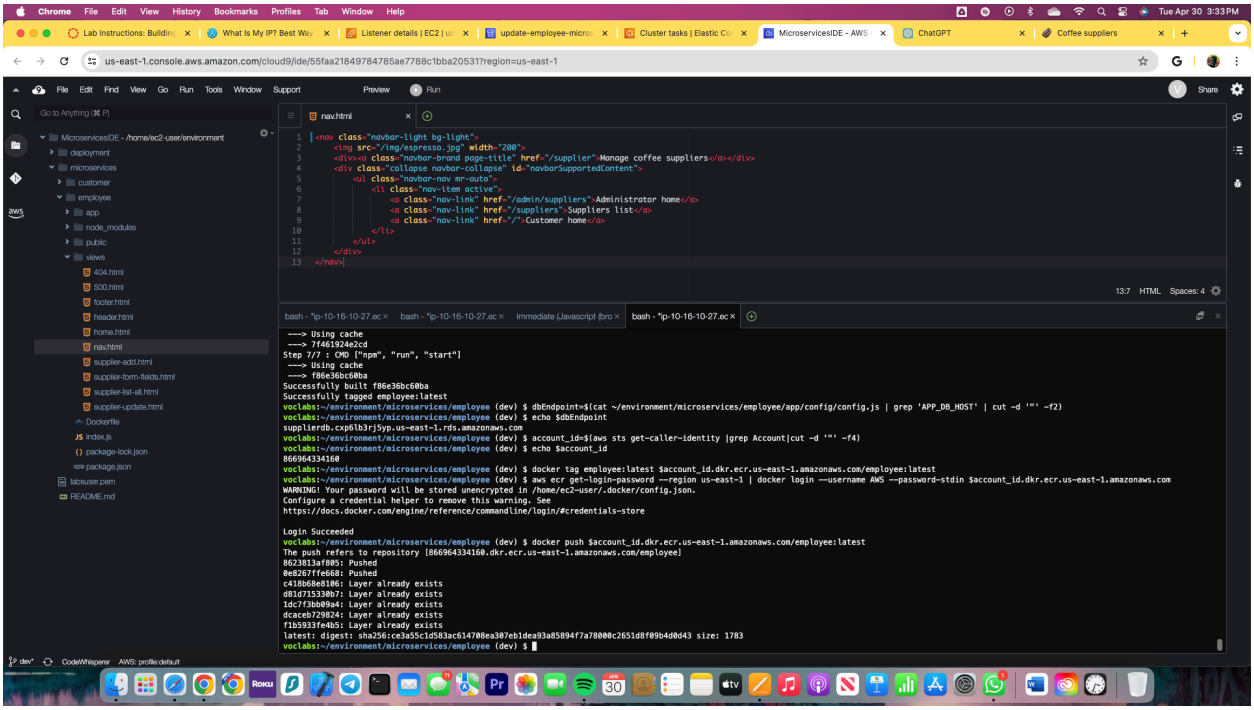
Save the updated file.



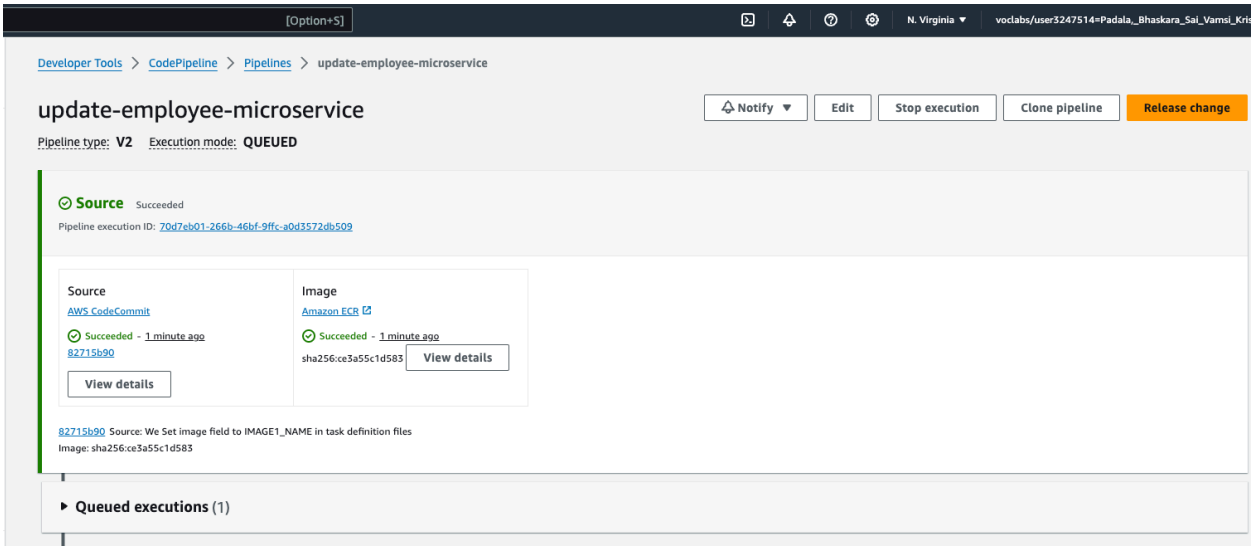
To trigger the update-employee-microservice pipeline, push an updated image to Amazon ECR.

In the CodePipeline console, access the details page for the update-employee-microservice pipeline and keep it open.

To upload the new Docker image for the employee microservice to Amazon ECR, execute the following commands in your AWS Cloud9 IDE:

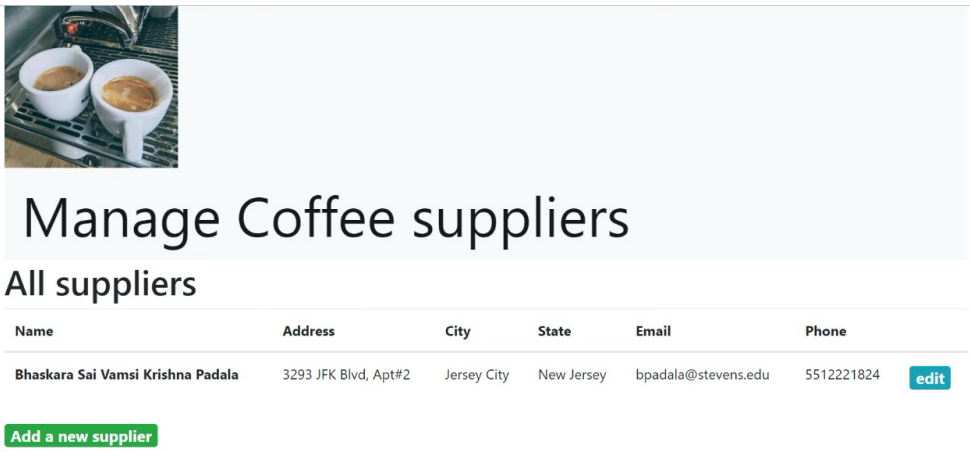


Task 9.3: Confirm that the employee pipeline ran and the microservice was updated



Observe that the pipeline was triggered and executed when a new Docker image was uploaded to Amazon ECR. Please be aware that it may take a minute or two for the pipeline to detect the updated Docker image before it is invoked.

Task 9.4: Test access to the employee microservice



Test access to the employee microservice pages by navigating to http://<alb-endpoint>/admin/suppliers and http://<alb-endpoint>:8080/admin/suppliers from the same device you've been using for this project. Replace <alb-endpoint> with the DNS name of the microservicesLB load balancer.

Ensure that at least one of the two pages loads successfully.

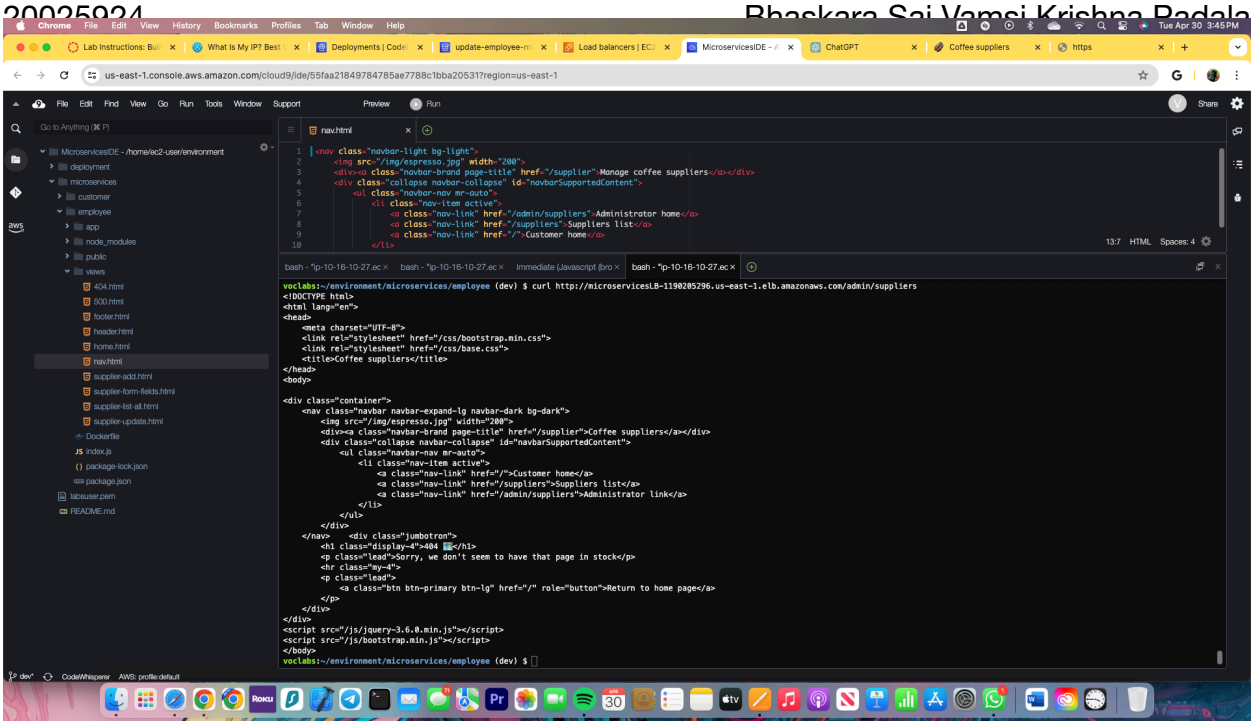
You'll notice that the banner with the page title now has a light color, reflecting the change made to the nav.html file. In contrast, pages hosted by the customer microservice retain the dark banner. This illustrates the flexibility of a microservices architecture, allowing independent modification of UI or features for each microservice without impacting others.

Test the accessibility of the employee microservice pages from a different device.

For instance, you can utilize your smartphone to connect via the cellular network, ensuring it is not on the same Wi-Fi network as your computer. This ensures that the device uses a distinct IP address to access the internet compared to your computer.

If the pages load, you should receive a 404 error, and the page content should display "Coffee suppliers" instead of "Manage coffee suppliers." This indicates that you are unable to successfully connect to the employee microservice from an alternate IP address.

Note: If you do not have access to another network, execute the following command in the AWS Cloud9 terminal: curl http://<alb-endpoint>/admin/suppliers. The source IP address of the AWS Cloud9 instance differs from that of your browser. The response should contain Sorry, we don't seem to have that page in stock .

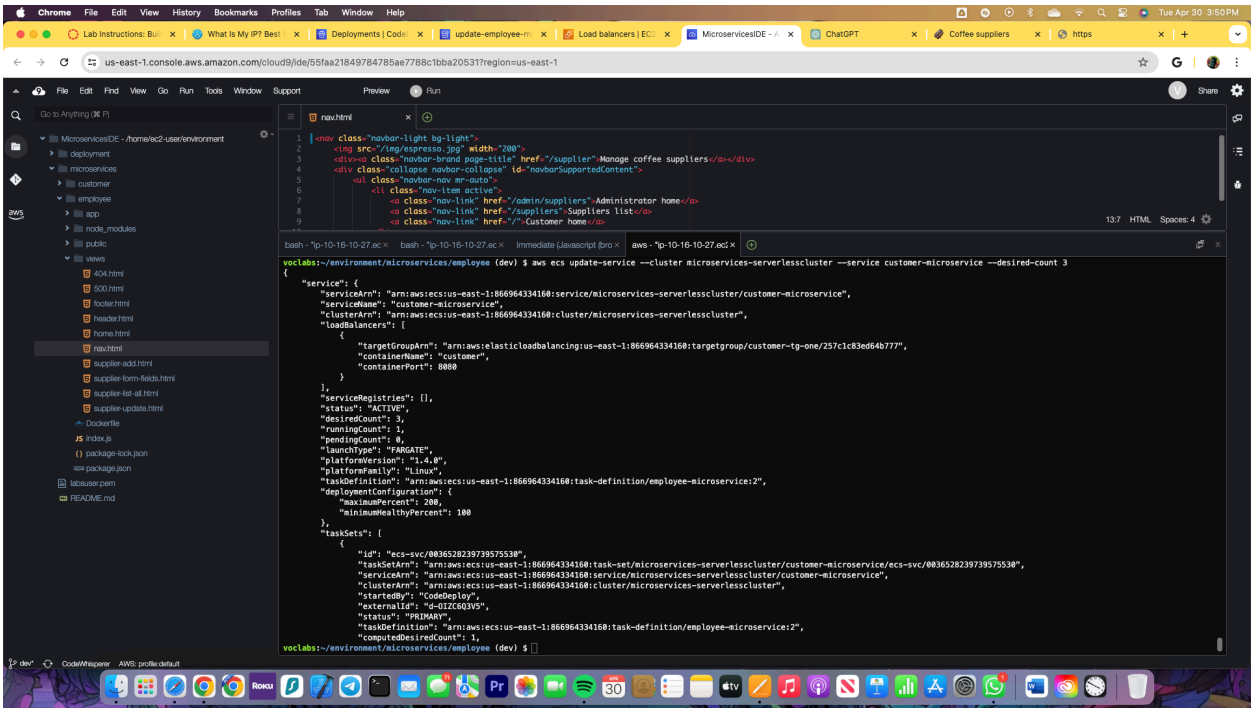


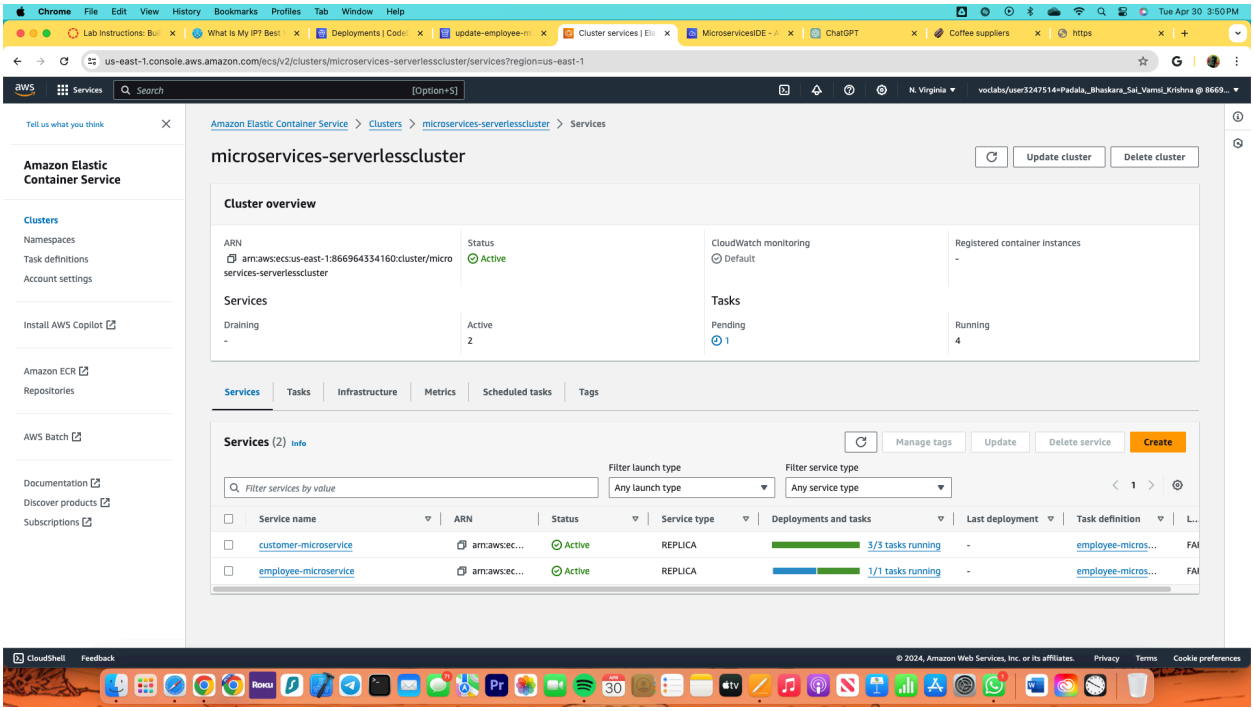
This validates that the updated rules on the load balancer listener are functioning as intended.

Task 9.5: Scale the customer microservice

In this task, we will increase the number of containers running to support the customer microservice. This adjustment can be made without triggering the update-customer-microservice pipeline.

To update the customer service in Amazon ECS, execute the following command:





In this scenario, the customer-microservice currently displays 1/3 tasks running because the desired count has been increased from 1 to 3. However, the two additional containers are still in the process of starting up. If you observe the Tasks tab and wait patiently, you will eventually notice that three containers are running to support the customer microservice. This highlights the capability to independently scale microservices as needed.

Analysis - Part - 2

CWID:20025924

Analyses of Part 2: Testing and Monolithic Application

In-depth examination and testing of a monolithic application running on a MonolithicAppServer instance are the main objectives of Phase 2. Important findings consist of:

- The program runs on port 80 over HTTP, which may be found using the command `sudo lsof -i :80}. This suggests that HTTPS—secure communication protocols—might not be being used by the application, which poses a security concern.
- The `ps -ef | grep node} command was used to examine processes associated with the application. This revealed that the `ubuntu} user is running the application, indicating that Node.js environment has to be explicitly setup.
- The application's main functionality is included in the `index.js` file, which is deployed directly via a file.

This structure's strong coupling of all operations inside a single codebase may cause scalability problems. A MySQL RDS database manages the application's data, demonstrating the separation of data storage from application functionality while preserving a conventional architectural layout.

In order to verify database interactions, supplier data was included as part of the processes. Phase 2 activities were successfully completed when the program and database were shut down after the database entries were verified.

Step 3: Establishing a Code Repository and Development Environment

Phase 3 involved setting up an AWS Cloud9 environment to handle the source code for the application. Using SCP and the following command, the source code was safely moved from the EC2 instance to Cloud9:

Ubuntu@$appServerPrivIp:/home/ubuntu/resources/codebase_partner/* ~/environment/temp/ {{{bash scp -r -i ~/environment/labsuser.pem

The source code was then arranged to represent the different functionality of the customer and employee directories. Using the following Git instructions, a microservices repository on AWS CodeCommit was started to promote version control and cooperation:

{{{cd ~/environment/microservices in a shell
git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices git init git branch -m dev git add. git commit -m 'two unaltered copies of the application code'
git push origin dev ~{{

This configuration facilitates future code revisions and deployment techniques in addition to encouraging effective code management.

Phase 4: Microservices and Containerization Transition

Isolating employee and customer operations into distinct services was a necessary step in the switch from a monolithic design to microservices. Through this reorganization, each service's superfluous functionality may be eliminated, improving maintainability and scalability.

Docker was used to implement containerization, and Dockerfiles were created to provide the dependencies and runtime environment for each microservice. This stage established the foundation for a scalable and flexible architecture by verifying the microservices' functionality inside separate Docker environments.

Stage 5: Container Administration and Deployment Readiness

To provide safe and effective maintenance of container images, ECR repositories were established for the purpose of storing Docker images. To improve security, these repositories are controlled by intricate IAM roles and policies that manage access permissions.

Furthermore, files for task definition and application specification were generated in order to specify the parameters of operation for every microservice container. Important information that is necessary for successful deployments is specified in these files, including container names, images, environmental variables, and logging parameters.

Second Phase: Traffic Control and Load Distribution

Security groups and target groups were put up in tandem with an Application Load Balancer (ALB). This configuration maintains high availability and fault tolerance while enabling dependable access to the microservices through effective traffic allocation and strong security management.

Deployment Automation and Continuous Integration/Continuous Deployment (CI/CD) (Phase 7 and Phase 8)

For the microservices, ECS services were developed, enabling automatic administration and scaling in response to traffic demands. In order to reduce downtime and improve deployment dependability, AWS CodeDeploy and CodePipeline were set up to automate the deployment process using blue/green deployment methodologies.

However, the target group's health checks presented difficulties for the employee microservice's implementation, emphasizing the necessity for more research and the resolution of underlying problems.

In summary, these stages emphasize the shift towards a microservices architecture that is more durable and scalable, while also stressing the significance of strong continuous integration and deployment (CI/CD) procedures in contemporary software development and deployment environments.