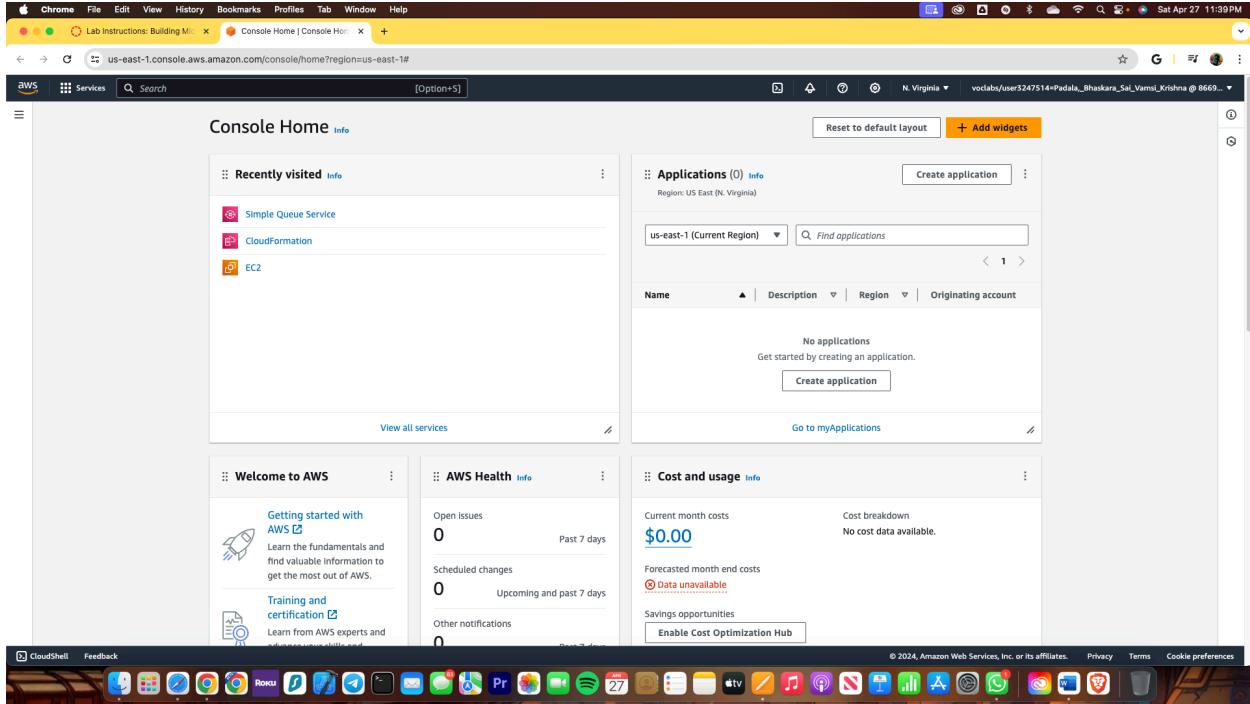


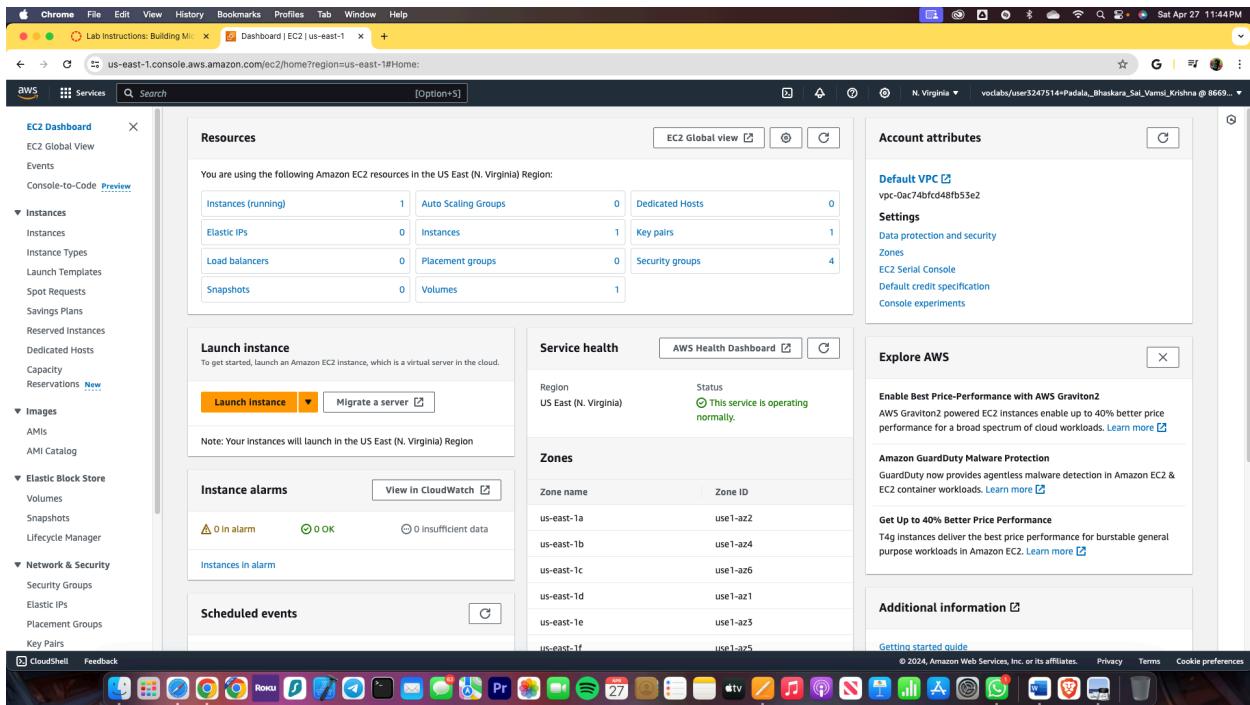
CS 524 - Final_Project

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

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various services like EC2 Dashboard, Events, Instances, Images, and Network & Security. The main area displays a table of instances. One instance is selected: "MonolithicAppServer" (i-0cdebb12ceb4bc220). The instance details pane on the right shows the Public IPv4 address as 44.206.241.29, which is highlighted and has a tooltip "Public IPv4 DNS copied". Other details include Instance ID (i-0cdebb12ceb4bc220), Instance state (Running), Instance type (t2.micro), and Private IP DNS name (ip-10-16-10-137.ec2.internal).

below.

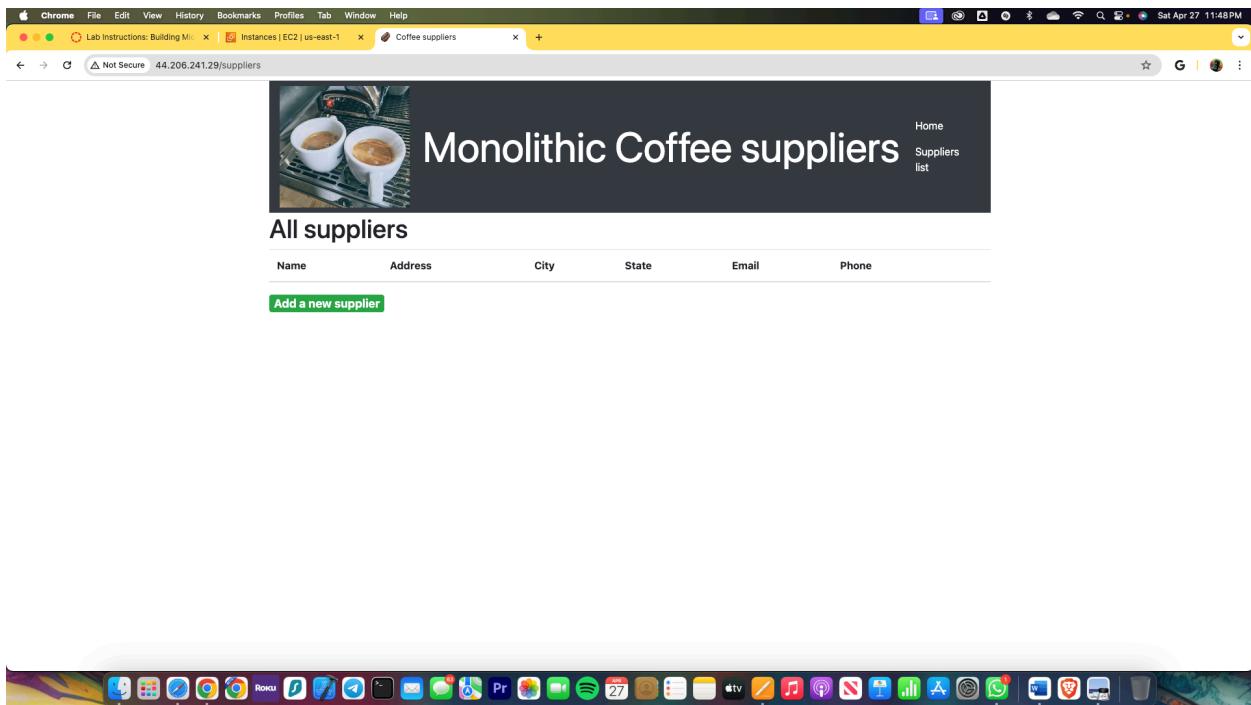
The screenshot shows a web browser displaying a monolithic web application titled "Monolithic Coffee suppliers". The URL in the address bar is 44.206.241.29. The page features a header with the title and a "Welcome" message. Below the header is a section for "List of suppliers" with a small image of two cups of coffee. The browser's toolbar and menu bar are visible at the top, and the Mac OS X dock is visible at the bottom.

Task 2.2: Test the monolithic web application

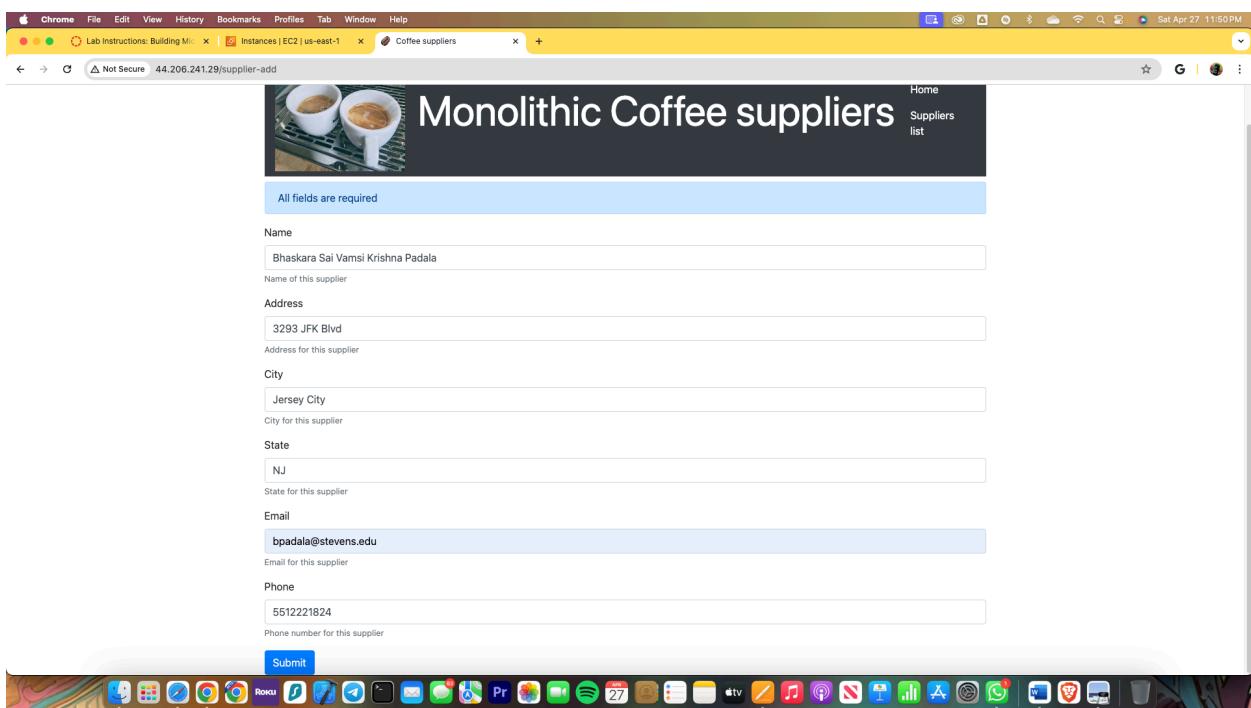
20025924

Bhaskara Sai Vamsi Krishna Padala

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.



20025924

Bhaskara Sai Vamsi Krishna Padala

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

All suppliers

Name	Address	City	State	Email	Phone
Bhaskara Sai Vamsi Krishna Padala	3293 JFK Blvd	Jersey City	NJ	bpadala@stevens.edu	5512221824

Add a new supplier

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

All fields are required

Name
Bhaskara Sai Vamsi Krishna Padala

Address
3293 JFK Blvd, Apt#2

City
Jersey City

State
New Jersey

Email
bpadala@stevens.edu

Phone
5512221824

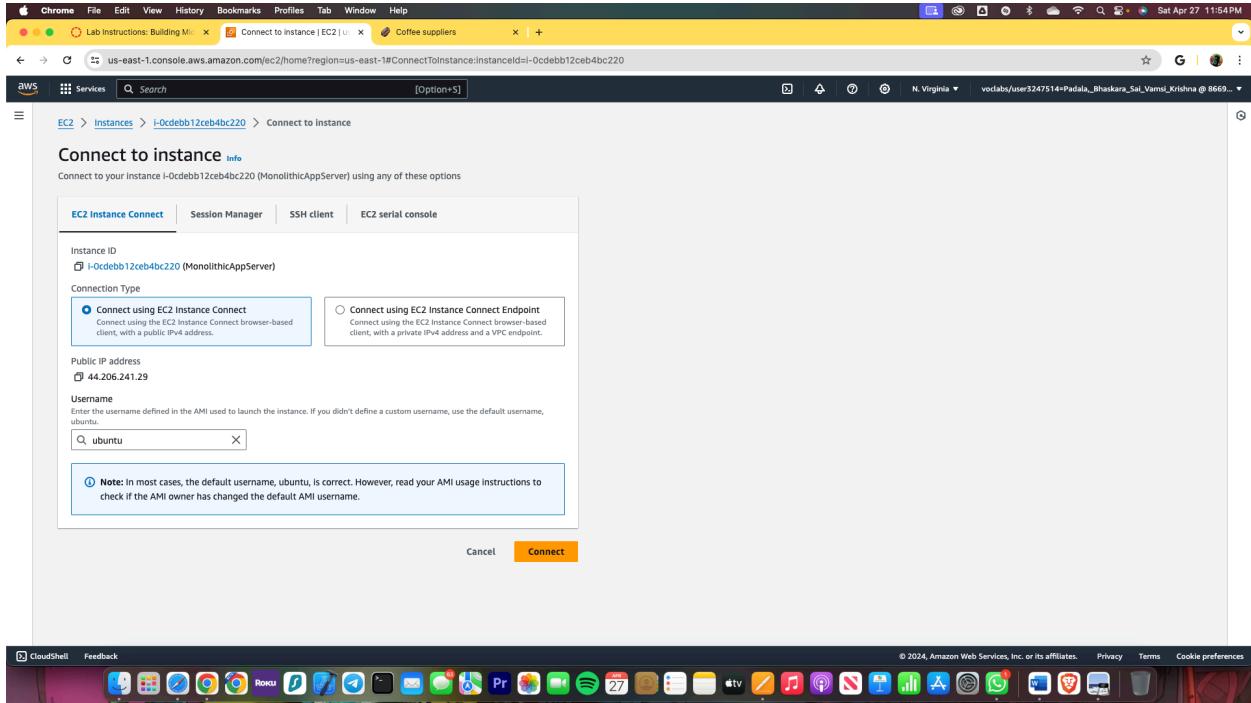
Submit

Delete this supplier

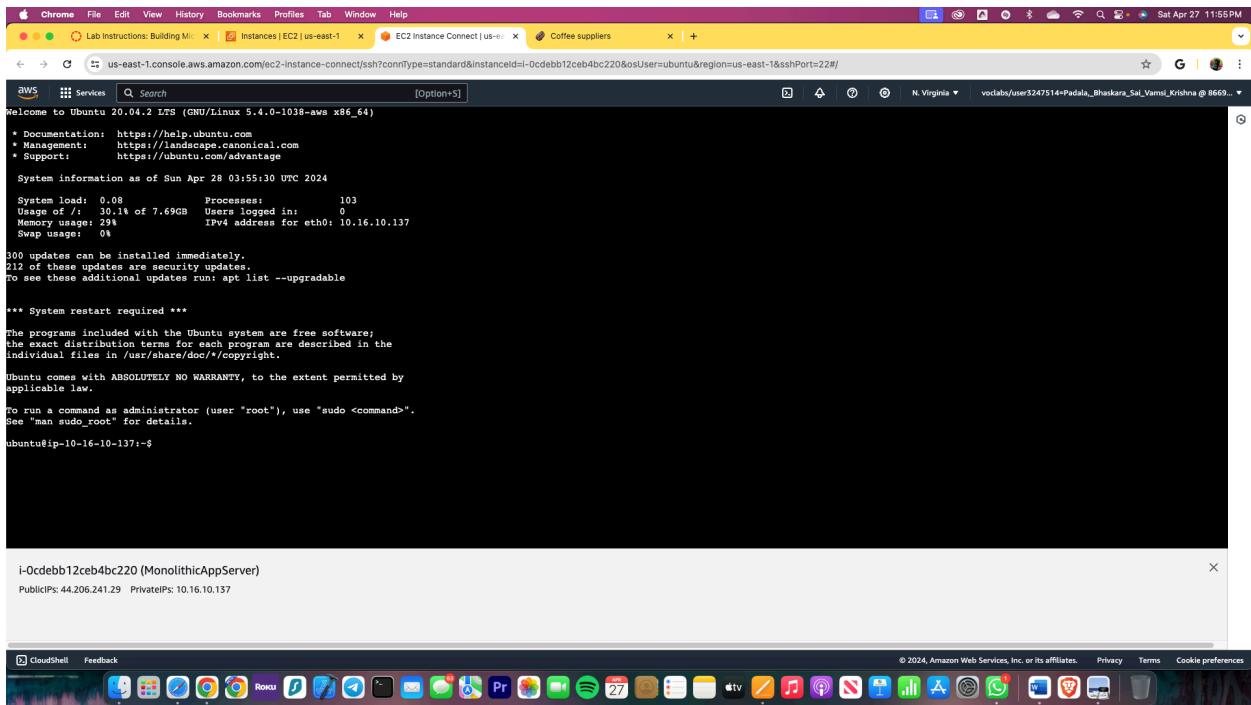
Now let's move on the TASK 2.3

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

20025924

Bhaskara Sai Vamsi Krishna Padala

A screenshot of a macOS desktop environment. At the top, there's a yellow menu bar with standard OS X icons like File, Edit, View, History, Bookmarks, Profiles, Tab, Window, and Help. Below the menu bar, the Dock contains various application icons. A central window is titled "Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1038-aws x86_64)". It displays system information, including memory usage (30.1% of 7.69GB), swap usage (0%), and network interfaces (eth0 at 10.16.10.137). It also lists 300 updates available. Below this, a command-line interface shows the user running "ls -l /" and "ps -ef | head -1; ps -ef | grep node". A small window titled "i-0cdebb12ceb4bc220 (MonolithicAppServer)" is open, showing its IP address (44.206.241.29) and private IP (10.16.10.137).

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`

A screenshot of a macOS desktop environment, identical to the previous one but with a different terminal command. The terminal window now shows the command `ps -ef | head -1; ps -ef | grep node` being run. The output of this command is visible in the terminal window, showing the process ID (PID) 15641 and the command "node index.js". The rest of the terminal window and the system status window are the same as the first screenshot.

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

`cd ~/resources/codebase_partner`

2025024

```
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1038-aws x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

System information as of Sun Apr 28 03:55:30 UTC 2024

System load: 0.08 Processes: 103
Usage of /: 30.1% of 7.69GB Users logged in: 0
Memory usage: 29% IPv4 address for eth0: 10.16.10.137
Swap usage: 0%

000 updates can be installed immediately.
212 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-137:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15090 root 18u IPv6 50627 0t0 TCP *:http (LISTEN)
ubuntu@ip-10-16-10-137:~$ ps -ef | head -1; ps -ef | grep node
UID PID PPID C STIME TTY TIME CMD
root 15090 1 0 03:41 ? 0:00:00 node index.js
ubuntu 15641 15615 0 03:41 pts/0 0:00:00 grep --color=auto node
ubuntu@ip-10-16-10-137:~$ cd ~/resources/codebase/partner
ubuntu@ip-10-16-10-137:~/resources/codebase/partner$ ls

i-Ocdebb12ceb4bc220 (MonolithicAppServer)

PublicIPs: 44.206.241.29 PrivateIPs: 10.16.10.137
```

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

```
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1038-aws x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

System information as of Sun Apr 28 03:55:30 UTC 2024

System load: 0.08 Processes: 103
Usage of /: 30.1% of 7.69GB Users logged in: 0
Memory usage: 29% IPv4 address for eth0: 10.16.10.137
Swap usage: 0%

000 updates can be installed immediately.
212 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-137:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15090 root 18u IPv6 50627 0t0 TCP *:http (LISTEN)
ubuntu@ip-10-16-10-137:~$ ps -ef | head -1; ps -ef | grep node
UID PID PPID C STIME TTY TIME CMD
root 15090 1 0 03:41 ? 0:00:00 node index.js
ubuntu 15641 15615 0 03:41 pts/0 0:00:00 grep --color=auto node
ubuntu@ip-10-16-10-137:~$ cd ~/resources/codebase/partner
ubuntu@ip-10-16-10-137:~/resources/codebase/partner$ ls
app index.js node_modules package-lock.json package.json public views
ubuntu@ip-10-16-10-137:~/resources/codebase/partner$ 

i-Ocdebb12ceb4bc220 (MonolithicAppServer)

PublicIPs: 44.206.241.29 PrivateIPs: 10.16.10.137
```

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

20025924

Bhaskara Sai Vamsi Krishna Padala

Introducing Aurora I/O-Optimized
Aurora's I/O-Optimized is a new cluster storage configuration that offers predictable pricing for all applications and improved price-performance, with up to 40% costs savings for I/O-Intensive applications.

Consider creating a Blue/Green Deployment to minimize downtime during upgrades
You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)

DB identifier	Status	Role	Engine	Region & AZ	Size	Recommendations	CPU	Current activity	Maintenance
supplierdb	Available	Instance	MySQL Community	us-east-1b	db.t3.micro		3.03%	1 Connections	none

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

DB identifier: supplierdb

Status: Available

Role: Instance

Engine: MySQL Community

Region & AZ: us-east-1b

Current activity: 1 Connections

Endpoint: supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com

Port: 3306

Availability Zone: us-east-1b

VPC: LabVPC (vpc-089ece87b4fc2f9bb)

Subnet group: c110323a260559816559001w866964334160-dbsubnetgroup-k9jqcpnufanz

Subnets: subnet-0d60eb650a27eca16, subnet-0f81de41a95dc5d31

Network type: IPv4

VPC security groups: DBSecurityGroup (sg-0ba27b1cb1ea4b45e) Active

Publicly accessible: No

Certificate authority: rds-ca-rsa2048-g1

Certificate authority date: May 25, 2061, 19:34 (UTC-04:00)

DB instance certificate expiration date: April 27, 2025, 23:36 (UTC-04:00)

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

```

System information as of Sun Apr 28 03:55:30 UTC 2024
System load: 0.08 Processes: 103
Usage of /: 30.1% of 7.69GB Users logged in: 0
Memory usage: 29% IPv4 address for eth0: 10.16.10.137
Swap usage: 0B

300 updates can be installed immediately.
212 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-137:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15090 root 1bu IPv6 50627 0t0 TCP *:http (LISTEN)
ubuntu@ip-10-16-10-137:~$ ps aux | head -1; ps aux | grep node
UID      PID    PPID  C STIME TTY          TIME CMD
root     15090     1  0 03:41 ?        0:00:00 node index.js
ubuntu   15641  15615  0 03:57 pts/0    0:00:00 grep --color=auto node
ubuntu@ip-10-16-10-137:~$ cd ~/resources/codibase_partner
ubuntu@ip-10-16-10-137:~/resources/codibase_partner$ nmap -Pn supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com
Nmap done: 1 IP address (1 host up) scanned in 8.16 seconds
ubuntu@ip-10-16-10-137:~/resources/codibase_partner$ 

i-Ocdebb12ceb4bc220 (MonolithicAppServer)
PublicIPs: 44.206.241.29 PrivateIPs: 10.16.10.137

```

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

```

System information as of Sun Apr 28 12:20 AM
System load: 0.08 Processes: 103
Usage of /: 30.1% of 7.69GB Users logged in: 0
Memory usage: 29% IPv4 address for eth0: 10.16.10.137
Swap usage: 0B

300 updates can be installed immediately.
212 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-137:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15090 root 1bu IPv6 50627 0t0 TCP *:http (LISTEN)
ubuntu@ip-10-16-10-137:~$ ps aux | head -1; ps aux | grep node
UID      PID    PPID  C STIME TTY          TIME CMD
root     15090     1  0 03:41 ?        0:00:00 node index.js
ubuntu   15641  15615  0 03:57 pts/0    0:00:00 grep --color=auto node
ubuntu@ip-10-16-10-137:~/resources/codibase_partner$ nmap -Pn supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-28 04:19 UTC
Nmap scan report for supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com (10.16.40.10)
Host is up (0.0006s latency).
Nmap done: 1 IP address (1 host up) scanned in 8.16 seconds
ubuntu@ip-10-16-10-137:~/resources/codibase_partner$ 

i-Ocdebb12ceb4bc220 (MonolithicAppServer)
PublicIPs: 44.206.241.29 PrivateIPs: 10.16.10.137

```

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

20025924

Rhaskara Sai Vamsi Krishna Padala

```
Swap usage: 0%
300 updates can be installed immediately.
212 of these updates are security updates.
To see these additional updates run: apt list --upgradable

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-137:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15090 root 18u IPv6 50627 0t0 TCP *:http (LISTEN)
ubuntu@ip-10-16-10-137:~$ ps -ef | grep node
node 15090 1 0 0:00:00 node index.js
ubuntu 15615 1 0 0:00:00 node index.js
ubuntu@ip-10-16-10-137:~$ cd ~/resources/codebase_partner
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ nmap -Pn supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-28 04:19 UTC
Nmap scan report for supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com (10.16.40.10)
Host is up (0.0006s latency).
rDNS record for 10.16.40.10: ip-10-16-40-10.ec2.internal
Not shown: 999 filtered ports
PORT      STATE SERVICE
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 8.16 seconds
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ mysql -h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p
i-Ocdebb12ceb4bc220 (MonolithicAppServer)
```

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

```
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.16.10.137' (using password: YES)
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ mysql -h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.16.10.137' (using password: YES)
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ mysql -h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.16.10.137' (using password: YES)
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ nmap -Pn supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com (10.16.40.10)
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-28 04:30 UTC
Nmap scan report for supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com (10.16.40.10)
Host is up (0.0006s latency).
rDNS record for 10.16.40.10: ip-10-16-40-10.ec2.internal
Not shown: 999 filtered ports
PORT      STATE SERVICE
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 6.55 seconds
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ mysql -h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.16.10.137' (using password: YES)
ubuntu@ip-10-16-10-137:~/resources/codebase_partner$ mysql -h supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 49
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

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

```

Map done: 1 IP address (1 host up) scanned in 6.55 seconds
ubuntu@ip-10-16-10-137:~/resources/database_partner$ mysql -h supplierdb.cxp6lb3rj5vp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.16.10.137' (using password: YES)
ubuntu@ip-10-16-10-137:~/resources/database_partner$ mysql -h supplierdb.cxp6lb3rj5vp.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 49
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type 'c' to clear the current input statement.

mysql> COFFEE
      ->
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'COFFEE' at line 1
mysql> USE COFFEE;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM SUPPLIERS;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'COFFEE.SUPPLIERS' doesn't exist
mysql> SELECT * FROM SUPPLIERS
      -> Describe Suppliers;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Suppliers' at line 2
mysql> SELECT * from suppliers;
+----+-----+-----+-----+-----+-----+
| id | name | address | city | state | email | phone |
+----+-----+-----+-----+-----+-----+
| 1  | Bhaskara Sai Vamsi Krishna Padala | 3293 JFK Blvd, Apt#2 | Jersey City | New Jersey | bpadala@stevens.edu | 5512221824 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

i-0cdebb12ceb4bc220 (MonolithicAppServer)
PublicIPs: 44.206.241.29 PrivateIPs: 10.16.10.137

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

AWS Cloud9

A cloud IDE for writing, running and debugging code

AWS Cloud9 allows you to write, run and debug your code with just a browser. With AWS Cloud9, you have immediate access to a rich code editor, integrated debugger and built-in terminal with pre-configured AWS CLI. You can get started in minutes and no longer have to spend the time to install local applications or configure your development machine.

How it works

Create an AWS Cloud9 development environment on a new Amazon EC2 instance or connect it to your own Linux server through SSH. Once you've created an AWS Cloud9 environment, you will have immediate access to a rich code editor, integrated debugger and built-in terminal with pre-configured AWS CLI – all within your browser.

Using the AWS Cloud9 dashboard, you can create and switch between many different AWS Cloud9 environments, each one containing the customized tools, runtimes and files for a specific project.

[Find out more](#)

Benefits and features

Code with just a browser	Code together in real time
AWS Cloud9 allows you to write, run, and debug applications with just a browser and without the need to install or maintain a Desktop IDE.	AWS Cloud9 makes collaborating on code easy. You can share your development environment with your team in just a few clicks and pair program together.

Getting started

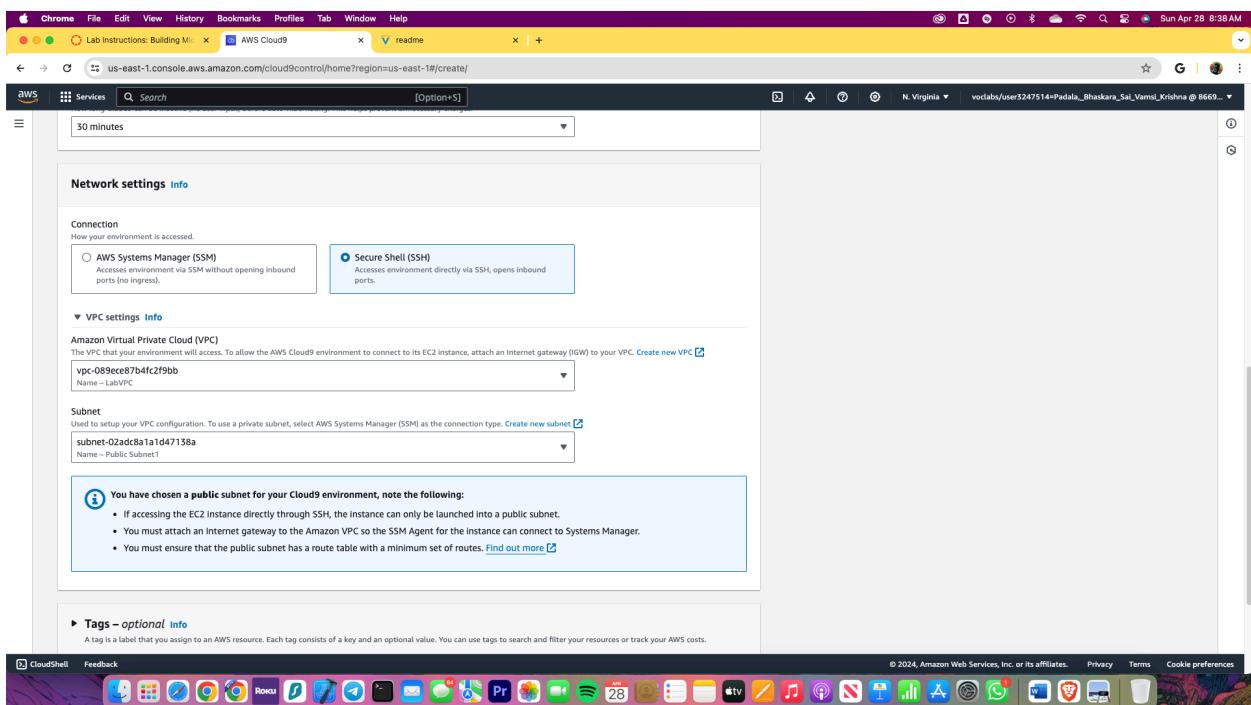
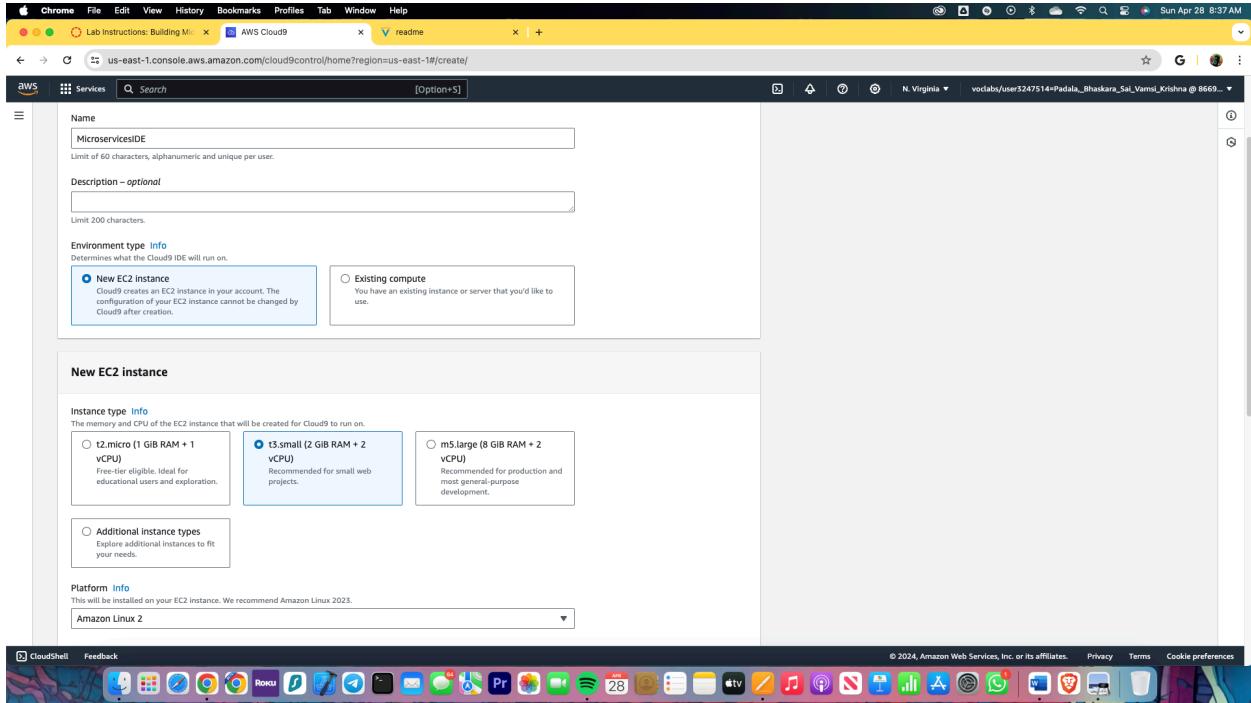
- [Before you start \(2-min read\)](#)
- [Create an environment \(2-min read\)](#)
- [Working with environments \(15-min read\)](#)
- [Working with the IDE \(10-min read\)](#)
- [Working with AWS Lambda \(5-min read\)](#)

More resources

- [FAQs](#)
- [Forum](#)

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 configurations we get the following screen.

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS Cloud9 interface. The left sidebar has sections for 'My environments', 'Shared with me', and 'All account environments'. The main content area is titled 'Environments (1)' and shows a single environment named 'MicroservicesIDE'. The table columns are Name, Cloud9 IDE, Environment type, Connection, Permission, and Owner ARN. The environment details are: Name - MicroservicesIDE, Cloud9 IDE - Open, Environment type - EC2 Instance, Connection - Secure Shell (SSH), Permission - Owner, and Owner ARN - arn:aws:sts::866964334160:assumed-role/voclabs/user5247514=Padala,_Bhaskara,_Sai,_Vamsi,_Krishna. There are buttons for 'Delete', 'View details', 'Open in Cloud9', and 'Create environment'.

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

The screenshot shows the AWS EC2 Instances page. The left sidebar includes sections like EC2 Dashboard, EC2 Global View, Events, Console-to-Code, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity, Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, and Key Pairs. The main content area displays two instances: 'MonolithicApp...' (Instance ID: i-0cdebb12ceb4bc220, State: Running, Type: t2.micro) and 'aws-cloud9-MI...' (Instance ID: i-069a81ebb4995de17, State: Running, Type: t3.small). Both instances have 2/2 checks passed. The table columns include Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, Public IPv4 DNS, Public IPv4 IP, and Elastic IP. A 'Select an instance' dropdown is visible at the bottom.

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.

AWS CLI: [Show](#)

Cloud Labs

Remaining session time: 11:54:52(715 minutes)

Session started at: 2024-04-27T20:33:49-0700

Session to end at: 2024-04-28T17:52:37-0700

Accumulated lab time: 09:23:00 (563 minutes)

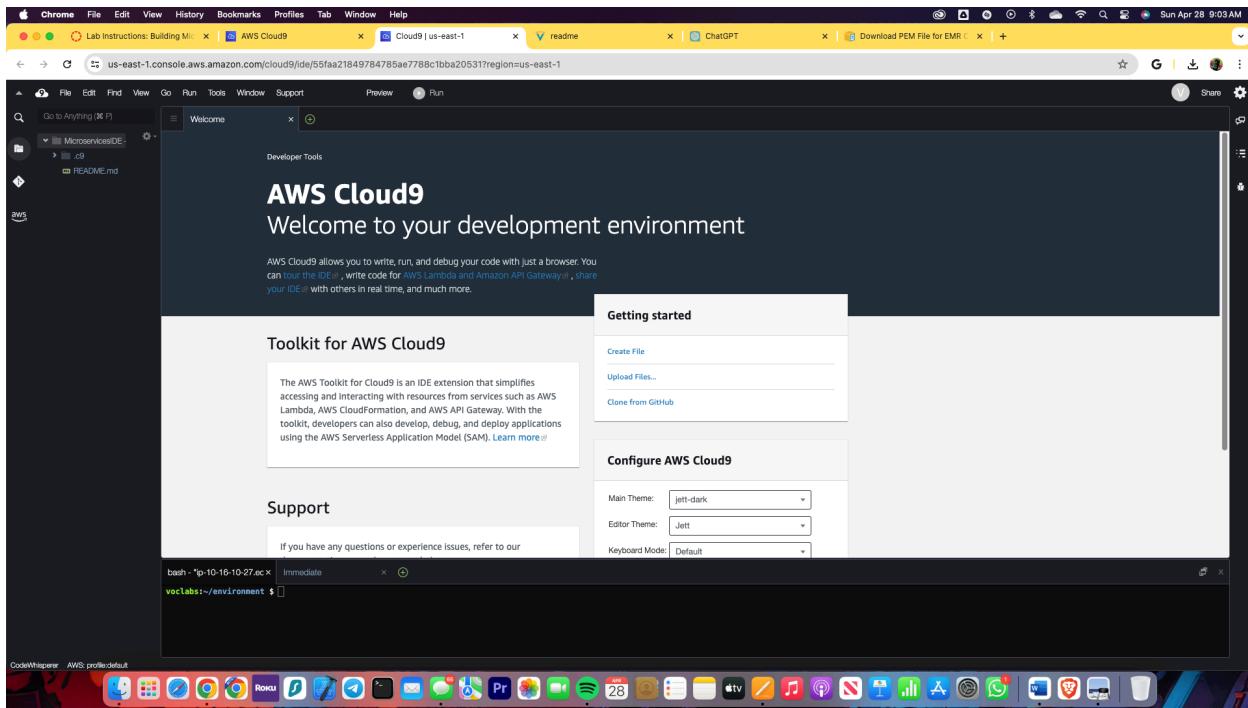
ips -- public:44.206.241.29, private:10.16.10.137

SSH key [Show](#) [Download PEM](#) [Download PPK](#)AWS SSO [Download URL](#)

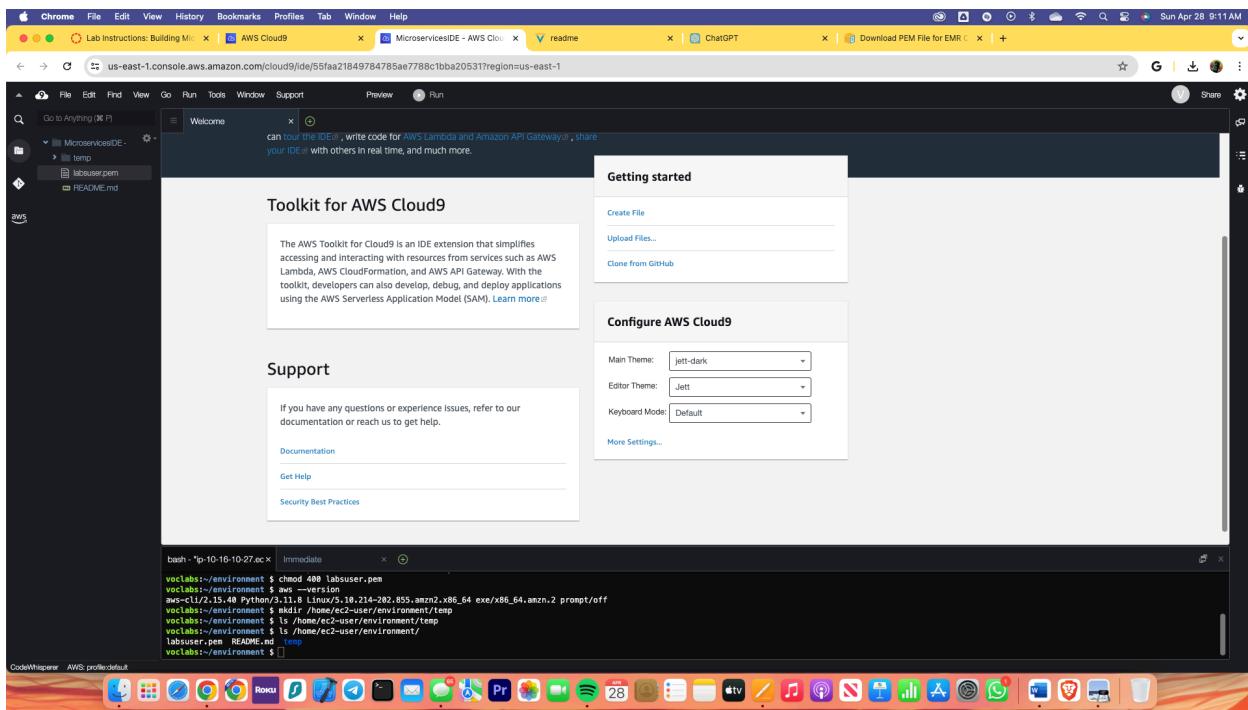
RDSEndpoint supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com

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

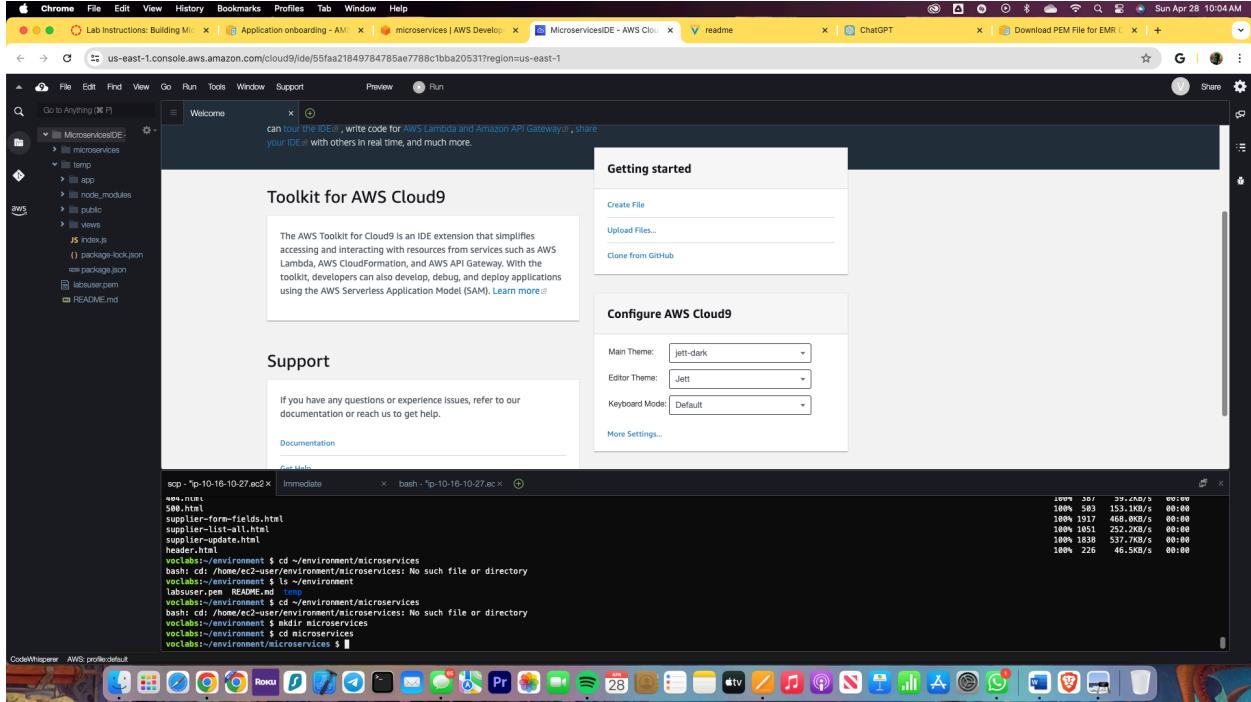
Now after copying and running the following command below, we get the screen showing that we downloaded all these files.

```
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.

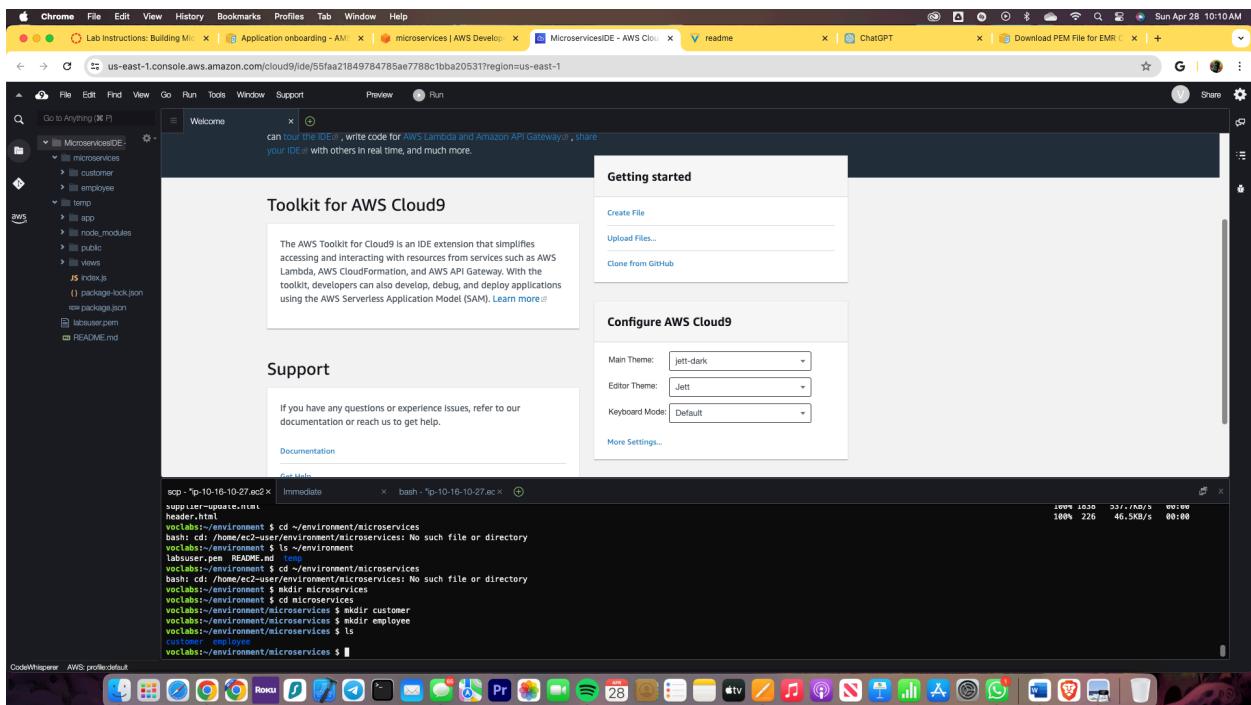
Now we created micro services directory in command prompt.



The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a file tree for a project named 'MicroservicesIDE'. In the center, a 'Welcome' panel provides information about the toolkit and links to 'Getting started', 'Support', and 'Documentation'. On the right, a terminal window titled 'sop - ip-10-16-10-27.ec2.x' is open, showing the command history for creating directories:

```
sop - ip-10-16-10-27.ec2.x bash - ip-10-16-10-27.ec2.x
new.html
500.html
supplier-form-fields.html
supplier-list-all.html
supplier-update.html
header.html
voclabs:/environment $ cd ~/environment/microservices
bash: cd: /home/ec2-user/environment/microservices: No such file or directory
voclabs:/environment $ ls -l
total 0
voclabs:/environment $ labuser.pem README.md temp
voclabs:/environment $ cd ..
voclabs:/environment $ ls
microservices
voclabs:/environment $ mkdir microservices
voclabs:/environment $ cd microservices
voclabs:/environment/microservices $
```

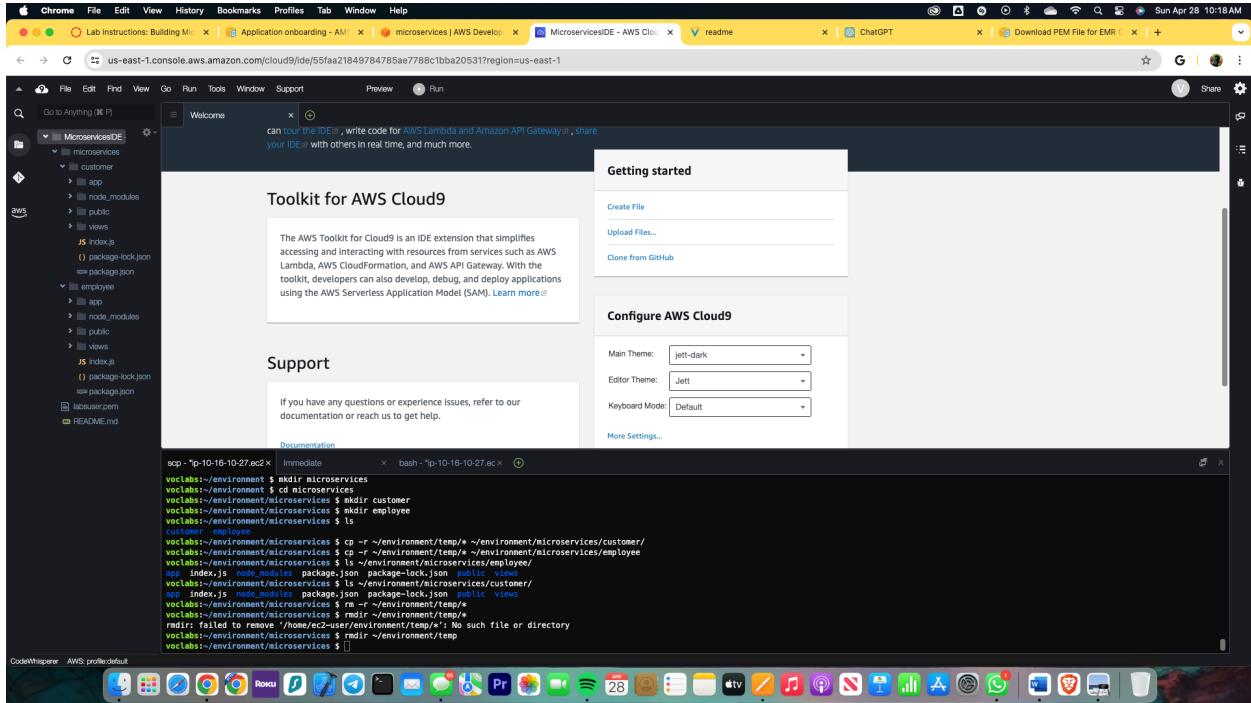
Now we've created the customer and employee directories in the micro services directory.



This screenshot is identical to the previous one, showing the AWS Cloud9 IDE interface. The terminal window now shows the completed directory structure:

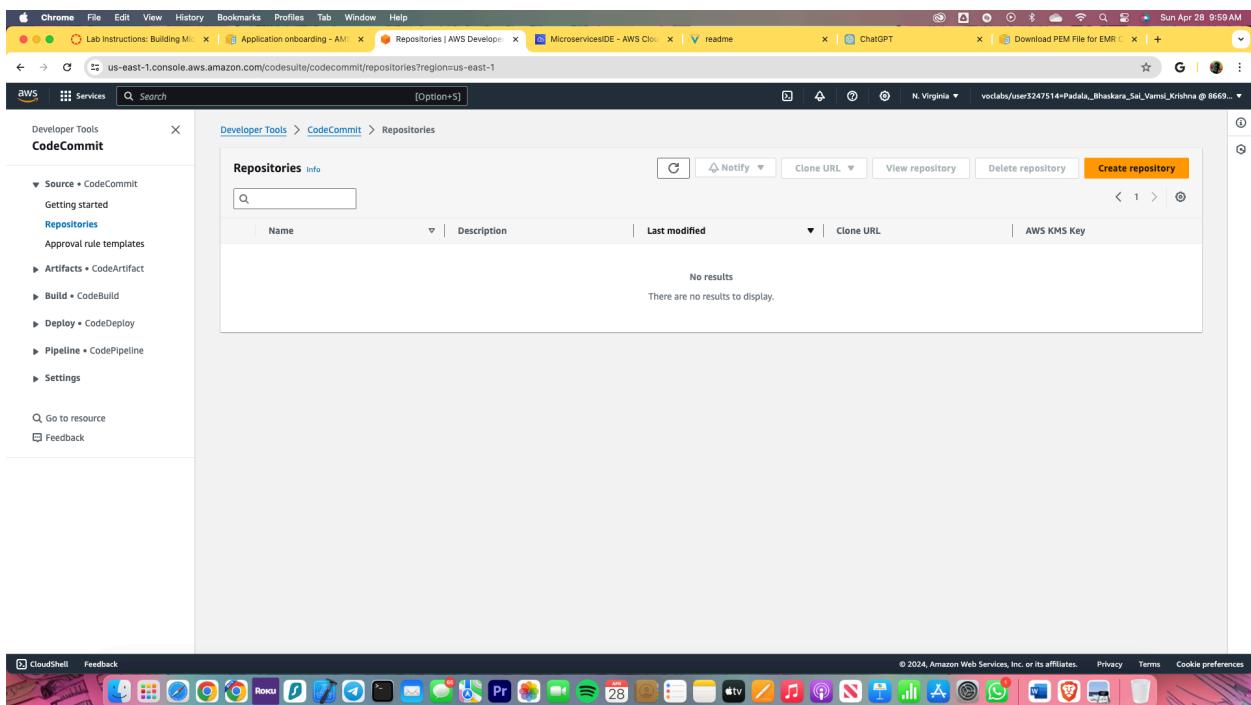
```
sop - ip-10-16-10-27.ec2.x bash - ip-10-16-10-27.ec2.x
header.html
voclabs:/environment $ cd ~/environment/microservices
bash: cd: /home/ec2-user/environment/microservices: No such file or directory
voclabs:/environment $ ls -l
total 0
voclabs:/environment $ labuser.pem README.md temp
voclabs:/environment $ cd ..
voclabs:/environment $ ls
customer employee
voclabs:/environment $ cd microservices
voclabs:/environment/microservices $ mkdir customer
voclabs:/environment/microservices $ mkdir employee
voclabs:/environment/microservices $ ls
customer employee
voclabs:/environment/microservices $
```

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.



20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS CodeCommit interface. On the left, a sidebar lists various repository categories like Code, Pull requests, Commits, Branches, Git tags, Settings, Approval rule templates, Artifacts, Build, Deploy, Pipeline, and Settings. The main content area displays a success message: "Success Repository successfully created". Below this, the repository name "microservices" is shown, along with a "Clone URL" button. The "Connection steps" section provides instructions for connecting via HTTPS, SSH, or HTTPS (GRC), noting that HTTPS is the only supported method for sign-in types. A note states: "⚠ You are signed in using federated access or temporary credentials. The only supported connection method for these sign-in types is to use the credential manager included with the AWS CLI, as documented below. To configure a connection using SSH or Git credentials over HTTPS, sign in as an IAM user." The "Step 2: Set up the AWS CLI Credential Helper" section explains how to set up the AWS CLI for root access, federated access, and temporary credentials. An "Additional details" section links to more documentation.

The screenshot shows the AWS Cloud9 IDE interface. The left sidebar shows the file structure of the "microservices" repository, including files like index.js, package-lock.json, package.json, and README.md. The main workspace shows the "Welcome" screen of the Toolkit for AWS Cloud9, which is an IDE extension for AWS Lambda and Amazon API Gateway. It includes sections for "Getting started" (Create File, Upload Files..., Clone from GitHub) and a terminal window. The terminal window displays the following commands and output:

```
git -l@10-16-10-27.2e2.X Immediate x bash -l@10-16-10-27.2e2.X
voclab@~/environment/microservices $ mkdir customer
voclab@~/environment/microservices $ mkdir employee
voclab@~/environment/microservices $ ls
customer employee
voclab@~/environment/microservices $ cp -r ~/environment/temp/* ~/environment/microservices/customer/
voclab@~/environment/microservices $ cp -r ~/environment/temp/* ~/environment/microservices/employee/
voclab@~/environment/microservices $ ls ~/environment/microservices/
app index.js node_modules package-lock.json public views
voclab@~/environment/microservices $ ls ~/environment/microservices/customer/
customer index.js node_modules package-lock.json public views
voclab@~/environment/microservices $ cp -r ~/environment/temp/*
voclab@~/environment/microservices $ rm ~/environment/temp/*
radir: failed to remove '/home/ec2-user/environment/temp/*': No such file or directory
voclab@~/environment/microservices $ ls ~/environment/temp/*
voclab@~/environment/microservices $ cd ~/environment/microservices
voclab@~/environment/microservices $ git init
hint: Using 'master' as the name for the initial branch. This default branch name 'master' is subject to change. To configure the initial branch name to use in all hints of your new repositories, which will suppress this warning, call:
hint:
hint: git config --global init.defaultBranch <name>
hint: Name commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint: git branch --move <name>
Initialized empty Git repository in /home/ec2-user/environment/microservices/.git/
voclab@~/environment/microservices (master) $ git branch -m dev
voclab@~/environment/microservices (dev) $ git add .
voclab@~/environment/microservices (dev) $ git commit -m 'two unmodified copies of the application code'
```

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'
```

```

git -> p-10-16-10-27.ec2 > Immediate > bash -> p-10-16-10-27.ec2 >
create mode 10644 employee/node_modules/vary/LICENSE
create mode 10644 employee/node_modules/vary/index.js
create mode 10644 employee/node_modules/vary/README.md
create mode 10644 employee/node_modules/vary/package.json
create mode 10644 employee/node_modules/yallist/README.md
create mode 10644 employee/node_modules/yallist/iterator.js
create mode 10644 employee/node_modules/yallist/list.js
create mode 10644 employee/node_modules/yallist/package.json
create mode 10644 employee/node_modules/yallist/yallist.js
create mode 10644 employee/package-lock.json
create mode 10644 employee/package.json
create mode 10644 employee/public/css/base.css
create mode 10644 employee/public/css/bootstrap.min.css
create mode 10644 employee/public/css/bootstrap.min.css.map
create mode 10644 employee/public/img/espresso.jpg
create mode 10644 employee/public/img/favicon.ico
create mode 10644 employee/public/js/bootstrap.min.js
create mode 10644 employee/public/js/bootstrap.min.js.map
create mode 10644 employee/public/js/jquery-3.6.0.min.js
create mode 10644 employee/public/views/about.html
create mode 10644 employee/public/views/footer.html
create mode 10644 employee/public/views/header.html
create mode 10644 employee/views/nav.html
create mode 10644 employee/views/supplier-add.html
create mode 10644 employee/views/supplier-form-fields.html
create mode 10644 employee/views/supplier-list-all.html
create mode 10644 employee/views/supplier-update.html
voclabs:-~/environment/microservices (dev) $

```

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

```

```

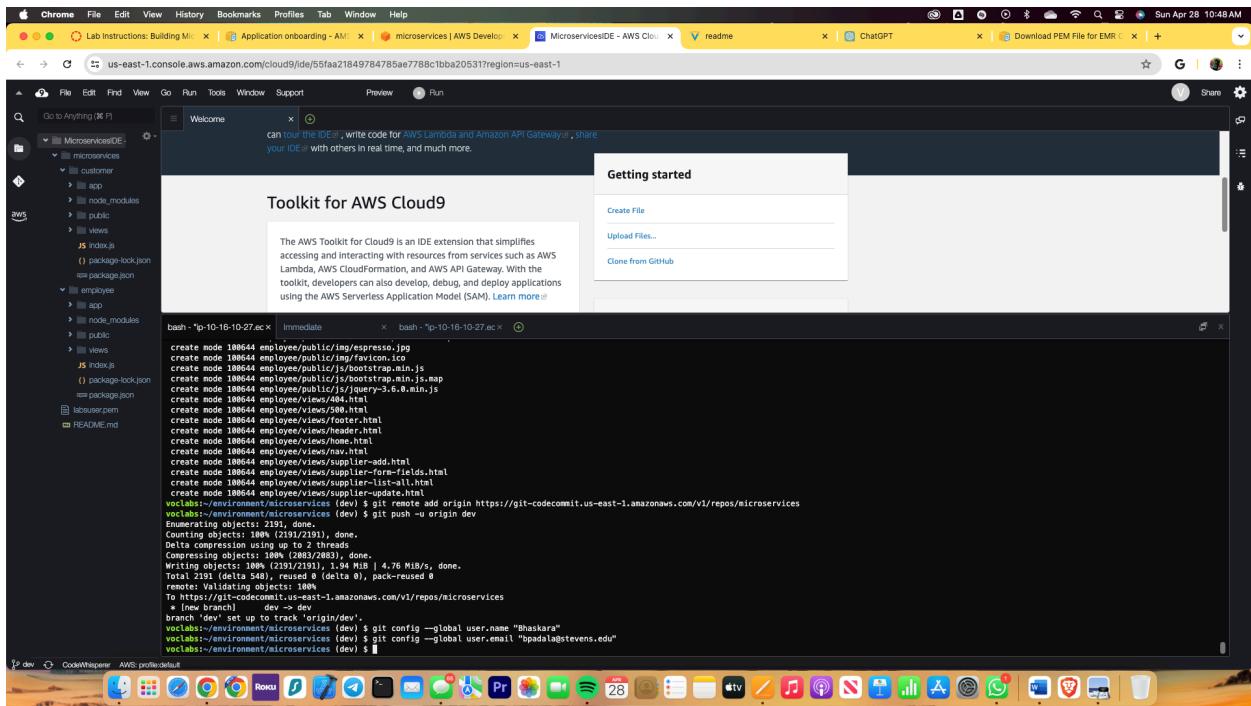
git -> p-10-16-10-27.ec2 > Immediate > bash -> p-10-16-10-27.ec2 >
create mode 10644 employee/public/css/bootstrap.min.css
create mode 10644 employee/public/css/bootstrap.min.css.map
create mode 10644 employee/public/img/espresso.jpg
create mode 10644 employee/public/img/favicon.ico
create mode 10644 employee/public/js/bootstrap.min.js
create mode 10644 employee/public/js/bootstrap.min.js.map
create mode 10644 employee/public/js/jquery-3.6.0.min.js
create mode 10644 employee/views/about.html
create mode 10644 employee/views/footer.html
create mode 10644 employee/views/header.html
create mode 10644 employee/views/nav.html
create mode 10644 employee/views/supplier-add.html
create mode 10644 employee/views/supplier-form-fields.html
create mode 10644 employee/views/supplier-list-all.html
create mode 10644 employee/views/supplier-update.html
voclabs:-~/environment/microservices (dev) $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
voclabs:-~/environment/microservices (dev) $ git push -u origin dev
Enumerating objects: 2193, done.
Counted objects: 2193,Delta compression using up to 2 threads
Compressing objects: 100% (2083/2083), done.
Writing objects: 100% (2193/2193), 1.94 MiB | 4.76 MiB/s, done.
Total 2193 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 * [new branch] dev > origin/dev
branch 'dev' set up to track 'origin/dev'.
voclabs:-~/environment/microservices (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.



```

can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more.

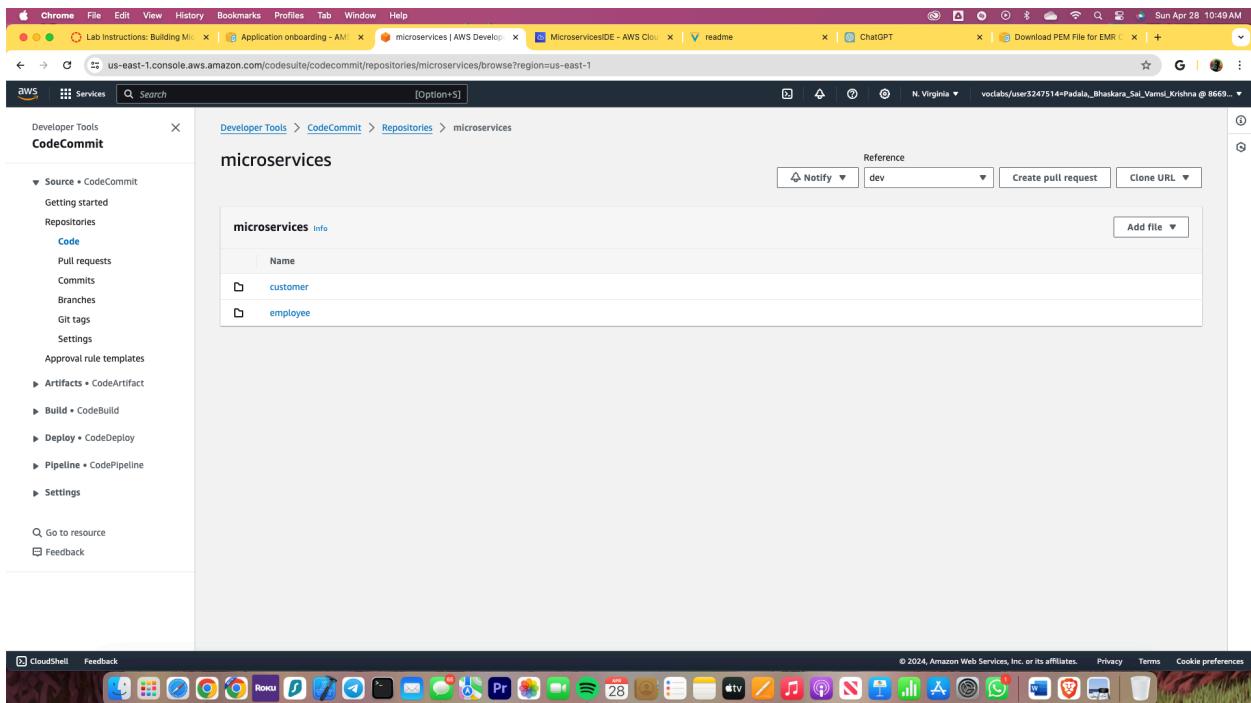
Welcome
Getting started
Create File
Upload Files...
Clone From GitHub

Toolkit for AWS Cloud9

The AWS Toolkit for Cloud9 is an IDE extension that simplifies accessing and interacting with resources from services such as AWS Lambda, AWS CloudFormation, and AWS API Gateway. With the toolkit, developers can also develop, debug, and deploy applications using the AWS Serverless Application Model (SAM). Learn more.

create mode 10644 employee/public/img/espresso.jpg
create mode 10644 employee/public/img/favicon.ico
create mode 10644 employee/public/js/bootstrap.min.js
create mode 10644 employee/public/js/jquery-3.6.0.min.js.map
create mode 10644 employee/views/404.html
create mode 10644 employee/views/footer.html
create mode 10644 employee/views/header.html
create mode 10644 employee/views/home.html
create mode 10644 employee/views/layouts.html
create mode 10644 employee/views/layouts/min.html
create mode 10644 employee/views/layouts/min.map
voclabs:/environment/microservices [dev] $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
voclabs:/environment/microservices [dev] $ git push -u origin dev
Enumerating objects: 2331, done.
Counting objects: 100% (2331/2331), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2083/2083), done.
Writing objects: 100% (2331/2331), 1.94 MiB | 4.76 MiB/s, done.
Total 2331 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 * [new branch] dev set up to track origin/dev'.
voclabs:/environment/microservices [dev] $ git config --global user.name "Bhaskara"
voclabs:/environment/microservices [dev] $ git config --global user.email "bpadala@stevens.edu"
voclabs:/environment/microservices [dev] $

```



Developer Tools > CodeCommit > Repositories > microservices

microservices

Name

- customer
- employee

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS CodeCommit interface. The left sidebar is collapsed, showing 'Source + CodeCommit' and 'Code'. The main area displays the 'microservices' repository. A breadcrumb navigation bar at the top indicates 'Developer Tools > CodeCommit > Repositories > microservices'. Below this is a 'microservices / customer' info section with a file tree. The tree shows the following structure:

- Name
- ..
- app
- node_modules
- public
 - views
 - index.js
 - package-lock.json
 - package.json

The screenshot shows the AWS CodeCommit interface again, this time with the 'employee' branch selected. The breadcrumb navigation bar indicates 'Developer Tools > CodeCommit > Repositories > microservices'. Below this is a 'microservices / employee' info section with a file tree. The tree structure is identical to the 'customer' branch:

- Name
- ..
- app
- node_modules
- public
 - views
 - index.js
 - package-lock.json
 - package.json

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

Task 4.1: Adjust the AWS Cloud9 instance security group settings

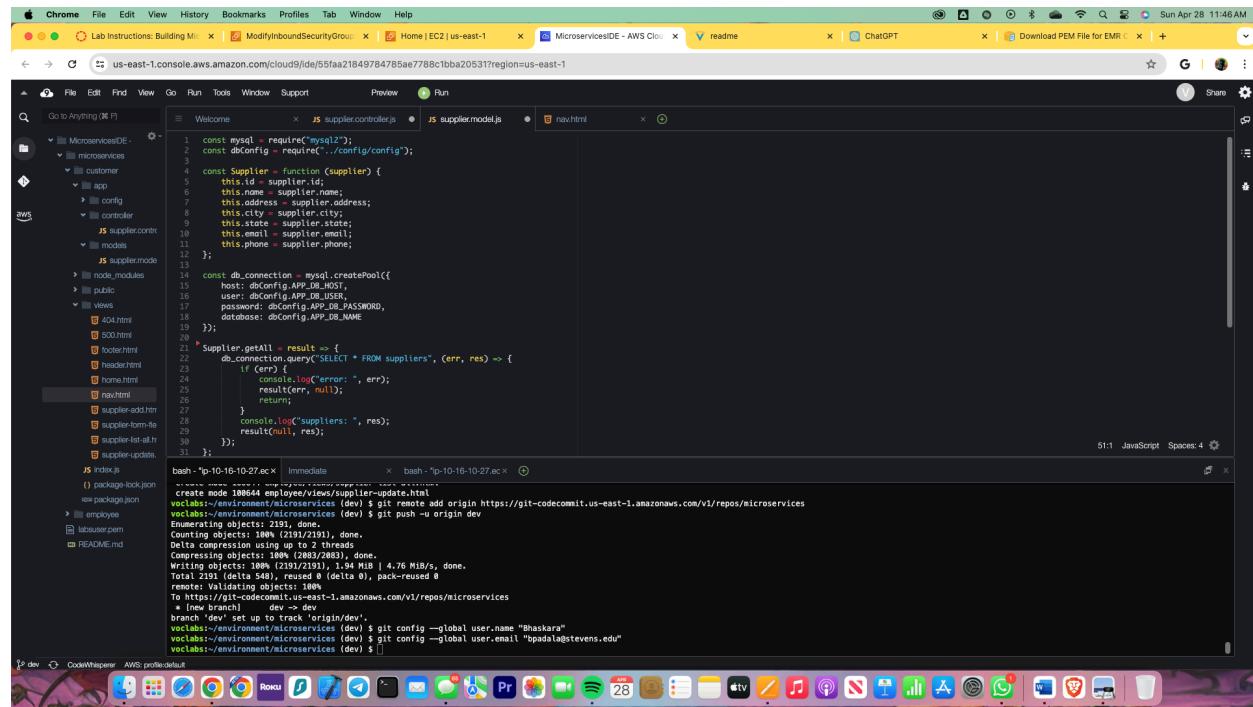
Now we will adjust the security group of the AWS Cloud9 EC2 instance to allow inbound network traffic on TCP ports 8080 and 8081.

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.



The screenshot shows the AWS Cloud9 IDE interface. The left sidebar displays the project structure:

```

MicroservicesDE
  - microservices
    - customer
      - app
        - config
        - controller
          - supplier.controller.js
        - models
        - views
          - 404.html
          - 500.html
          - footer.html
          - header.html
          - home.html
          - nav.html
          - supplier-add.html
          - supplier-form.html
          - supplier-list-all.html
          - supplier-update.html
    - public
    - views
  - index.js
  - package-lock.json
  - package.json
  - README.md
  - .gitignore
  - .lscssrc.json
  - .vscode
    - settings.json
  - .aws
    - profile.default

```

The main code editor window shows the content of `supplier.controller.js`:

```

const mysql = require('mysql2');
const dbConfig = require('../config/config');

const Supplier = function (supplier) {
  this.id = supplier.id;
  this.name = supplier.name;
  this.address = supplier.address;
  this.city = supplier.city;
  this.state = supplier.state;
  this.email = supplier.email;
  this.phone = supplier.phone;
};

const db_connection = mysql.createPool({
  host: dbConfig.APP_DB_HOST,
  user: dbConfig.APP_DB_USER,
  password: dbConfig.APP_DB_PASSWORD,
  database: dbConfig.APP_DB_NAME
});

Supplier.getAll = result => {
  db_connection.query("SELECT * FROM suppliers", (err, res) => {
    if (err) {
      console.log("Error: ", err);
      return err;
    }
    result(null, res);
  });
  console.log("Suppliers: ", res);
};

Supplier.getOne = result => {
  db_connection.query("SELECT * FROM suppliers WHERE id = ? ", [id], (err, res) => {
    if (err) {
      console.log("Error: ", err);
      return err;
    }
    result(null, res);
  });
};

Supplier.findById = result => {
  db_connection.query("SELECT * FROM suppliers WHERE id = ? ", [id], (err, res) => {
    if (err) {
      console.log("Error: ", err);
      return err;
    }
    result(null, res);
  });
};

Supplier.update = (id, supplier) => {
  db_connection.query("UPDATE suppliers SET name = ?, address = ?, city = ?, state = ?, email = ?, phone = ? WHERE id = ? ", [supplier.name, supplier.address, supplier.city, supplier.state, supplier.email, supplier.phone, id], (err, res) => {
    if (err) {
      console.log("Error: ", err);
      return err;
    }
    result(null, res);
  });
};

Supplier.delete = (id) => {
  db_connection.query("DELETE FROM suppliers WHERE id = ? ", [id], (err, res) => {
    if (err) {
      console.log("Error: ", err);
      return err;
    }
    result(null, res);
  });
};

```

Below the code editor, a terminal window shows the command-line output of a git push operation:

```

$ git push -u origin dev
Enumerating objects: 2191, done.
Counting objects: 100% (2191/2191), done.
Delta compression using up to 2 threads.
Compressing objects: 100% (1984/1984), done.
Writing objects: 100% (2191/2191), 1.94 MiB | 4.76 MiB/s, done.
Total 2191 (delta 548), reused 0 (delta 0), pack-reused 0
remote: Verifying objects: 100%
remote: Done
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 * [new branch] dev -> dev
branch 'dev' set up to track 'origin/dev'.
branch 'dev' set up to track 'origin/dev'.
$ git config --global user.name "Bhaskara"
$ git config --global user.email "bpadala@stevens.edu"
$ 

```

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

```

git add .
git commit -m "Add a new supplier"
[origin/dev] Add a new supplier
 1 file changed, 1 insertion(+)

```

The screenshot shows a Mac desktop with a browser window open to the AWS Cloud9 code editor. The terminal window shows a git commit message:

```

git add .
git commit -m "Add a new supplier"
[origin/dev] Add a new supplier
 1 file changed, 1 insertion(+)

```

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.

```

git add .
git commit -m "Remove supplier-update.html"
[origin/dev] Remove supplier-update.html
 1 file changed, 1 deletion(-)

```

The screenshot shows a Mac desktop with a browser window open to the AWS Cloud9 code editor. The terminal window shows a git commit message:

```

git add .
git commit -m "Remove supplier-update.html"
[origin/dev] Remove supplier-update.html
 1 file changed, 1 deletion(-)

```

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

```

const bodyParser = require('body-parser');
const cors = require('cors');
const supplier = require('../app/controller/supplier.controller');
const app = require('express');
const mustacheExpress = require('mustache-express');
const favicon = require('serve-favicon');

// parse requests of content-type: application/json
app.use(bodyParser.json());
// parse requests of content-type: application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: true}));
app.use(cors());
app.options('*', cors());
app.engine('html', mustacheExpress());
app.set('view engine', 'html');
app.set('views', __dirname + '/views');
app.use(express.static('public'));
app.use(favicon(__dirname + '/public/img/favicon.ico'));


// list all the suppliers
app.get('/', (req, res) => {
  res.render('home', {});
});

app.get('/suppliers', supplier.findAll);
// show the odd supplier form
app.get('/supplier-add', supplier.create);
// receive the odd supplier POST
app.post('/supplier-add', supplier.create);
// show the update form
app.get('/supplier-update/:id', supplier.findOne);
// receive the update POST
app.post('/supplier-update/:id', supplier.update);
// handle the POST to delete a supplier
app.delete('/supplier-remove/:id', supplier.remove);
// handle 404
app.use(function (req, res, next) {
  res.status(404).render('404', {});
});

// set port to listen for requests
const app_port = process.env.APP_PORT || 8080;
app.listen(app_port, () => {
  console.log(`Server is running on port ${app_port}.`);
});

```

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

The screenshot shows the AWS Cloud9 IDE interface. On the left, a file tree displays a project structure under 'MicroservicesIDE - /home/vc'. The 'Dockerfile' tab is selected, showing the following Dockerfile content:

```

FROM node:13-alpine
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]

```

The terminal window at the bottom shows the user performing several git operations:

```

vclabs:~/environment/microservices$ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
vclabs:~/environment/microservices$ git push -u origin dev
Enumerating objects: 100% (2191/2191), done.
Counting objects: 100% (2191/2191), done.
Delta compression using up to 2 threads.
Compressing objects: 100% (1204/1204), done.
Writing objects: 100% (2191/2191), 1.64 MiB | 4.76 MiB/s, done.
Total 2191 (delta 548), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 * [new branch] dev -> dev
branch 'dev' set up to track 'origin/dev'.
vclabs:~/environment/microservices$ git config --global user.name "Bhaskara"
vclabs:~/environment/microservices$ git config --global user.email "bpadala@stevens.edu"
vclabs:~/environment/microservices$ 

```

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 .`

The screenshot shows the AWS Cloud9 IDE interface. The terminal window at the bottom displays the output of the `docker build` command:

```

audited 78 packages in 0.747s
found 18 vulnerabilities (3 high, 2 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
Removing intermediate container 7f511740e3c1
--> bfde204f6801
Step 0 : EXPOSE 8080
--> Running in f1c86ee61eec
Removing intermediate container f1c86ee61eec
--> 7192268a72cd
Step 0 : CMD ["npm", "run", "start"]
--> Running in 140ec53821fb
Removing intermediate container 140ec53821fb
--> 3f4803b2cd9d
Successfully built 3f4803b2cd9d
Successfully tagged customer:latest
vclabs:~/environment/microservices$ 

```

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

```

FROM node:11-alpine
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["run", "start"]

```

The terminal window shows the Dockerfile content. Below it, a command is run:

```

bash -> ip-10-16-10-27.ec2 ~
└──> ./node_modules/.bin/coffee-wiggle &
    <-- Running in ffe2573803d
Removing intermediate container ffe2573803d
--> f7a1f835ecb2
Step 4/7 : RUN npm install
--> bc2197267a1
Step 5/7 : RUN npm install
--> Running in f751174e04c1
npm WARN coffee-wiggle@0.8 No repository field.

audited 78 packages in 0.747s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit --audit-only` for details
Removing intermediate container f751174e04c1
--> bff8e204f708
Step 6/7 : EXPOSE 8080
--> f1c66ee61eec
Removing intermediate container f1c66ee61eec
--> 7192206a72cd
Step 7/7 : CMD ["run", "run", "start"]
--> Running in 148ec53821fb
Removing intermediate container 148ec53821fb
--> 3fa0302c909b
Successfully built customer:latest
vocabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
customer latest 3f1402ed9b 59 seconds ago 82.7MB
node 11-alpine f18da2f58c3d 3 years ago 75.5MB
vocabs:~/environment/microservices/customer (dev) $

```

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.

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS Cloud9 IDE interface. On the left, a file tree displays the project structure: MicroservicesIDE - /home/ec2-user/.aws/microservices/customer/app/node_modules/public/views/Dockerfile. The Dockerfile content is:

```
FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

The terminal window shows the execution of the Dockerfile:

```
bash - *p-10-16-10-27.ecx Immediate bash - *p-10-16-10-27.ecx
--> running in ffe92573083d
Removing intermediate container ffe92573083d
--> be219f2d67a3
Step 4/7 : COPY . .
--> be219f2d67a3
Step 5/7 : RUN npm install
--> 7f511740e3c1
--> 7f511740e3c1
npm WARN coffee-script@1.0.0 No repository field.

audited 78 packages in 0.77s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
Removing intermediate container 7f511740e3c1
--> 7f511740e3c1
Step 6/7 : EXPOSE 8080
--> Running in f1c86ee61eec
Removing intermediate container f1c86ee61eec
--> f1c86ee61eec
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 140ec53821fb
Removing intermediate container 140ec53821fb
--> 140ec53821fb
Successfully built 3f408b2cd9d0
Successfully tagged customer:latest
vocabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
customer           latest   3f408b2cd9d0  6 minutes ago  82.7MB
node               11-alpine  f18da2f58c3d  4 years ago   75.9MB
vocabs:~/environment/microservices/customer (dev) $ dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
vocabs:~/environment/microservices/customer (dev) $ echo $dbEndpoint
```

The screenshot shows the AWS Cloud9 IDE interface. On the left, a file tree displays the project structure: MicroservicesIDE - /home/ec2-user/.aws/microservices/customer/app/node_modules/public/views/Dockerfile. The Dockerfile content is identical to the previous screenshot:

```
FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

The terminal window shows the execution of the Dockerfile, identical to the first screenshot.

Here we will run the following command :

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

The screenshot shows the AWS Lambda IDE interface. On the left, a sidebar displays project files: Dockerfile, index.js, package-lock.json, package.json, employee, labsource.pem, and README.md. The main workspace shows a Dockerfile with the following content:

```

FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]

```

Below the Dockerfile is a terminal window titled "bash - *p-10-16-10-27.ec x Immediate". It shows the build process of the Docker image:

```

Step 1/7 : FROM node:11-alpine
--> 7f511740e3c1
Step 2/7 : RUN mkdir -p /usr/src/app
--> Running in 7f511740e3c1
npm WARN coffee_apigl@0.0 No repository field.

audited 78 packages in 0.747s
Found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 7f511740e3c1
Step 3/7 : WORKDIR /usr/src/app
--> Removing intermediate container f1c86ee61eec
Step 4/7 : COPY .
--> Removing intermediate container 39223672a72d
Step 5/7 : RUN npm install
--> Running in 140ec53821fb
Removing intermediate container 140ec53821fb
--> 3f4832c2d9d8
Successfully built 3f4832c2d9d8
Successfully tagged customer:latest

```

After the build, commands are run to start the container and check its configuration:

```

voclabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
customer latest 3f4832c2d9d8 6 minutes ago 82.7MB
node 11-alpine f18da2f58cd3 4 years ago 75.9MB
voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST=$dbEndpoint
supplerdb.cpxlb3rj5yp.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST=$dbEndpoint customer
0f2b59ffread customer "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
voclabs:~/environment/microservices/customer (dev) $ 

```

Now we will run the following command :
docker images

The screenshot shows the AWS Lambda IDE interface. On the left, a sidebar displays project files: Dockerfile, index.js, package-lock.json, package.json, employee, labsource.pem, and README.md. The main workspace shows a Dockerfile with the following content:

```

FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]

```

Below the Dockerfile is a terminal window titled "bash - *p-10-16-10-27.ec x Immediate". It shows the build process of the Docker image:

```

Step 1/7 : FROM node:11-alpine
--> 7f511740e3c1
Step 2/7 : RUN mkdir -p /usr/src/app
--> Running in 7f511740e3c1
npm WARN coffee_apigl@0.0 No repository field.

audited 78 packages in 0.747s
Found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 7f511740e3c1
Step 3/7 : WORKDIR /usr/src/app
--> Removing intermediate container f1c86ee61eec
Step 4/7 : COPY .
--> Removing intermediate container 39223672a72d
Step 5/7 : RUN npm install
--> Running in 140ec53821fb
Removing intermediate container 140ec53821fb
--> 3f4832c2d9d8
Successfully built 3f4832c2d9d8
Successfully tagged customer:latest

```

After the build, commands are run to start the container and check its configuration:

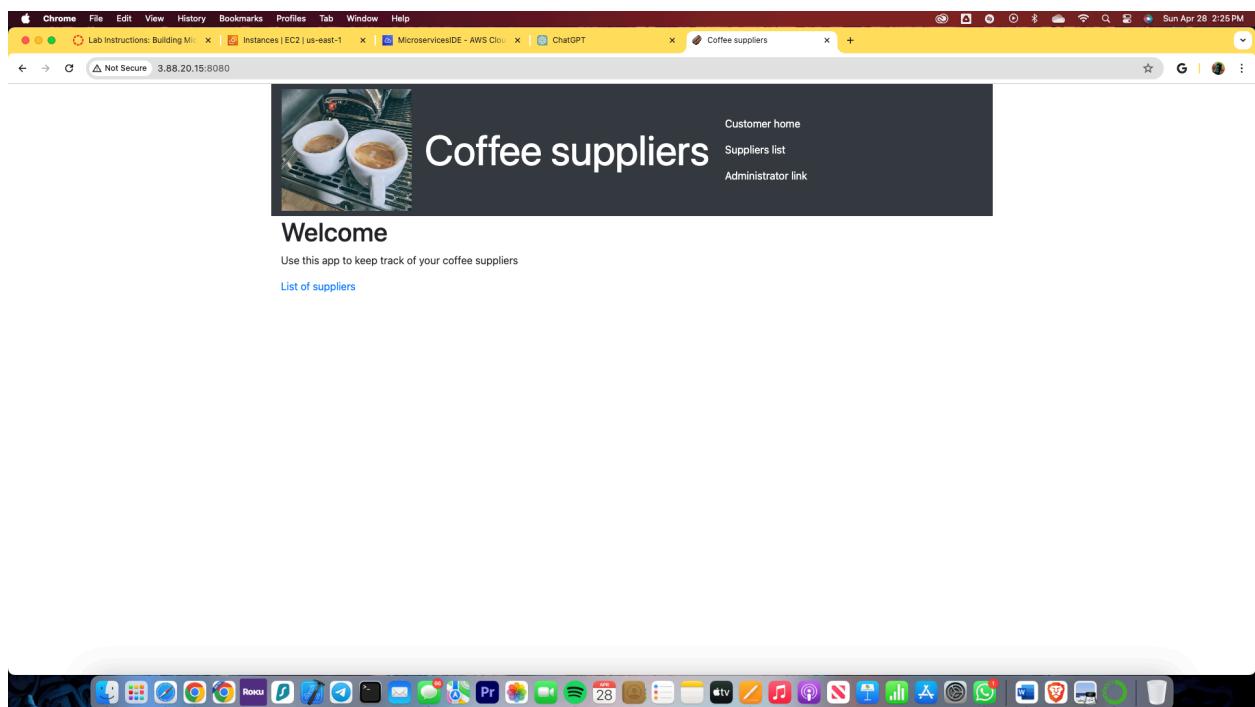
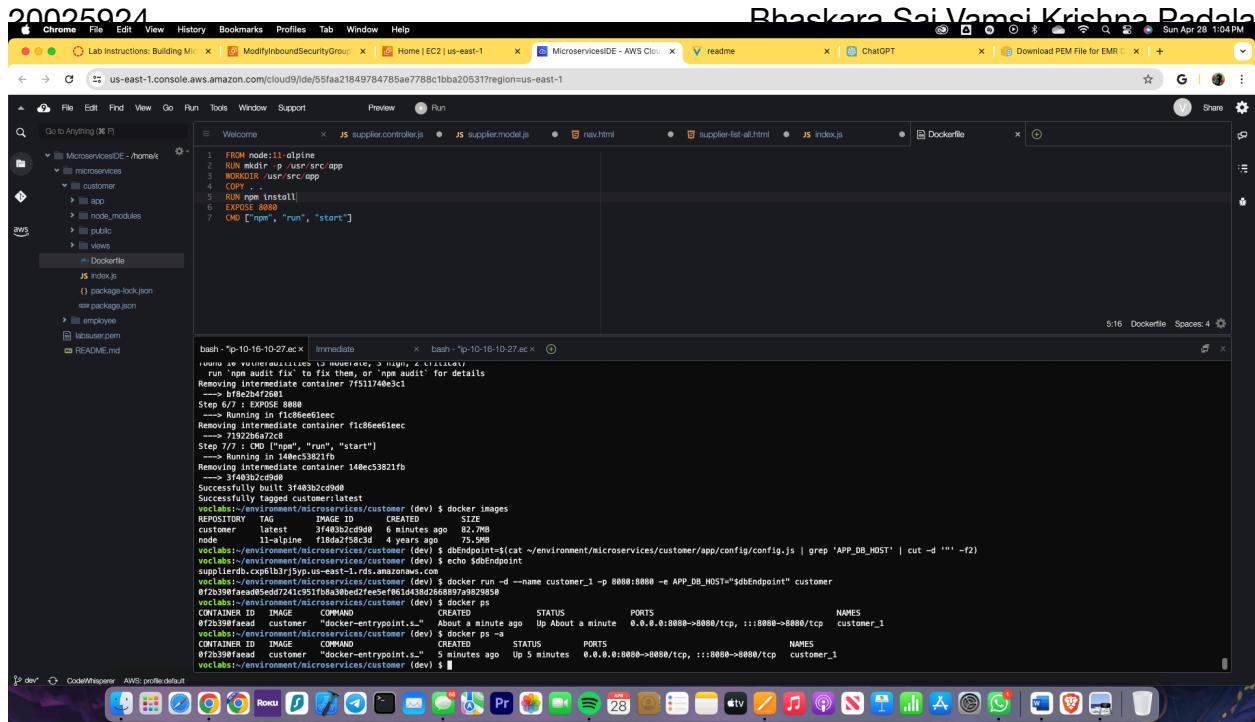
```

voclabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
customer latest 3f4832c2d9d8 6 minutes ago 82.7MB
node 11-alpine f18da2f58cd3 4 years ago 75.9MB
voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST=$dbEndpoint
supplerdb.cpxlb3rj5yp.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST=$dbEndpoint customer
0f2b59ffread customer "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
voclabs:~/environment/microservices/customer (dev) $ 

```

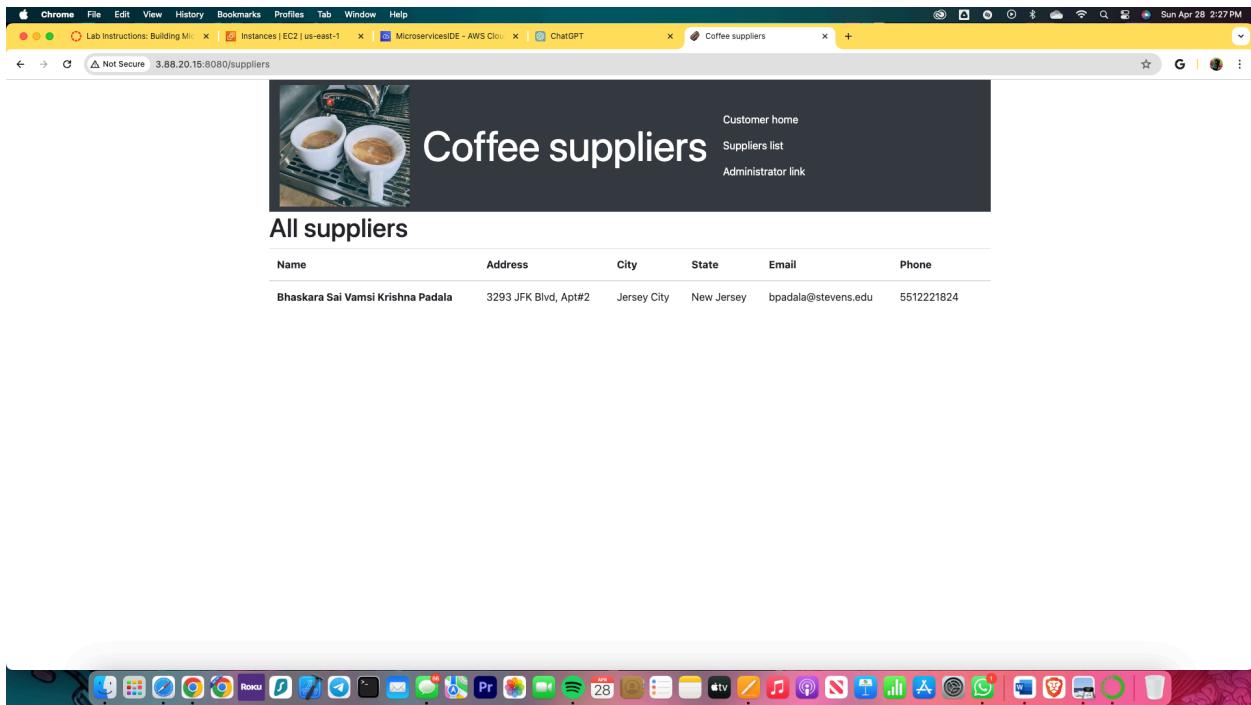
Now we ran the following code in the terminal and we got the following output.
voclabs:~/environment/microservices/customer (dev) \$ docker ps -a

COUNTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
CONTAINER NAMES					
0f2b59ffread	customer	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes	
					0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
					customer_1

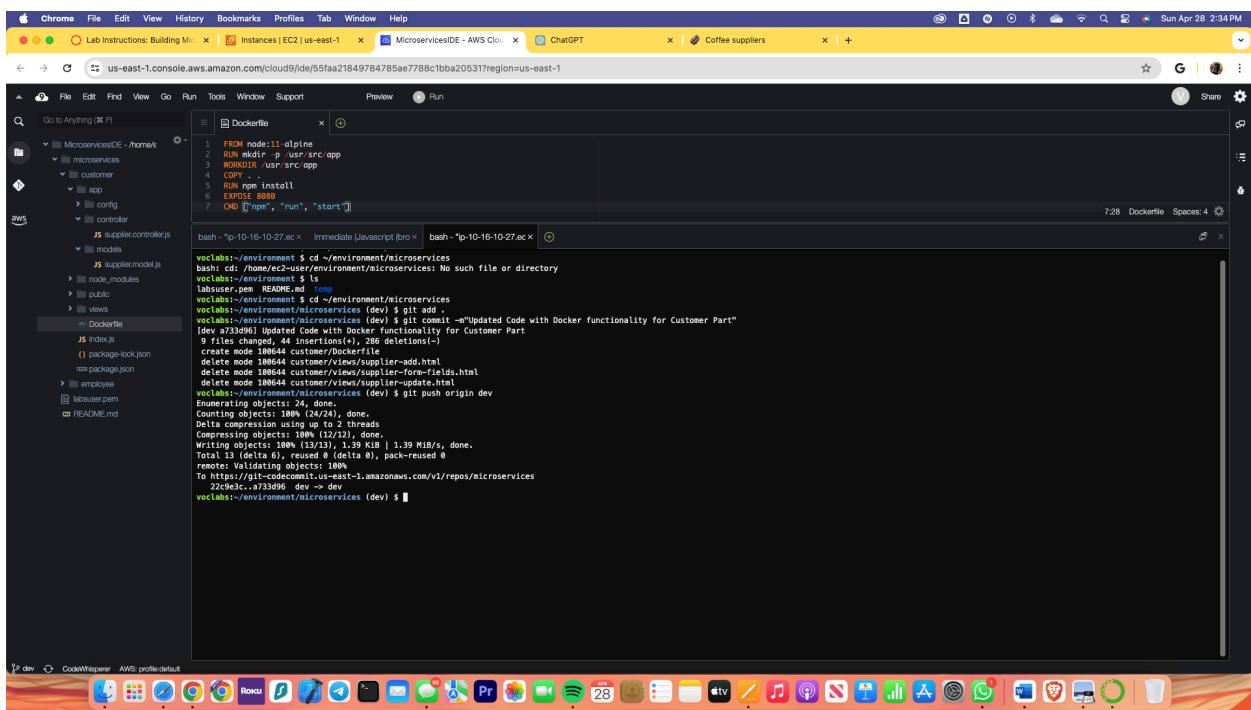


Now we load the following page in a new browser tab. Then we replaced the IP address placeholder with the public IPv4 address of the AWS Cloud9 instance <http://3.88.20.15:8080>. The web application microservice then got loaded in the browser as shown above.

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



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



We can verify our commits as follows:

Commit ID	Commit message	Commit date	Authored date	Author	Committer	Actions
a733d967	Updated Code with Docker functionality for Customer Part	4 minutes ago	4 minutes ago	Bhaskara	Bhaskara	Copy ID Browse
22c9e3c2	two unmodified copies of the application code	3 hours ago	3 hours ago	EC2 Default User	EC2 Default User	Copy ID Browse

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.

The screenshot shows a browser window with several tabs open, including "Lab Instructions: Building Microservices", "Instances | EC2 us-east-1", "Commits | AWS Developer Tools", "MicroservicesIDE - AWS CloudWatch Metrics", "ChatGPT", and "Coffee suppliers". The main content area displays a code editor for a Node.js application. The file being edited is `app/controllers/supplier.controller.js`. The code handles POST requests to create a supplier, validating the input and saving it to the database. A terminal window at the bottom shows the command to run the application and the logs indicating successful creation of a supplier record.

```
const Supplier = require('../models/supplier.model.js');
const { body, validationResult } = require('express-validator');

exports.create = [
    // Validate and sanitize the name field.
    body('name', 'The supplier name is required').trim().isLength({min: 1}).escape(),
    body('address', 'The supplier address is required').trim().isLength({min: 1}).escape(),
    body('city', 'The supplier city is required').trim().isLength({min: 1}).escape(),
    body('state', 'The supplier state is required').trim().isLength({min: 1}).escape(),
    body('phone', 'Phone number should be 10 digit number plus optional country code').trim().isMobilePhone().escape(),
    // Process request after validation and sanitization.
    (req, res) => {
        // Extract the validation errors from a request.
        const errors = validationResult(req);
        // Create a genre object with escaped and trimmed data.
        const supplier = new Supplier(req.body);
        if (errors.isEmpty()) {
            // There are no errors. Render the form again with sanitized values/error messages.
            res.render('supplier-add', {title: 'Create Supplier', supplier, errors: errors.array()});
        } else {
            // Data from form is valid, save to db
            Supplier.create(supplier, (err, data) => {
                if (err)
                    res.render('500', {message: 'Error occurred while creating the Supplier.'});
                else
                    res.redirect('/suppliers');
            });
        }
    }
];

module.exports = exports.create;
```

```
bash -p 10-16-10-27.ac x bash -p 10-16-10-27.ac x bash -p 10-16-10-27.ac x
vocabs:~/environment/microservices/employee
vocabs:~/environment/microservices/employee (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
25:         else res.redirect("/suppliers");
80:     } else res.redirect("/suppliers");
103:   } else res.redirect("/suppliers");
vocabs:~/environment/microservices/employee (dev) $
```

After changes, we get the following output.

The screenshot shows a browser window with multiple tabs open, displaying AWS Lambda function code and terminal logs.

Browser Tabs:

- Lab Instructions: Building M...
- Instances | EC2 us-east-1
- Commits AWS Developer Tools
- MicroservicesIDE - AWS Cloud9
- ChatGPT
- Coffee suppliers

Code Editor:

File Edit View Go Run Tools Window Support Preview Run

Go to Anything (6 E)

MicroservicesIDE - home.html

- microservices
 - customer
 - employee
 - app
 - config
 - controller
 - JS supplier.controller.js
 - models
 - JS supplier.model.js
 - node_modules
 - public
 - views
 - JS index.js
 - package-lock.json
- lstsources.yaml
- README.md

Dockerfile

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const cors = require('cors');
4 const supplier = require('../app/controller/supplier.controller');
5 const app = express();
6 const mustacheExpress = require('mustache-express')
7 const favicon = require('serve-favicon');
8
9 // parse requests of content-type: application/json
10 app.use(bodyParser.json());
11 // parse requests of content-type: application/x-www-form-urlencoded
12 app.use(bodyParser.urlencoded({extended: true}));
13 app.use(cors());
14 app.options('*', cors());
15 app.engine('html', mustacheExpress());
16 app.set('view engine', 'html');
17 app.set('views', __dirname + '/views')
18 app.use(express.static('public'));
19 app.use(favicon._dirname + '/public/img/favicon.ico');
20
21 // list all the suppliers
22 app.get('/admin', (req, res) => {
23   res.render('home', {});
24 });
25
26 // show the add supplier form
27 app.get('/admin/supplier-add', (req, res) => {
28   res.render('supplier-add', {});
29 });
30
31 app.get('/supplier-add', supplier.create);
32 app.get('/', (req, res) => {
33   app.get('/suppliers', supplier.findAll);
34 });
35 app.get('/supplier-add', (req, res) => {
36   app.get('/supplier-update/:id', supplier.findOne);
37 });
38 app.post('/supplier-update/:id', supplier.update);
39 app.get('/supplier-remove/:id', supplier.remove);
40
41 module.exports = app;
42
```

48:4 JavaScript Spaces: 4

Terminal:

```
bash -> 10-16-10-27.ec x bash -> 10-16-10-27.ec x Immediate Javascript (bro x bash -> 10-16-10-27.ec x
voclab:~/environment/microservices/employee
voclab:~/environment/microservices/employee (dev) $ grep -n "redirect" app/controller/supplier.controller.js
25:           else res.redirect("suppliers");
86:           } else res.redirect("suppliers");
133:           } else res.redirect("suppliers");
voclab:~/environment/microservices/employee (dev) $ grep -n "app.get" app.post* index.js
22:app.get("/", (req, res) => {
23:  app.get("/suppliers", supplier.findAll);
24:  app.get("/supplier-add", (req, res) => {
25:    app.post("supplier-add", supplier.create);
26:    app.get("/supplier-update/:id", supplier.findOne);
27:    app.post("/supplier-update/:id", supplier.update);
28:    app.get("/supplier-remove/:id", supplier.remove);
29:  }
voclab:~/environment/microservices/employee (dev) $
```

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

```

Go to Anything (⌘ F)
MicroservicesIDE - home
  └── microservices
    └── employee
      └── app
        └── controller
          └── supplier.controller.js
            └── JS supplier.model.js
  └── node_modules
  └── public
  └── views
    └── JS index.js
  package-lock.json
  package.json
  labuser.pem
  README.md

bash -p ip-10-16-10-27.ec x bash -p ip-10-16-10-27.ec x Immediate (JavaScript) (bro) x bash -p ip-10-16-10-27.ec x
voclabs:~/environment $ cd ~/environment/microservices/employee
voclabs:~/environment/microservices/employee (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
25:           else res.redirect('/suppliers');
86:           } else res.redirect('/suppliers');
87:         } else res.redirect('suppliers');
103:       } else res.redirect('/suppliers');
voclabs:~/environment/microservices/employee (dev) $ grep -n 'app.get' index.js
22:app.get('/', (req, res) => {
23:  res.render('index', { });
24:  // receive the add supplier POST
25:  app.post('/admin/supplier-add', supplier.create);
26:  // show the update form
27:  app.get('/admin/supplier-update/:id', supplier.findOne);
28:  // receive the update POST
29:  app.post('/admin/supplier-update/:id', supplier.update);
30:  // receive the POST to delete a supplier
31:  app.post('/admin/supplier-remove/:id', supplier.remove);
32:  // handle 404
33:  app.use(function (req, res, next) {
34:    res.status(404).render('404', { });
35:  });
36:  // set port, listen for requests
37:  const app_port = process.env.APP_PORT || 8081
38:  app.listen(app_port, () => {
39:    console.log(`Server is running on port ${app_port}.`);
40:  });
41: });

voclabs:~/environment/microservices/employee (dev) $ grep -n 'app.get' index.js
22:app.get('/', (req, res) => {
23:  res.render('index', { });
24:  // receive the add supplier POST
25:  app.post('/admin/supplier-add', supplier.create);
26:  // show the update form
27:  app.get('/admin/supplier-update/:id', supplier.findOne);
28:  // receive the update POST
29:  app.post('/admin/supplier-update/:id', supplier.update);
30:  // receive the POST to delete a supplier
31:  app.post('/admin/supplier-remove/:id', supplier.remove);
32:  // handle 404
33:  app.use(function (req, res, next) {
34:    res.status(404).render('404', { });
35:  });
36:  // set port, listen for requests
37:  const app_port = process.env.APP_PORT || 8081
38:  app.listen(app_port, () => {
39:    console.log(`Server is running on port ${app_port}.`);
40:  });
41: });

voclabs:~/environment/microservices/employee (dev) $ 

```

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/*

```

Go to Anything (⌘ F)
MicroservicesIDE - home
  └── microservices
    └── employee
      └── app
        └── controller
          └── supplier.controller.js
            └── JS supplier.model.js
  └── node_modules
  └── public
  └── views
    └── JS index.js
  package-lock.json
  package.json
  labuser.pem
  README.md

bash -p ip-10-16-10-27.ec x bash -p ip-10-16-10-27.ec x Immediate (JavaScript) (bro) x bash -p ip-10-16-10-27.ec x
voclabs:~/environment $ cd ~/environment/microservices/employee
voclabs:~/environment/microservices/employee (dev) $ grep -n 'action' views/*
25:           else res.redirect('suppliers');
86:           } else res.redirect('/suppliers');
87:         } else res.redirect('suppliers');
103:       } else res.redirect('/suppliers');
voclabs:~/environment/microservices/employee (dev) $ grep -n 'app.get' index.js
22:app.get('/', (req, res) => {
23:  res.render('index', { });
24:  // receive the add supplier POST
25:  app.post('/admin/supplier-add', supplier.create);
26:  // show the update form
27:  app.get('/admin/supplier-update/:id', supplier.findOne);
28:  // receive the update POST
29:  app.post('/admin/supplier-update/:id', supplier.update);
30:  // receive the POST to delete a supplier
31:  app.post('/admin/supplier-remove/:id', supplier.remove);
32:  // handle 404
33:  app.use(function (req, res, next) {
34:    res.status(404).render('404', { });
35:  });
36:  // set port, listen for requests
37:  const app_port = process.env.APP_PORT || 8081
38:  app.listen(app_port, () => {
39:    console.log(`Server is running on port ${app_port}.`);
40:  });
41: });

voclabs:~/environment/microservices/employee (dev) $ grep -n 'app.get' index.js
22:app.get('/', (req, res) => {
23:  res.render('index', { });
24:  // receive the add supplier POST
25:  app.post('/admin/supplier-add', supplier.create);
26:  // show the update form
27:  app.get('/admin/supplier-update/:id', supplier.findOne);
28:  // receive the update POST
29:  app.post('/admin/supplier-update/:id', supplier.update);
30:  // receive the POST to delete a supplier
31:  app.post('/admin/supplier-remove/:id', supplier.remove);
32:  // handle 404
33:  app.use(function (req, res, next) {
34:    res.status(404).render('404', { });
35:  });
36:  // set port, listen for requests
37:  const app_port = process.env.APP_PORT || 8081
38:  app.listen(app_port, () => {
39:    console.log(`Server is running on port ${app_port}.`);
40:  });
41: });

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">
method="POST">

```

```

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">
method="POST">

```

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

```
Chrome File Edit View History Bookmarks Profiles Tab Window Help
Lab Instructions: Building Mi... Instances [EC2] us-east-1 Commits AWS Developer Tools MicroservicesDE - AWS CloudWatch ChatGPT Coffee suppliers
us-east-1.console.aws.amazon.com/cloud9/ide/5f5fa21849784785ae7788c1ba20531?region=us-east-1

bash *-p-10-16-10-27.ec x bash *-p-10-16-10-27.ec x Immediate Javascript file x bash *-p-10-16-10-27.ec x

voclab:~/environment/microservices/employee
voclab:~/environment/microservices/employee $ cd ~/environment/microservices/employee
voclab:~/environment/microservices/employee (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
23:         res.redirect('http://'+req.headers.host);
86:     } else res.redirect('/suppliers');
103: }
voclab:~/environment/microservices/employee (dev) $ grep -n 'app.get' index.js
12: app.get('/', (req, res) =>
25: app.get('/suppliers', supplier.findAll);
27: app.get('/supplier-add', (req, res) =>
31: app.post('/supplier-add', supplier.create);
32: app.get('/supplier-update/:id', supplier.findOne);
35: app.post('/supplier-update', supplier.update);
37: app.post('/supplier-remove/:id', supplier.remove);
voclab:~/environment/microservices/employee (dev) $ grep -n 'action' views/*
views/supplier-add.html[1]:        <form action="/supplier-add" method="POST">
views/supplier-add.html[1]:            <input type="submit" value="Add a new supplier" />
views/supplier-add.html[1]:        </form>
views/supplier-add.html[3]:        <form action="/supplier-remove/({id})" method="POST">
views/supplier-add.html[3]:            <input type="submit" value="Remove supplier" />
views/supplier-list-all.html[32]:        <td><a href="/supplier-add">Add a new supplier</a></td>
views/home.html[7]:        <p><a href="#">List of suppliers</a></p>
voclab:~/environment/microservices/employee (dev) $
```

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

The screenshot shows a browser window with several tabs open, including 'Lab Instructions: Building Microservices' (active), 'Instances | EC2 - us-east-1', 'Commits | AWS Developer Tools', 'MicroservicesIDE - AWS Cloud9', 'ChatGPT', and 'Coffee suppliers'. The main content area displays a code editor with a file structure for a microservices application. The file structure includes 'supplier.controller.js', 'index.js', 'supplier.model.js', 'supplier-add.html', 'supplier-update.html', 'supplier-list-all.html', and 'home.html'. The code in 'supplier.controller.js' handles requests for suppliers, including listing, adding, updating, and deleting. It uses AWS Lambda functions and CloudWatch logs for logging and tracking. The AWS Lambda functions are named 'supplier-controller' and 'supplier-post'. The CloudWatch logs show the execution of these functions with various parameters and responses.

```
1  ({>header})
2   <div class="container">
3     {>nav}
4     <div class="container">
5       <a href="#">Home
6       <p>Use this app to keep track of your coffee suppliers</p>
7       <a href="#">List of suppliers</a></p>
8     </div>
9   </div>
10  {>footer}
11
```

```
bash - *p-10-16-10-27.ecx bash - *p-10-16-10-27.ecx Immediate Javascript (bro) bash - *p-10-16-10-27.ecx
vocabs:~/environment/.local/share/vocabs
vocabs:~/environment/microservices/employees (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
25           else res.redirect('/suppliers');
26         } else res.redirect('/suppliers');
27       }
28     } else res.redirect('suppliers');
29 app.get('/', (req, res) => {
30   app.get('/suppliers', supplier.findAll);
31   app.post('/supplier-add', supplier.create);
32   app.get('/supplier-update/:id', supplier.findOne);
33   app.post('/supplier-update', supplier.update);
34   app.delete('/supplier-remove/:id', supplier.remove);
35 }
vocabs:~/environment/microservices/employees (dev) $ grep -n 'action' views/*
views/supplier-add.html:11      <form action="/supplier-add" method="POST">
views/supplier-update.html:11    <form action="/supplier-update" method="POST">
views/supplier-remove.html:11    <form action="/supplier-remove/(:id)" method="POST">
vocabs:~/environment/microservices/employees (dev) $ grep -n 'href' views/supplier-list-all.html views/home.html
views/supplier-list-all.html:27      href="#">/supplier-update((id))>edit</a></span></td>
views/supplier-list-all.html:28      <td><a href="#">Delete badge-success</a></td>
views/supplier-list-all.html:29      <td><a href="#">Suppliers List of suppliers</a></td>
vocabs:~/environment/microservices/employees (dev) $
```

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:

```
<a class="nav-link" href="/admin/suppliers">Administrator home</a>
```

The screenshot shows a Mac desktop with a terminal window and a browser window. The terminal window is running on a Mac OS X system, displaying AWS Lambda code for a 'supplier.controller.js' file. The browser window shows the AWS Lambda console for a function named 'SupplierController'. The Lambda function's code is identical to the one shown in the terminal window.

```

1 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
2   < class="navbar-brand page-title" href="/supplier">Manage coffee suppliers</a></div>
4   <div class="collapse navbar-collapse" id="navbarsupportedContent">
5     <ul class="navbar-nav mr-auto">
6       <li> <a class="nav-link" href="/admin/suppliers">Administrator home</a>
7       <a class="nav-link" href="/suppliers">Suppliers list</a>
8     </ul>
9   </div>
10 </nav>
11
12

```

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

The screenshot shows a Mac desktop with a terminal window and a browser window. The terminal window is running on a Mac OS X system, displaying AWS Lambda code for a 'supplier.controller.js' file. The browser window shows the AWS Lambda console for a function named 'SupplierController'. The Lambda function's code is identical to the one shown in the terminal window.

```

1 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
2   < class="navbar-brand page-title" href="/supplier">Manage coffee suppliers</a></div>
4   <div class="collapse navbar-collapse" id="navbarsupportedContent">
5     <ul class="navbar-nav mr-auto">
6       <li> <a class="nav-link" href="/admin/suppliers">Administrator home</a>
7       <a class="nav-link" href="/suppliers">Suppliers list</a>
8       <a class="nav-link" href="/">Customer home</a>
9     </ul>
10 </div>
11
12

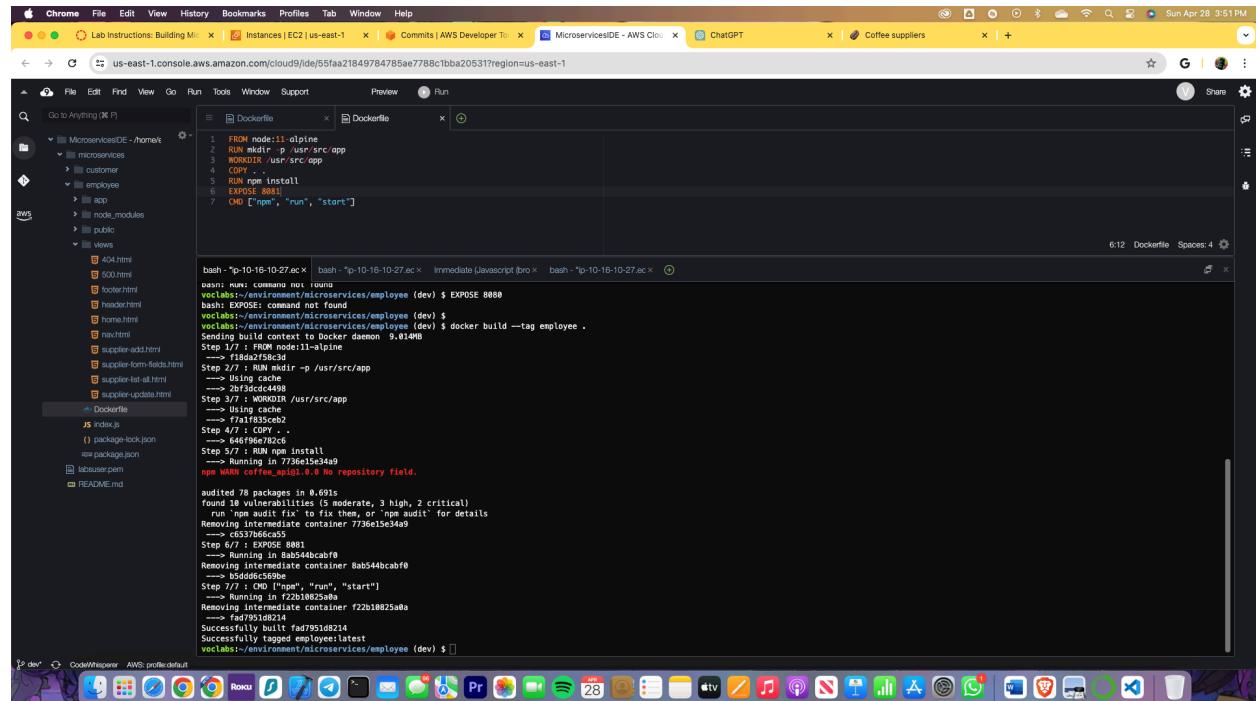
```

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.



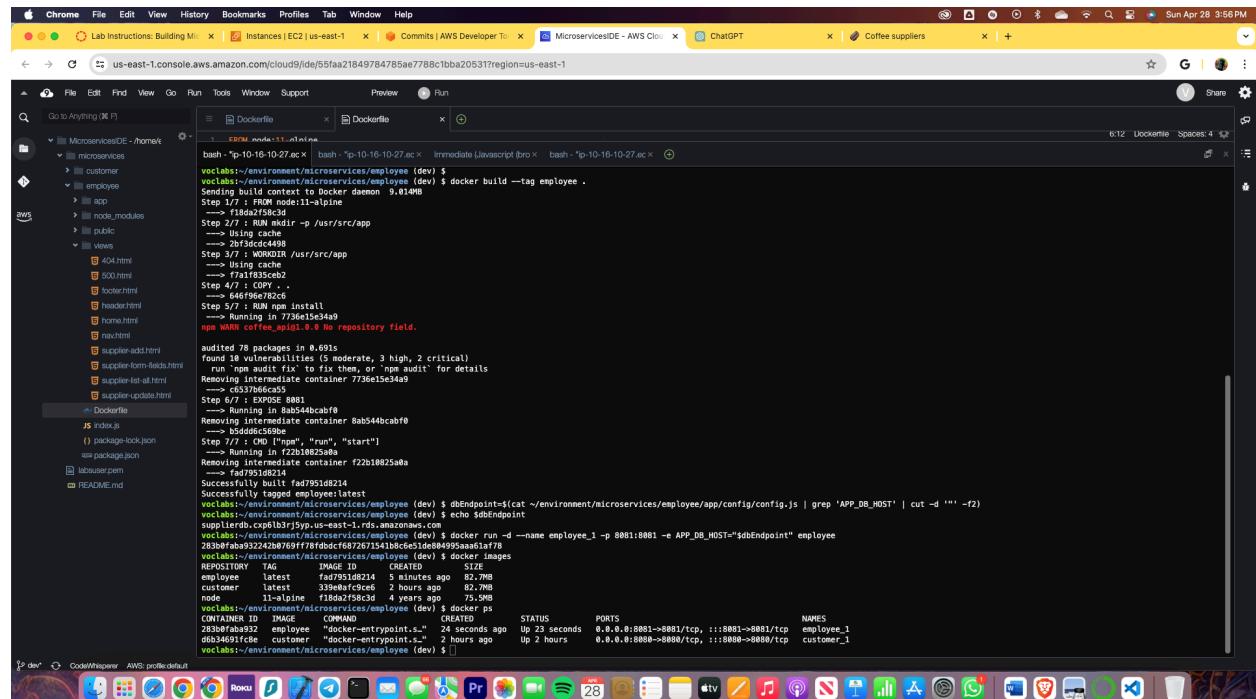
```

Dockerfile
FROM node:14-alpine
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8081
CMD ["npm", "run", "start"]

bash *-p10-16-10-27.ac* bash *-p10-16-10-27.ac* Immediate Javascript (bro) bash *-p10-16-10-27.ac*
voclabs:~/environment/microservices/employee (dev) $ EXPOSE 8081
bash *-p10-16-10-27.ac* bash *-p10-16-10-27.ac*
voclabs:~/environment/microservices/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.01MB
Step 1/7 : FROM node:14-alpine
--> f18ddcf5c8cb
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> 2b73dc4d498
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> f7a1f835ceb2
Step 4/7 : COPY . .
--> 646f96e702c6
Step 5/7 : RUN npm install
--> 73de276773b49
npm WARN coffee_uglify@0.0 No repository field.

audited 78 packages in 0.69s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 773be15e34a9
--> c6537066ca53
Step 6/7 : EXPOSE 8081
--> Running in fad7951d8214
Removing intermediate container Bab544bcabf0
--> b5dddc589ba
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in f22010825a0
Removing intermediate container f22010825a0
--> fa07951d8214
Successfully built fad7951d8214
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ 

```



```

Dockerfile
EXPOSE 8081
bash *-p10-16-10-27.ac* bash *-p10-16-10-27.ac* Immediate Javascript (bro) bash *-p10-16-10-27.ac*
voclabs:~/environment/microservices/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.01MB
Step 1/7 : FROM node:14-alpine
--> f18ddcf5c8cb
Step 2/7 : WORKDIR /usr/src/app
--> Using cache
--> 2b73dc4d498
Step 3/7 : COPY . .
--> 646f96e702c6
Step 4/7 : RUN npm install
--> 73de276773b49
npm WARN coffee_uglify@0.0 No repository field.

audited 78 packages in 0.69s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 773be15e34a9
--> c6537066ca53
Step 6/7 : EXPOSE 8081
--> Running in Bab544bcabf0
Removing intermediate container Bab544bcabf0
--> b5dddc589ba
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in f22010825a0
Removing intermediate container f22010825a0
--> fa07951d8214
Successfully built fad7951d8214
Successfully tagged employee:latest
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplieRDB.cpxlblrj9yp.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST=$dbEndpoint employee
283b0fab0924ab769f7ff6dcd6768276151b2cfc5e31de84909aaaf1af78
voclabs:~/environment/microservices/employee (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
employee latest f18ddcf5c8cb 2 hours ago 62.9MB
customer latest 339eafa9c6e6 2 hours ago 62.9MB
node 11-alpine f18ddcf5c8cb 4 years ago 75.9MB
voclabs:~/environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND STATUS PORTS NAMES
283b0fab0924... employee "docker-entrypoint.s..." 24 seconds ago Up 23 seconds 0.0.0.0:8081->8081/tcp, ::1:8081->8081/tcp employee_1
d6b34691fc8... customer "docker-entrypoint.s..." 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, ::1:8080->8080/tcp customer_1
voclabs:~/environment/microservices/employee (dev) $ 

```

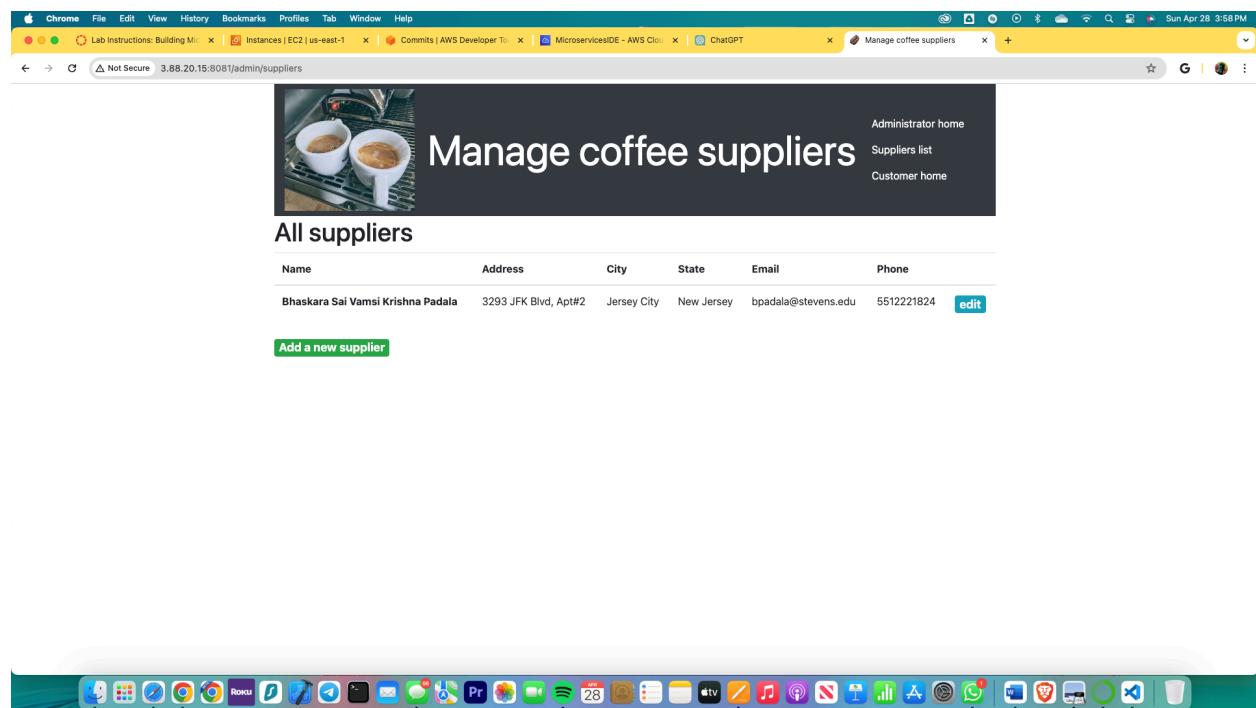
For the above screen,

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://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.

20025924

Bhaskara Sai Vamsi Krishna Padala

All suppliers

Name	Address	City	State	Email	Phone
Bhaskara Sai Vamsi Krishna Padala	3293 JFK Blvd, Apt#2	Jersey City	New Jersey	bpadala@stevens.edu	5512221824
Jamie Murphy	18 Stagg St	Jersey City	New Jersey	hvcjaf@wireconnected.com	8985691687

Add a new supplier

Delete supplier

Are you sure you want to delete supplier Jamie Murphy?

Close Delete this supplier

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.

All suppliers

Name	Address	City	State	Email	Phone
Bhaskara Sai Vamsi Krishna Padala	3293 JFK Blvd, Apt#2	Jersey City	New Jersey	bpadala@stevens.edu	5512221824

[Add a new supplier](#)

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

```

FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8081
CMD ["npm", "run", "start"]
  
```

```

bash -> p-10-16-10-27.ec < bash -> p-10-16-10-27.ec < Immediate Javascript (bro) < bash -> p-10-16-10-27.ec <
step 1 : RUN node:11-alpine
--> Running in f22010825a
Removing intermediate container f22010825a
-> fa079510214
Successfully built fa079510214
Successfully tagged employee:latest
voclabs:~/.environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/.environment/microservices/employee (dev) $ echo $dbEndpoint
supplier_endpoint=tcp://ds-aws.amazonaws.com
voclabs:~/.environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
283b9fab932420b769f7f8fdcd6f82767151b1bc5c51de94995aa61af78
voclabs:~/.environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9fab9324 employee "docker-entrypoint.s" 24 seconds ago Up 23 seconds 0.0.0.0:8081->8081/tcp, :::8080->8080/tcp employee_1
6085011c65 customer "docker-entrypoint.s" 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
voclabs:~/.environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9fab9324 employee "docker-entrypoint.s" 21 minutes ago Up 21 minutes 0.0.0.0:8081->8081/tcp, :::8080->8080/tcp employee_1
6085011c65 customer "docker-entrypoint.s" 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
  
```

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

20025924

Bhaskara Sai Vamsi Krishna Padala

A screenshot of a Mac desktop showing a terminal window with Docker commands and AWS CloudWatch logs. The terminal shows the creation of a Docker container named 'employee_1' with port 8081 mapped to 8080. It also lists other containers like 'customer_1' and 'node'. The AWS CloudWatch logs show the deployment of the 'employee' microservice.

```
bash -*p-10-16-10-27.ec x bash -*p-10-16-10-27.ec x Immediate Javascript [bro] x bash -*p-10-16-10-27.ec x
step //: lnu ["npm", "run", "start"]
--> Running in f22b10825a8a
Removing intermediate container f22b10825a8a
Successfully built fad795108214
Successfully tagged employee:latest
voclabs:/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierDb.endpoint.us-east-1.rds.amazonaws.com
voclabs:/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
283b9ab9324ab769f7fbfd6f8276151b8c651de8b4995aa81af78
voclabs:/environment/microservices/employee (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
employee latest fad795108214 5 minutes ago 82.7MB
customer latest 339ebafac3e5 2 hours ago 82.7MB
node latest 19a2a2a2a2a2 4 years ago 75.9MB
voclabs:/environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9ab932 employee "docker-entrypoint.s--" 24 seconds ago Up 23 seconds 0.0.0.0:8081->8081/tcp, ::1:8081->8081/tcp employee_1
d634691fcbe customer "docker-entrypoint.s--" 2 hours ago Up 2 hours 0.0.0.0:8880->8880/tcp, ::1:8880->8880/tcp customer_1
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9ab932 employee "docker-entrypoint.s--" 21 minutes ago Up 2 minutes 0.0.0.0:8080->8080/tcp, ::1:8080->8080/tcp employee_1
d634691fcbe customer "docker-entrypoint.s--" 2 hours ago Up 2 hours 0.0.0.0:8880->8880/tcp, ::1:8880->8880/tcp customer_1
voclabs:/environment/microservices/employee (dev) $ 
```

Now we will change the port number on docker file.

A screenshot of a Mac desktop showing a terminal window with Docker commands and AWS CloudWatch logs. The terminal shows the creation of a Docker container named 'employee_1' with port 8081 mapped to 8080. It also lists other containers like 'customer_1' and 'node'. The AWS CloudWatch logs show the deployment of the 'employee' microservice.

```
bash -*p-10-16-10-27.ec x bash -*p-10-16-10-27.ec x Immediate Javascript [bro] x bash -*p-10-16-10-27.ec x
step //: lnu ["npm", "run", "start"]
--> Running in f22b10825a8a
Removing intermediate container f22b10825a8a
Successfully built fad795108214
Successfully tagged employee:latest
voclabs:/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierDb.endpoint.us-east-1.rds.amazonaws.com
voclabs:/environment/microservices/employee (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
employee latest fad795108214 5 minutes ago 82.7MB
customer latest 339ebafac3e5 2 hours ago 82.7MB
node latest 19a2a2a2a2a2 4 years ago 75.9MB
voclabs:/environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9ab932 employee "docker-entrypoint.s--" 22 seconds ago Up 22 seconds 0.0.0.0:8081->8081/tcp, ::1:8081->8081/tcp employee_1
d634691fcbe customer "docker-entrypoint.s--" 2 hours ago Up 2 hours 0.0.0.0:8880->8880/tcp, ::1:8880->8880/tcp customer_1
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
283b9ab932 employee "docker-entrypoint.s--" 21 minutes ago Up 21 minutes 0.0.0.0:8881->8881/tcp, ::1:8881->8881/tcp employee_1
d634691fcbe customer "docker-entrypoint.s--" 2 hours ago Up 2 hours 0.0.0.0:8880->8880/tcp, ::1:8880->8880/tcp customer_1
voclabs:/environment/microservices/employee (dev) $ 
```

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_1`), run the following command:

```
docker rm -f employee_1
```

Now we will build docker

```

FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD [ "npm", "run", "start" ]

```

```

bash *-p-10-16-10-27.ac x bash *-p-10-16-10-27.ac x Immediate Javascript [bro] > bash *-p-10-16-10-27.ac x
--> Removing intermediate container 8ab544bcabf0
--> b5dddc5c9b8e
Step 7/7 : CMD ["npm", "run", "start"]
Removing intermediate container f220108250ea
--> fad7951d821a
Successfully built fad7951d821a
Successfully tagged employee_1:latest
voclaabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclaabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cxpblbjr5y.us-east-1.rds.amazonaws.com
voclaabs:~/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee_1
283bbfab932240b769f7f8f0dcf687267151b2c6c51de04995aa0la78
voclaabs:~/environment/microservices/employee (dev) $ docker images
REPOSITORY          TAG        IMAGE ID      CREATED          SIZE
employee_1          latest    fad7951d8214   4 days ago     527MB
customer           latest    339edaaf9c9e6   2 hours ago    82.9MB
node               11-alpine  f18dca2f28c3d   4 years ago    75.9MB
voclaabs:~/environment/microservices/employee (dev) $ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS          PORTS          NAMES
283bbfab9322        employee_1         "docker-entrypoint.s"   24 seconds ago   Up 23 seconds  0.0.0.0:8081->8081/tcp, 0.0.0.0:8880->8880/tcp   employee_1
d6b3491fcd8        customer           "docker-entrypoint.s"   2 hours ago     Up 2 hours       0.0.0.0:8880->8880/tcp, 0.0.0.0:8080->8080/tcp   customer_1
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS          PORTS          NAMES
283bbfab9322        employee_1         "docker-entrypoint.s"   21 minutes ago   Up 21 minutes  0.0.0.0:8081->8081/tcp, 0.0.0.0:8880->8880/tcp   employee_1
d6b3491fcd8        customer           "docker-entrypoint.s"   2 hours ago     Up 2 hours       0.0.0.0:8880->8880/tcp, 0.0.0.0:8080->8080/tcp   customer_1
voclaabs:~/environment/microservices/employee (dev) $ docker rm -f employee_1
voclaabs:~/environment/microservices/employee (dev) $ docker build
"docker build" requires exactly 1 argument.
See 'docker build --help'.
Usage: docker build [OPTIONS] PATH | URL | -
Build an image from a Dockerfile
voclaabs:~/environment/microservices/employee (dev) $ docker build --tag employee_1

```

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

```

FROM node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD [ "npm", "run", "start" ]

```

```

bash *-p-10-16-10-27.ac x bash *-p-10-16-10-27.ac x Immediate Javascript [bro] > bash *-p-10-16-10-27.ac x
--> Creating container...
--> 283bbfab9322        employee_1         "docker-entrypoint.s"   21 minutes ago   Up 21 minutes  0.0.0.0:8081->8081/tcp, 0.0.0.0:8880->8880/tcp   employee_1
voclaabs:~/environment/microservices/employee (dev) $ docker build --tag employee_1
"docker build" requires exactly 1 argument.
See 'docker build --help'.
Usage: docker build [OPTIONS] PATH | URL | -
Build an image from a Dockerfile
voclaabs:~/environment/microservices/employee (dev) $ docker build --tag employee_1
Sending build context to Docker daemon 9.01MB
Step 1/7 : FROM node:11-alpine
--> Using cache
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> 2b73dcd4498
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> f7a1f635ceb2
Step 4/7 : COPY . .
--> Using cache
--> 646f96e732c6
Step 5/7 : RUN npm install
--> Using cache
--> eec3a2500255
Step 6/7 : EXPOSE 8080
--> Using cache
Step 7/7 : CMD ["npm", "run", "start"]
--> Using cache
--> fad7951d821a
Successfully built fad7951d821a
Successfully tagged employee_1:latest
voclaabs:~/environment/microservices/employee (dev) $ 

```

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

```

1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const cors = require('cors');
4 const controller = require('../app/controller/supplier.controller');
5 const app = express();
6 const mustacheExpress = require('mustache-express')
7 const favicon = require('serve-favicon');
8
9 // parse requests of content-type: application/json
10 app.use(bodyParser.json());
11 // parse requests of content-type: application/x-www-form-urlencoded
12 app.use(bodyParser.urlencoded({extended: true}));
13 app.use(cors());
14 app.options('*', cors());
15 app.set('view engine', mustacheExpress)
16 app.set('view engine', 'html')
17 app.set('views', __dirname + '/views')
18 app.use(express.static(public));
19 app.use(favicon(__dirname + '/public/img/favicon.ico'));
20
21 // list all the suppliers
22 app.get('/', (req, res) => {
23   res.render('home', {});
24 });
25 app.get('/suppliers', supplier.findAll);
26 // show the add supplier form
27 app.get('/supplier-add', supplier.create);
28 // receive the add supplier POST
29 app.post('/supplier-add', supplier.create);
30 // show the update form
31 app.get('/supplier-update/:id', supplier.findOne);
32 // receive the update supplier
33 app.post('/supplier-update', supplier.update);
34 // receive the POST to delete a supplier
35 app.post('/supplier-remove/:id', supplier.remove);
36 // set port, listen for requests
37 const app_port = process.env.APP_PORT || 80
38 app.listen(app_port, () => {
39   app.use(function (req, res, next) {
40     res.status(404).render("404");
41   })
42 }
43
44 // set port, listen for requests
45 const app_port = process.env.APP_PORT || 80
46 app.listen(app_port, () => {
47   console.log(`Microservices is running on port ${app_port}`);
48 })
}

```

```

1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const cors = require('cors');
4 const controller = require('../app/controller/supplier.controller');
5 const app = express();
6 const mustacheExpress = require('mustache-express')
7 const favicon = require('serve-favicon');
8
9 // parse requests of content-type: application/json
10 app.use(bodyParser.json());
11 // parse requests of content-type: application/x-www-form-urlencoded
12 app.use(bodyParser.urlencoded({extended: true}));
13 app.use(cors());
14 app.options('*', cors());
15 app.set('view engine', mustacheExpress)
16 app.set('view engine', 'html')
17 app.set('views', __dirname + '/views')
18 app.use(express.static(public));
19 app.use(favicon(__dirname + '/public/img/favicon.ico'));
20
21 // list all the suppliers
22 app.get('/home', (req, res) => {
23   res.render('home', {});
24 });
25 app.get('/admin/suppliers', supplier.findAll);
26 // show the add supplier form
27 app.get('/admin/supplier-add', supplier.create);
28 // receive the add supplier POST
29 app.post('/admin/supplier-add', supplier.create);
30 // show the update form
31 app.get('/admin/supplier-update/:id', supplier.findOne);
32 // receive the update supplier
33 app.post('/admin/supplier-update', supplier.update);
34 // receive the POST to delete a supplier
35 app.post('/admin/supplier-remove/:id', supplier.remove);
36 // set port, listen for requests
37 const app_port = process.env.APP_PORT || 80
38 app.listen(app_port, () => {
39   console.log(`Microservices is running on port ${app_port}`);
40   app.use(function (req, res, next) {
41     res.status(404).render("404");
42   })
43 }
44
45 // set port, listen for requests
46 const app_port = process.env.APP_PORT || 80
47 app.listen(app_port, () => {
48   console.log(`Microservices is running on port ${app_port}`);
49 })
}

```

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

```

git - ip-10-16-10-27.ec2.x bash - ip-10-16-10-27.ec2 x Immediate Javascript(bro x bash - ip-10-16-10-27.ec2 x
Step 5/7 : RUN npm install
--> c5337b6c053
Step 6/7 : EXPOSE 8081
--> Using cache
--> b5dddc5596
Step 7/7 : CMD ["npm", "run", "start"]
--> Using cache
--> fad7951d8214
Successfully built fad7951d8214
Successfully tagged employee:latest
vscodeshell:~/environment/microservices/employee (dev) $ cd ~/environment/microservices/employee
vscodeshell:~/environment/microservices/employee (dev) $ git add .
vscodeshell:~/environment/microservices/employee (dev) $ git commit -m "Updated code for employee microservice with New Dockerfile and Source Code Changes"
[dev f6fcfbd] Updated code for employee microservice with New Dockerfile and Source Code Changes
1 file changed, 1 insertion(+), 1 deletion(-)
create mode 100644 employee/dockerfile
vscodeshell:~/environment/microservices/employee (dev) $ git push origin dev
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 2 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), done.
Total 15 (delta 14), reused 0 delta, pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 a733596..f6fcfbd dev -> dev
vscodeshell:~/environment/microservices/employee (dev) $

```

We used the below commands to perform the action :

```

cd ~/environment/microservices/employee
git add .

```

20025924

Bhaskara Sai Vamsi Krishna Padala

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.

The screenshot shows the AWS CodeCommit interface. On the left, there's a sidebar with navigation links like 'Source', 'CodeCommit', 'Getting started', 'Repositories', 'Code', 'Pull requests', 'Commits', 'Branches', 'Git tags', 'Settings', 'Approval rule templates', 'Artifacts', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The 'Commits' link is currently selected. The main area displays a table of commits for the 'microservices' repository. The table has columns for Commit ID, Commit message, Commit date, Authored date, Author, Committer, and Actions. There are three commits listed:

Commit ID	Commit message	Commit date	Authored date	Author	Committer	Actions
f6fc6bda	Updated code for employee microservice with New Dockerfile and Source Code Chang...	1 minute ago	1 minute ago	Bhaskara	Bhaskara	Copy ID Browse
a733d967	Updated Code with Docker functionality for Customer Part	2 hours ago	2 hours ago	Bhaskara	Bhaskara	Copy ID Browse
22c9e3c2	two unmodified copies of the application code	6 hours ago	6 hours ago	EC2 Default User	EC2 Default User	Copy ID Browse

The screenshot shows a detailed view of a specific commit in the AWS CodeCommit interface. The commit ID is f6fc6bda9f0f91d45b3f614335613b946aeff8a1. The details page includes sections for 'Details', 'employee/Dockerfile', and 'employee/controller'. The 'employee/Dockerfile' section shows the following Dockerfile content:

```
1 + FROM node:11-alpine
2 + RUN mkdir -p /usr/src/app
3 + WORKDIR /usr/src/app
4 + COPY . .
5 + RUN npm install
6 + EXPOSE 8081
7 + CMD ["npm", "run", "start"]
```

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
```

```
Lab Instructions: Building Microservices - AWS Cloud9 | AWS Cloud9 - AWS Cloud9 | Copy Monolithic Source Code | Sun Apr 28 7:01PM
us-east-1.console.aws.amazon.com/cloud9/file/55faa21049784765ae7788c1bba20531?region=us-east-1

File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (⌘ P)
MicroservicesIDE - /home/ec2-user/environment
└── microservices
    └── latsuser.pem
    └── README.md
Dockerfile Dockerfile index.js
1 FROM node:12-alpine
2 RUN mkdir -p /usr/src/app
3 WORKDIR /usr/src/app
4 COPY . .
5 RUN npm install
6 EXPOSE 4888
7 CMD ["npm", "run", "start"]

aws
aws

export >~< bash ->~< bash ->~< bash ->~< Immediate Javascript Env > bash ->~< bash ->~<
veclabs:~/environment $ account_id=$(aws sts get-caller-identity | grep Account | cut -d '"' -f4)
veclabs:~/environment $ echo $account_id
8005454545454545
veclabs:~/environment $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
veclabs:~/environment $
```

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

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS ECR Create Repository wizard. The first section, 'General settings', includes fields for Repository name (86694343160.dkr.ecr.us-east-1.amazonaws.com/customer) and Tag immutability (set to 'Disabled'). A note states: 'Once a repository has been created, the visibility setting of the repository can't be changed.' The second section, 'Image scan settings', contains a 'Deprecation warning' about ScanOnPush being deprecated. It includes a 'Scan on push' option (disabled) and a note: 'The KMS encryption settings cannot be changed or disabled after the repository has been created.' The third section, 'Encryption settings', shows 'KMS encryption' is disabled. At the bottom, there are 'Cancel' and 'Create repository' buttons.

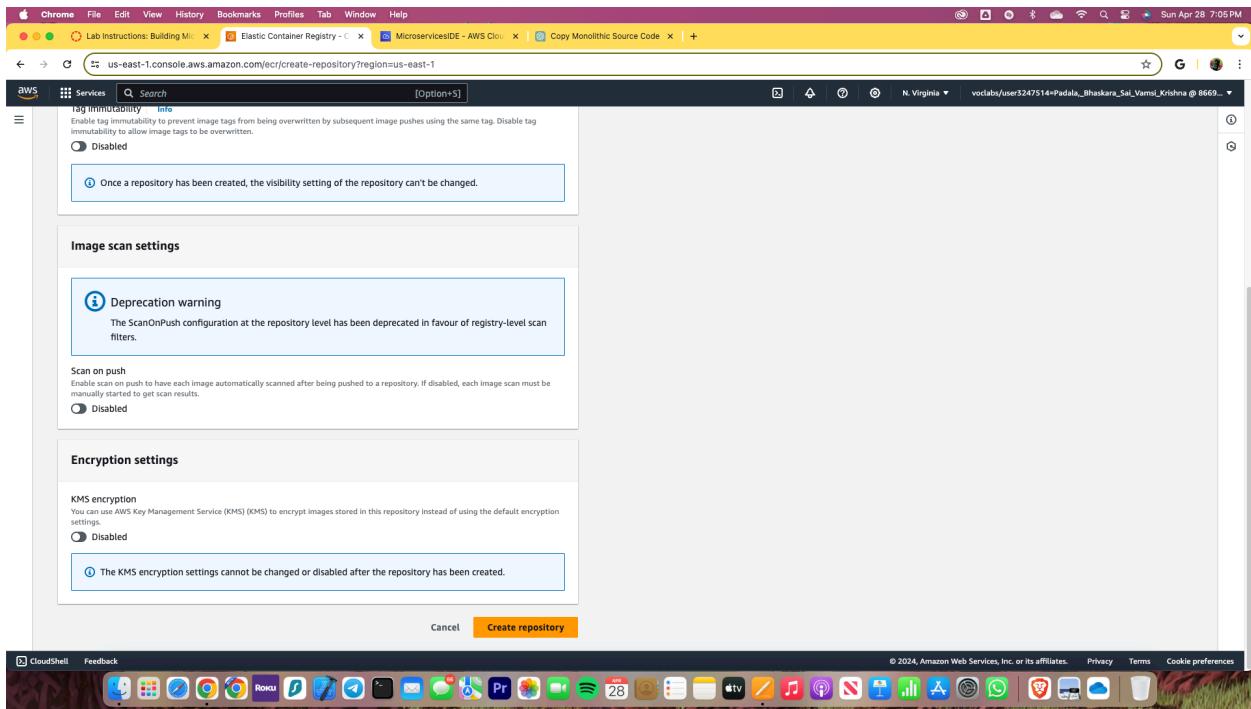
Hence, we successfully created the repository named customer.

The screenshot shows the AWS ECR console with a green success message: "Created private repository customer has been successfully created in private registry". The main page displays a table of repositories, showing one entry: "customer" with a URI of "866964334160.dkr.ecr.us-east-1.amazonaws.com/customer". The table includes columns for Repository name, URI, Created at, Tag immutability, Scan frequency, and Encryption type.

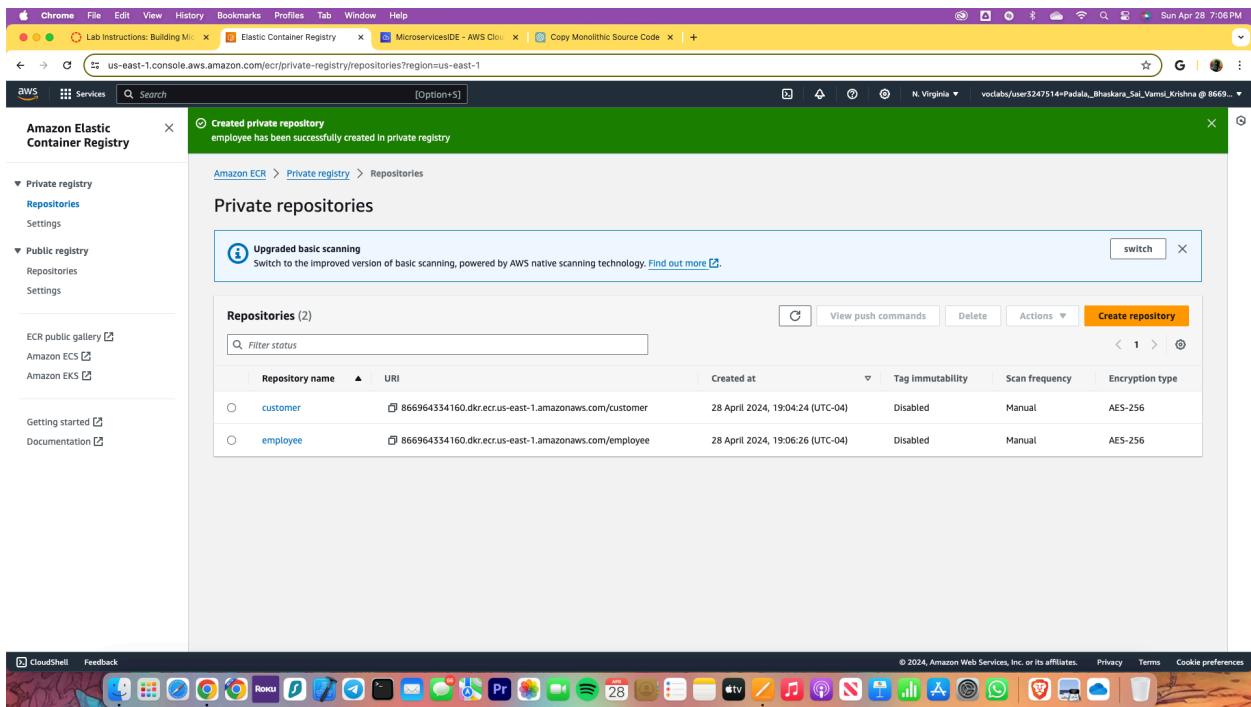
Now we will create the repository named employee.

The screenshot shows the "Create repository" wizard. In the "General settings" section, the visibility setting is set to "Private". The repository name is "employee". Under "Tag immutability", the option "Disabled" is selected. A note states: "Once a repository has been created, the visibility setting of the repository can't be changed." In the "Image scan settings" section, there is a "Deprecation warning" message: "The ScanOnPush configuration at the repository level has been deprecated in favour of registry-level scan settings." The bottom of the screen shows a Mac OS X dock with various application icons.

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.

```

{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "ecr:*"
    }
  ]
}

```

Edit permissions

Statements

No statements
You don't have any permission statement

Add statement

Edit policy JSON Add statement

Save Close

Amazon Elastic Container Registry

Private registry > Repositories > customer > Permissions

Statement 1

Effect: Allow
Principal: *
Actions: ecr:*

Service principals: -
AWS Account IDs: -

Edit policy JSON Edit

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
```

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

```
customer
employee
```

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

```

doker -> `p-10-16-10-27.x bash ->`p-10-16-10-27.ec x Immediate Javascript (x) bash ->`p-10-16-10-27.ec x
vocelabs:~/environment $ account_id=$(aws sts get-caller-identity | grep Account | cut -d ":" -f4)
vocelabs:~/environment $ echo $account_id
866964334168
vocelabs:~/environment $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vocelabs:~/environment $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vocelabs:~/environment $ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
employee           latest   fa07951d0214  3 hours ago   82.7MB
866964334168.dkr.ecr.us-east-1.amazonaws.com/customer   latest   fad7951d0214  3 hours ago   82.7MB
customer          latest   339ebafcc6e6  5 hours ago   82.7MB
node               latest   f18d82f58c3d  5 hours ago   82.7MB
node              11-alpine f18d82f58c3d  4 years ago  75.5MB
vocelabs:~/environment $ account_id=$(aws sts get-caller-identity | grep Account | cut -d ":" -f4)
vocelabs:~/environment $ echo $account_id
866964334168
vocelabs:~/environment $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
vocelabs:~/environment $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
The push refers to repository [866964334168.dkr.ecr.us-east-1.amazonaws.com/customer]
866964334168: Pushed
095da839ee5a: Pushed
c418668a8186: Pushed
cd1d71533807: Pushed
10d73b09944: Pushed
dcaceb729824: Pushed
fb5933fe4b5: Pushed
latest: digest: sha256:16b21a6a7d6226f19129cff35e94ef9186752772542f3826d978057dc6327b9 size: 1783
vocelabs:~/environment $ 

```

Now we successfully pushed both customer and employee

```

doker -> `p-10-16-10-27.x bash ->`p-10-16-10-27.ec x Immediate Javascript (x) bash ->`p-10-16-10-27.ec x
vocelabs:~/environment $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vocelabs:~/environment $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vocelabs:~/environment $ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
employee           latest   fa07951d0214  3 hours ago   82.7MB
866964334168.dkr.ecr.us-east-1.amazonaws.com/customer   latest   fad7951d0214  3 hours ago   82.7MB
customer          latest   339ebafcc6e6  5 hours ago   82.7MB
node               latest   f18d82f58c3d  5 hours ago   82.7MB
node              11-alpine f18d82f58c3d  4 years ago  75.5MB
vocelabs:~/environment $ account_id=$(aws sts get-caller-identity | grep Account | cut -d ":" -f4)
vocelabs:~/environment $ echo $account_id
866964334168
vocelabs:~/environment $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
vocelabs:~/environment $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
The push refers to repository [866964334168.dkr.ecr.us-east-1.amazonaws.com/customer]
866964334168: Pushed
095da839ee5a: Pushed
c418668a8186: Pushed
cd1d71533807: Pushed
10d73b09944: Pushed
dcaceb729824: Pushed
fb5933fe4b5: Pushed
latest: digest: sha256:16b21a6a7d6226f19129cff35e94ef9186752772542f3826d978057dc6327b9 size: 1783
vocelabs:~/environment $ clear
vocelabs:~/environment $ account_id=$(aws sts get-caller-identity | grep Account | cut -d ":" -f4)
vocelabs:~/environment $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
vocelabs:~/environment $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [866964334168.dkr.ecr.us-east-1.amazonaws.com/employee]
853a123ce2c1: Pushed
5c8ccf5735fb: Pushed
c418668a8186: Pushed
cd1d71533807: Pushed
10d73b09944: Pushed
dcaceb729824: Pushed
fb5933fe4b5: Pushed
latest: digest: sha256:75cfec3c78bf277ff1de675e97c4331534c3c34f0e67df849576fad9badd31861 size: 1783
vocelabs:~/environment $ 

```

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS ECR console with the 'customer' repository selected. The left sidebar shows navigation options for 'Private registry' and 'Public registry'. The main content area displays a table titled 'Images (1)' with one entry:

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	28 April 2024, 19:20:28 (UTC-04)	27.70	Copy URI	sha256:16b21a6a7d6226f19129cff35e94ae...

The screenshot shows the AWS ECR console with the 'employee' repository selected. The left sidebar shows navigation options for 'Private registry' and 'Public registry'. The main content area displays a table titled 'Images (1)' with one entry:

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	28 April 2024, 19:22:48 (UTC-04)	27.70	Copy URI	sha256:75fec3c70bf277f1de675e97c4331...

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).

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the 'Create cluster' wizard for the Amazon Elastic Container Service. The 'Cluster configuration' step is active, where the user has specified the cluster name as 'microservices-serverlesscluster'. Under the 'Infrastructure' section, the 'AWS Fargate (serverless)' checkbox is checked, indicating it's the chosen capacity provider. Other options like 'Amazon EC2 Instances' and 'External Instances using ECS Anywhere' are available but not selected.

The screenshot shows the 'Clusters' page for the Amazon Elastic Container Service. A blue banner at the top states 'Cluster microservices-serverlesscluster creation is in progress.' On the right side of the page, there is a prominent orange 'Create cluster' button. The main content area displays a table titled 'Clusters (0)', which currently shows 'No clusters to display'.

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

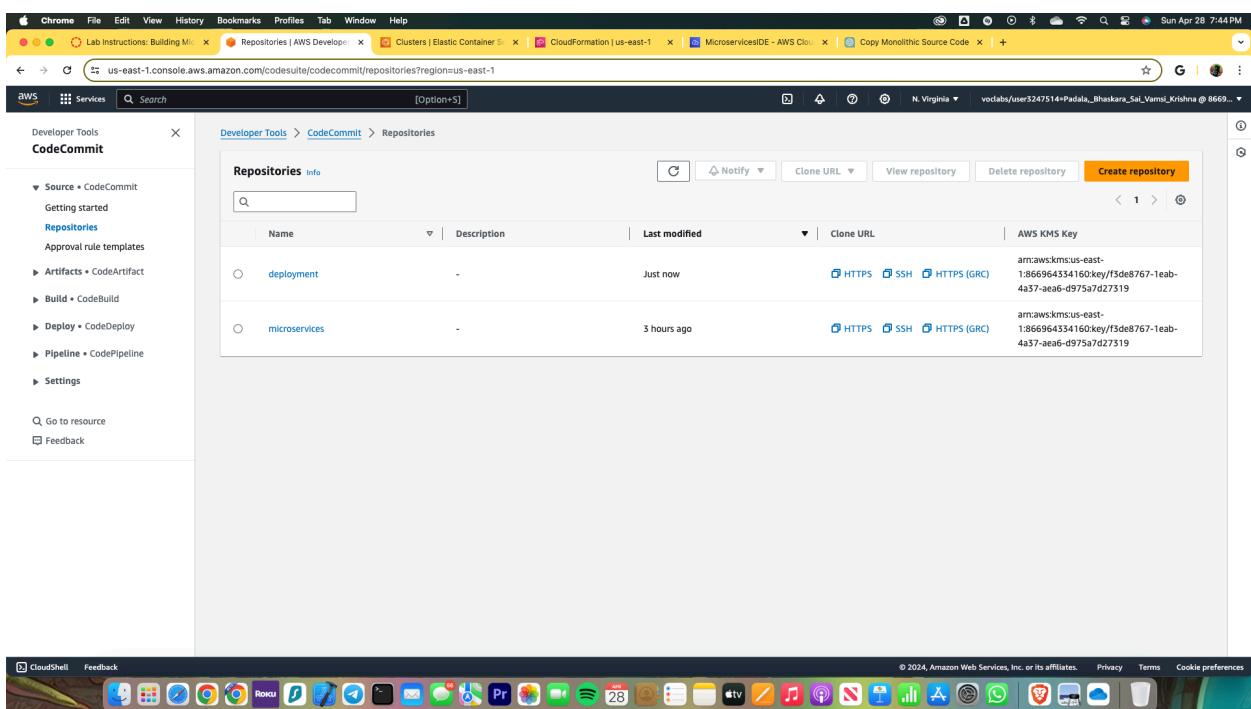
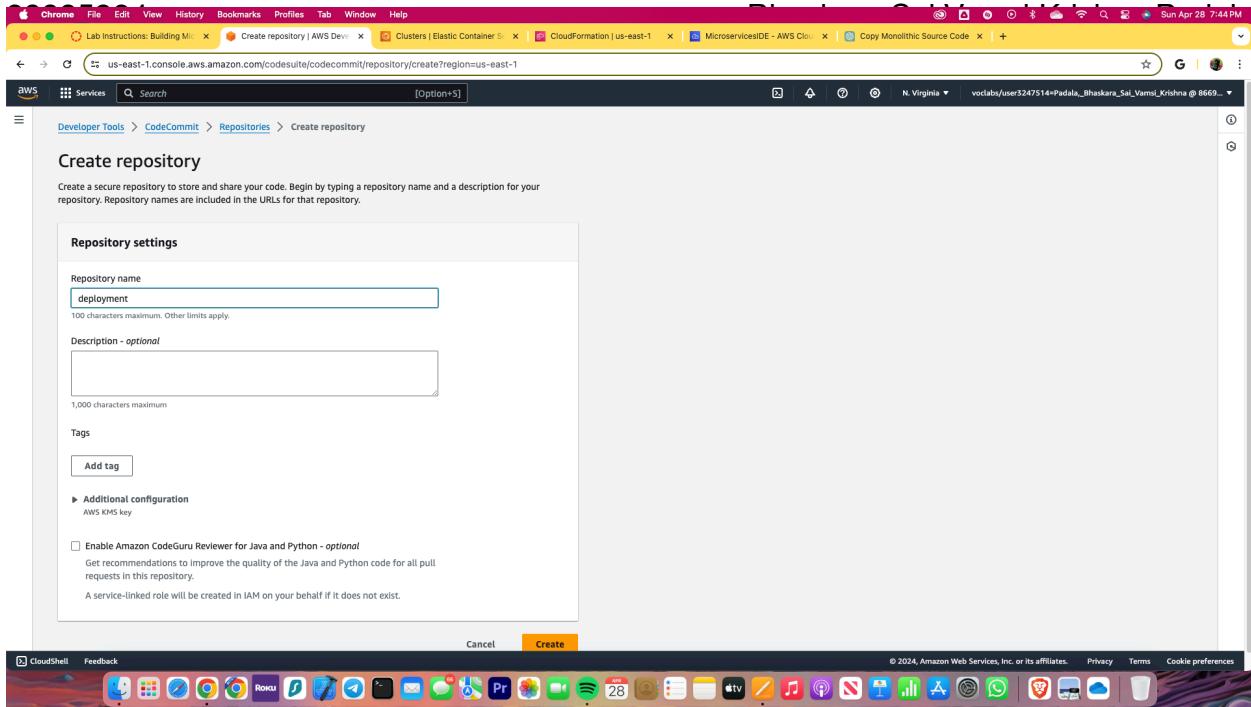
The screenshot shows the AWS Elastic Container Service (ECS) Clusters page. The left sidebar includes links for Clusters, Namespaces, Task definitions, Account settings, Install AWS Copilot, Amazon ECR, Repositories, AWS Batch, Documentation, Discover products, and Subscriptions. The main content area displays a table for the 'Clusters (1) Info' section. The table has columns for Cluster, Services, Tasks, Container instances, CloudWatch monitoring, and Capacity provider strategy. One row is shown for 'microservices-serverlesscluster' with values: 0 services, 0 tasks running, 0 EC2, Default CloudWatch monitoring, and No default found capacity provider strategy. A success message at the top states: 'Cluster microservices-serverlesscluster has been created successfully.' A 'Create cluster' button is visible in the top right corner.

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.

The screenshot shows the 'Create repository' wizard in the AWS CodeCommit Developer Tools. The current step is 'Repository settings'. The form includes fields for 'Repository name' (set to 'deployment'), 'Description - optional' (empty), and 'Tags' (empty). Below the form, there is an 'Additional configuration' section with a checkbox for 'Enable Amazon CodeGuru Reviewer for Java and Python - optional'. The 'Create' button is visible at the bottom right of the form.



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.

```

File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (⌘ F) Dockerfile index.js
MicroservicesIDE - home/ec2-user/environment Dockerfile JS index.js
└── deployment
    └── microservices
        └── labuser.pem
    └── README.md
aws

095da833eaa5: Pushed
c41866e8106: Pushed
db1d7153bb7: Pushed
1dc77bb0994: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:16b21a6a7d6226f19129cff35e94ef9186752772542f3826d978057dc6327b9 size: 1783
vclabs:~/environment $ clear
vclabs:~/environment $ aws lambda update-function-configuration --function-name vclabs-lambda --region us-east-1 | docker login --username AWS --password=$(aws ecr get-login-password --region us-east-1) docker login --username vclabs-lambda --password $(aws ecr get-login-password --region us-east-1)
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
vclabs:~/environment $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to a repository [866964334168.dkr.ecr.us-east-1.amazonaws.com/employee]
3832296: Pushed
5c8cf573b: Pushed
c41866e8106: Pushed
db1d7153bb7: Pushed
1dc77bb0994: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:75fec3c70b72771dd5e907c4301534c3c34f0e67df849576fad9badd31b61 size: 1783
vclabs:~/environment $ cd .. -> environment/deployment
bash: cd: /home/ec2-user/environment/deployment: No such file or directory
vclabs:~/environment $ mkdir deployment
vclabs:~/environment $ cd deployment
vclabs:~/environment/deployment $ git init
vclabs:~/environment/deployment $ git init master
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: future your new repositories, which will suppress this warning, call:
hint:
hint:     git config --global init.defaultBranch <name>
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:     git branch -m <name>
Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
vclabs:~/environment/deployment (master) $ git branch -m dev
vclabs:~/environment/deployment (dev) $ git status
On branch dev
No commits yet
nothing to commit (create/copy files and use "git add" to track)
vclabs:~/environment/deployment (dev) $ 

```

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.

```

File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (⌘ F) Dockerfile taskdef-customer.json
MicroservicesIDE - home/ec2-user/environment Dockerfile JS index.js
└── deployment
    └── microservices
        └── labuser.pem
    └── README.md
aws

vclabs:~/environment $ aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-customer.json"
vclabs:~/environment $ aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-customer.json"
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/customer-microservice:1",
        "containerDefinitions": [
            {
                "name": "customer",
                "image": "customer",
                "cpu": 0,
                "portMappings": [
                    {
                        "containerPort": 8080,
                        "hostPort": 8080,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "environment": [
                    ...
                ]
            }
        ],
        "taskDefinition": {
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/customer-microservice:1",
            "containerDefinitions": [
                {
                    "name": "customer",
                    "image": "customer",
                    "cpu": 0,
                    "portMappings": [
                        {
                            "containerPort": 8080,
                            "hostPort": 8080,
                            "protocol": "tcp"
                        }
                    ],
                    "essential": true,
                    "environment": [
                        ...
                    ]
                }
            ],
            "taskDefinition": {
                "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/customer-microservice:1",
                "containerDefinitions": [
                    ...
                ]
            }
        }
    }
}

```

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"
```

The screenshot shows a Mac OS X desktop with a terminal window and a code editor window.

The terminal window title is "aws" and the command run is:

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

The code editor window title is "taskdef-customer.json" and contains the JSON configuration for the task definition:

```

{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/customer-microservice:1",
    "containerDefinitions": [
      {
        "name": "customer",
        "image": "customer",
        "cpu": 0,
        "portMappings": [
          {
            "containerPort": 8888,
            "hostPort": 8888,
            "protocol": "tcp"
          }
        ],
        "essential": true,
        "environment": [
          ...
        ]
      }
    ],
    "environment": [
      ...
    ]
  }
}

```

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.

The screenshot shows the AWS CloudWatch Metrics interface.

The left sidebar shows the navigation path: "Amazon Elastic Container Service > Task definitions".

The main pane displays the "Task definitions (1) info" section. There is one task definition listed:

- Task definition:** customer-microservice
- Status of last revision:** ACTIVE

At the top right of the interface, there is a yellow button labeled "Create new revision".

```

1  {
2    "containerDefinitions": [
3      {
4        "name": "employee",
5        "image": "employee",
6        "environment": [
7          {
8            "name": "APP_DB_HOST",
9            "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
10           }
11         ],
12        "essential": true,
13        "portMappings": [
14          {
15            "hostPort": 8080,
16            "protocol": "tcp",
17            "containerPort": 8080
18          }
19        ],
20        "logConfiguration": {
21          "logDriver": "awslogs",
22          "options": {
23            "awslogs-create-group": "true",
24            "awslogs-group": "employee-capstone",
25            "awslogs-region": "us-east-1",
26            "awslogs-stream-prefix": "employee-capstone"
27          }
28        }
29      }
30    ],
31    "requiresCompatibilities": [
32      "FARGATE"
33    ],
34    "networkMode": "awsvpc",
35    "cpu": "12vCPU",
36    "memory": "30GB",
37    "executionRoleArn": "arn:aws:iam::866964334168:role/PipelineRole",
38    "family": "employee-microservice"
39  }
40 }

"taskDefinition": {
  "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
  "containerDefinitions": [
    {
      "name": "employee",
      "image": "employee",
      "cpu": 8,
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 8080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "environment": [
        {
          "name": "APP_DB_HOST",
          "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "employee-capstone",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "employee-capstone"
        }
      }
    ]
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "12vCPU",
  "memory": "30GB",
  "executionRoleArn": "arn:aws:iam::866964334168:role/PipelineRole",
  "family": "employee-microservice"
}
41

```

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.

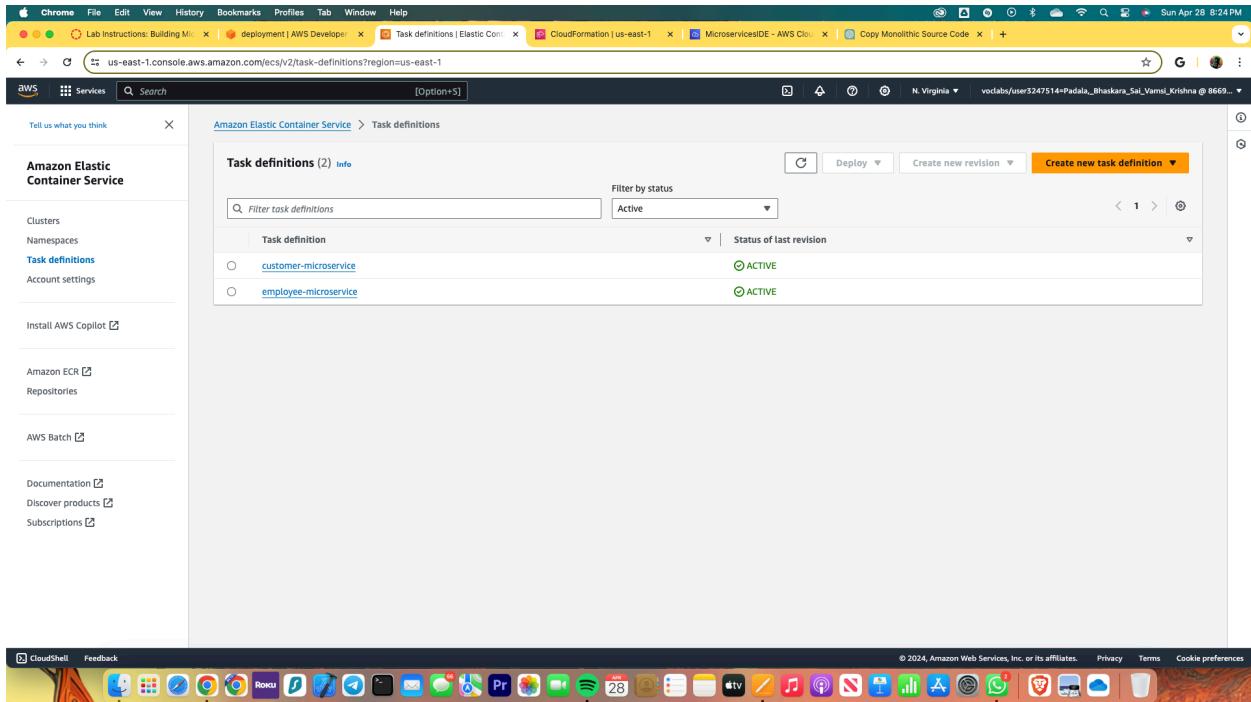
In the Amazon ECS console, we need to verify that the employee-microservice task definition now appears in the Task definitions pane. Also, we notice that the revision number displays after the task definition name.

```

1  {
2    "containerDefinitions": [
3      {
4        "name": "employee",
5        "image": "employee",
6        "environment": [
7          {
8            "name": "APP_DB_HOST",
9            "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
10           }
11         ],
12        "essential": true,
13        "portMappings": [
14          {
15            "hostPort": 8080,
16            "protocol": "tcp",
17            "containerPort": 8080
18          }
19        ],
20        "logConfiguration": {
21          "logDriver": "awslogs",
22          "options": {
23            "awslogs-create-group": "true",
24            "awslogs-group": "employee-capstone",
25            "awslogs-region": "us-east-1",
26            "awslogs-stream-prefix": "employee-capstone"
27          }
28        }
29      }
30    ],
31    "requiresCompatibilities": [
32      "FARGATE"
33    ],
34    "networkMode": "awsvpc",
35    "cpu": "12vCPU",
36    "memory": "30GB",
37    "executionRoleArn": "arn:aws:iam::866964334168:role/PipelineRole",
38    "family": "employee-microservice"
39  }
40 }

"taskDefinition": {
  "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
  "containerDefinitions": [
    {
      "name": "employee",
      "image": "employee",
      "cpu": 8,
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 8080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "environment": [
        {
          "name": "APP_DB_HOST",
          "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "employee-capstone",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "employee-capstone"
        }
      }
    ]
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "12vCPU",
  "memory": "30GB",
  "executionRoleArn": "arn:aws:iam::866964334168:role/PipelineRole",
  "family": "employee-microservice"
}
41

```

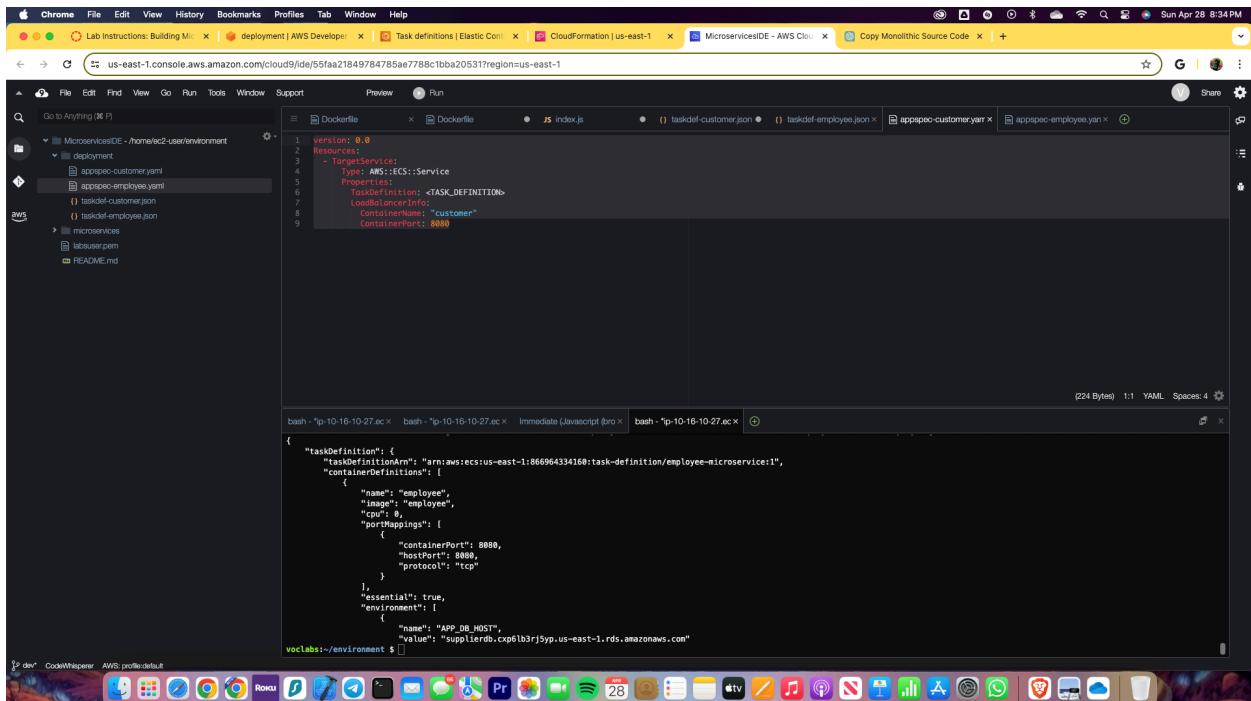


Task 5.5: Create AppSpec files for CodeDeploy for each microservice

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`



```

version: 0.0
Resources:
  - Type: AWS::ECS::Service
    Properties:
      TaskDefinitionArn: <TASK_DEFINITION>
      DesiredCount: 1
      LoadBalancersInfo:
        ContainerName: "employee"
        ContainerPort: 8080
  
```

```

{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
    "containerDefinitions": [
      {
        "name": "employee",
        "image": <IMAGE_NAME>,
        "environment": [
          {
            "name": "APP_DB_HOST",
            "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
          }
        ],
        "essential": true,
        "portMappings": [
          {
            "hostPort": 8080,
            "hostPortT": 8080,
            "protocol": "tcp"
          }
        ],
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": [
            {
              "awslogs-create-group": "true",
              "awslogs-group": "omelogs-capstone",
              "awslogs-region": "us-east-1",
              "awslogs-stream-prefix": "omelogs-capstone"
            }
          ]
        }
      }
    ],
    "taskDefinition": {
      "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
      "containerDefinitions": [
        {
          "name": "employee",
          "image": <IMAGE_NAME>,
          "environment": [
            {
              "name": "APP_DB_HOST",
              "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
            }
          ],
          "portMappings": [
            {
              "hostPort": 8080,
              "hostPortT": 8080,
              "protocol": "tcp"
            }
          ],
          "essential": true,
          "environment": [
            {
              "name": "APP_DB_HOST",
              "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
            }
          ]
        }
      ]
    }
  }
}
  
```

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 the containerNameline to be employee`

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.

```

version: 0.0
Resources:
  - Type: AWS::ECS::Service
    Properties:
      TaskDefinitionArn: <TASK_DEFINITION>
      DesiredCount: 1
      LoadBalancersInfo:
        ContainerName: "customer"
        ContainerPort: 8080
  
```

```

{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
    "containerDefinitions": [
      {
        "name": "customer",
        "image": <IMAGE_NAME>,
        "environment": [
          {
            "name": "APP_DB_HOST",
            "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
          }
        ],
        "essential": true,
        "portMappings": [
          {
            "hostPort": 8080,
            "protocol": "tcp",
            "containerPort": 8080
          }
        ],
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": [
            {
              "awslogs-create-group": "true",
              "awslogs-group": "omelogs-capstone",
              "awslogs-region": "us-east-1",
              "awslogs-stream-prefix": "omelogs-capstone"
            }
          ]
        }
      }
    ],
    "taskDefinition": {
      "taskDefinitionArn": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:1",
      "containerDefinitions": [
        {
          "name": "employee",
          "image": <IMAGE_NAME>,
          "environment": [
            {
              "name": "APP_DB_HOST",
              "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
            }
          ],
          "portMappings": [
            {
              "hostPort": 8080,
              "hostPortT": 8080,
              "protocol": "tcp"
            }
          ],
          "essential": true,
          "environment": [
            {
              "name": "APP_DB_HOST",
              "value": "supplierdb.cxp6lb3rj5yp.us-east-1.rds.amazonaws.com"
            }
          ]
        }
      ]
    }
  }
}
  
```

```

1   {
2     "containerDefinitions": [
3       {
4         "name": "employee",
5         "image": "<IMAGE_NAME>",
6         "environment": [
7           {
8             "name": "APP_DB_HOST",
9             "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
10            }
11          ],
12        "essential": true,
13        "portMappings": [
14          {
15            "hostPort": 8080,
16            "protocol": "tcp",
17            "containerPort": 8080
18          }
19        ],
20        "logConfiguration": {
21          "options": {
22            "awslogs-create-group": "true",
23            "awslogs-group": "awslogs-capstone",
24            "awslogs-region": "us-east-1",
25            "awslogs-stream-prefix": "awslogs-capstone"
26          }
27        }
28      }
29    ],
30    "taskDefinition": {
31      "taskDefinitionArn": "arn:aws:ecs:us-east-1:866904334108:task-definition/employee-microservice:1",
32      "containerDefinitions": [
33        {
34          "name": "employee",
35          "image": "<IMAGE_NAME>",
36          "cpu": 8,
37          "portMappings": [
38            {
39              "containerPort": 8080,
40              "hostPort": 8080,
41              "protocol": "tcp"
42            }
43          ],
44          "essential": true,
45          "environment": [
46            {
47              "name": "APP_DB_HOST",
48              "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
49            }
50          ]
51        }
52      ],
53      "essential": true,
54      "environment": [
55        {
56          "name": "APP_DB_HOST",
57          "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
58        }
59      ]
60    }
61  }
62
63  vclabs:~/environment $ 

```

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.

```

1   {
2     "containerDefinitions": [
3       {
4         "name": "customer",
5         "image": "<IMAGE_NAME>",
6         "environment": [
7           {
8             "name": "APP_DB_HOST",
9             "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
10            }
11          ],
12        "essential": true,
13        "environment": [
14          {
15            "name": "APP_DB_HOST",
16            "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
17          }
18        ]
19      }
20    ],
21    "taskDefinition": {
22      "taskDefinitionArn": "arn:aws:ecs:us-east-1:866904334108:task-definition/customer-microservice:1",
23      "containerDefinitions": [
24        {
25          "name": "customer",
26          "image": "<IMAGE_NAME>",
27          "cpu": 8,
28          "portMappings": [
29            {
30              "containerPort": 8080,
31              "hostPort": 8080,
32              "protocol": "tcp"
33            }
34          ],
35          "essential": true,
36          "environment": [
37            {
38              "name": "APP_DB_HOST",
39              "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
40            }
41          ]
42        }
43      ],
44      "essential": true,
45      "environment": [
46        {
47          "name": "APP_DB_HOST",
48          "value": "supplierdb.cxp6l3rj5yp.us-east-1.rds.amazonaws.com"
49        }
50      ]
51    }
52  }
53
54  vclabs:~/environment $ cd ~/environment/deployment
55 vclabs:~/environment/deployment (dev) $ git add .
56 vclabs:~/environment/deployment (dev) $ git commit -m "We Set image field to IMAGE1_NAME in task definition files"
57 [dev (root-commit) 82715b0] Set image field to IMAGE1_NAME in task definition files
58 4 files changed, 12 insertions(+)
59 create mode 100644 appspec-customer.yaml
59 create mode 100644 appspec-employee.yaml
59 create mode 100644 taskdef-customer.json
59 create mode 100644 taskdef-employee.json
60 vclabs:~/environment/deployment (dev) $ git push origin dev
61 fatal: 'origin' does not appear to be a git repository
62 fatal: Could not read from remote repository.
63
64 Please make sure you have the correct access rights
65 and the repository exists.
66 vclabs:~/environment/deployment (dev) $ cd ~/environment/deployment
67 vclabs:~/environment/deployment (dev) $ git remote -v
68 vclabs:~/environment/deployment (dev) $ git push origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
69 origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment (fetch)
70 origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment (push)
71 vclabs:~/environment/deployment (dev) $ git push origin dev
72 Enumerating objects: 6, done.
73 Counting objects: 100% (6/6), done.
74 Delta compression using up to 2 threads
75 Compressing objects: 100% (6/6), done.
76 Writing objects: 100% (6/6), 1.05 KB | 1.05 MB/s, done.
77 Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
78
79 To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
80  * [new branch]    dev => dev
81 vclabs:~/environment/deployment (dev) $ git commit -m "We Set image field to IMAGE1_NAME in task definition files"
82 On branch dev
83 nothing to commit, working tree clean
84 vclabs:~/environment/deployment (dev) $ 

```

Phase 6: Creating target groups and an Application Load Balancer

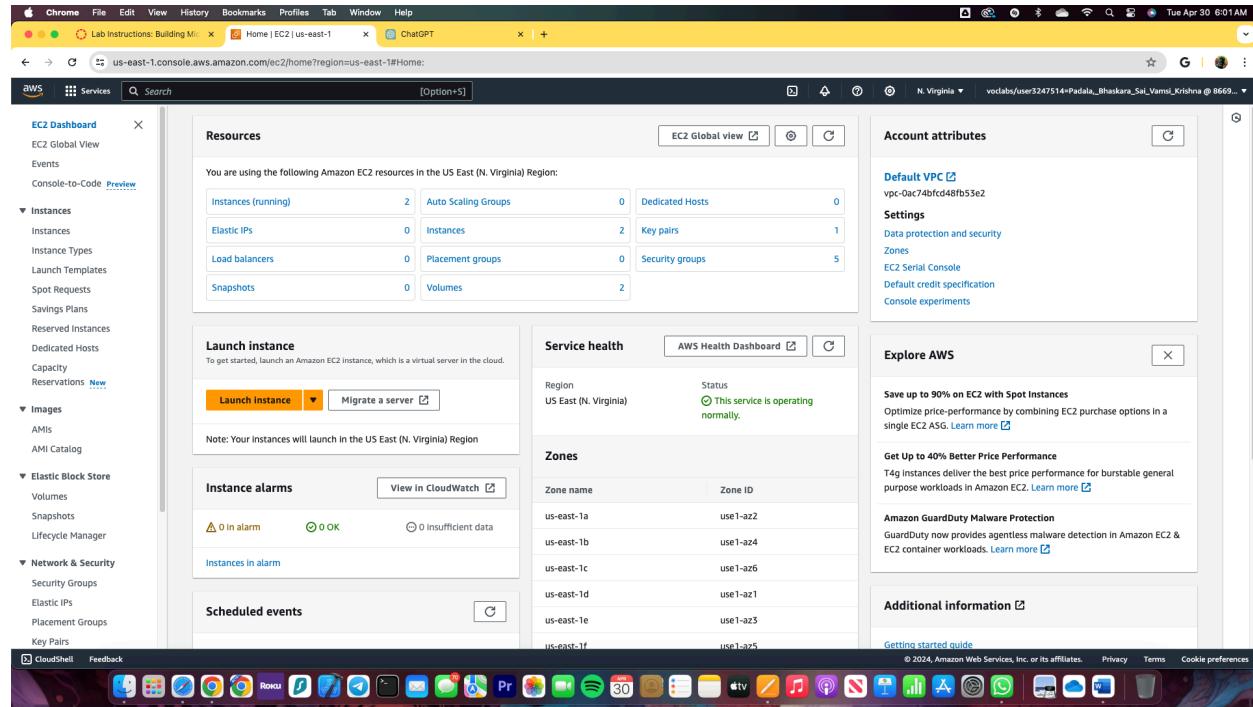
In this phase, we will create an Application Load Balancer, which provides an endpoint URL. This URL will act as the HTTPS entry point for customers and employees to access our application through a web browser.

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.

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 IPv4 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

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 8080 8080

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-089e8c87b4fc2f9bb
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

Health check path

/

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.

Next

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS Cloud9 IDE interface. The code editor displays a Python script for a Lambda function:

```
def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': "Hello World"
    }
```

The Lambda function configuration includes:

- Memory: 128 MB
- Timeout: 3 seconds
- Runtime: aws_lambda_python3
- Execution Role: LambdaBasicExecutionRole
- HTTP API Endpoint: https://[REDACTED].execute-api.us-east-1.amazonaws.com/

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

Lab Instructions: Building Microservices with AWS Step 1 Create target group | 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

aaws Services Search [Option+S]

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

CloudShell Feedback

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	Protocol	Protocol version
IP	HTTP: 8080	HTTP1
IP address type	Load balancer <u>None associated</u>	VPC vpc-089ece87b4fc2f9bb
0 Total targets	0 Healthy 0 Unhealthy 0 Anomalous	0 Unused 0 Initial 0 Draining

Targets Monitoring Health checks Attributes Tags

Registered targets (0) Info

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

Anomaly mitigation: Not applicable

Deregister Register targets

1 >

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 6:06 AM

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Lab Instructions: Building Microservices with AWS Step 1 Create target group | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:

aaws Services Search [Option+S]

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 IPv4 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 Balance

- 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

Tue Apr 30 6:08 AM

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Lab Instructions: Building Microservices with AWS Step 1 Create target group | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:

aaws Services Search [Option+S]

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

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

HTTP2

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

gRPC

Sends 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

Tue Apr 30 6:09 AM

Step 1: Create target group

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

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: /
 Up to 1024 characters allowed.

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.

Next

Step 2: Create target group

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-089ecc87b4fc2f9bb
 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.

 You can add up to 4 more IP addresses.

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

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 IPv4 address	Health status	IP address	Port	Zone
No IP addresses included yet Specify IP addresses above and add to list.				

0 pending

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

customer-tg-two

Details

arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-two/9140a4ede74581ec

Target type	Protocol : Port	Protocol version	VPC
IP	HTTP: 8080	HTTP1	vpc-089ece87b4fc2f9bb
IP address type	Load balancer		
IPv4	None associated		

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
0	0	0	0	0	0
	0 Anomalous				

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (0) Info

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.

Anomaly mitigation: Not applicable

Actions | **Create target group**

Target groups (2) Info

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-two/9140a4ede74581ec	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-one/9140a4ede74581ec	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb

0 target groups selected

Select a target group above.

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:

Lab Instructions: Building Microservices with AWS Step 1 Create target group | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:

Specify group details

Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Basic configuration

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-069e8c57d4f29bb
 IPv4 CIDR: 10.16.0.0/16

Protocol version

HTTP 1.1 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

Details

arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-one/d1488c0d4e046877

Targets

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.

employee-tg-one

Details

Target type	Protocol : Port	Protocol version	VPC
IP	HTTP: 8080	HTTP1	vpc-089ece87b4fc2f9bb
IP address type	Load balancer	<input type="radio"/> None associated	
IPv4			

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
0	0	0	0	0	0
	0 Anomalous				

Registered targets (0)

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.

Actions

Lab Instructions: Building Microservices with AWS Step 1 Create target group | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:

aws Services Search [Option+S]

EC2 > Target groups > Create target group

Step 1 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

Lab Instructions: Building Microservices Step 1 Create target group | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:

aws Services Search [Option+S]

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

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

Health checks

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

Health check protocol

HTTP

Health check path

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

/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

Lab Instructions: Building Microservices Target groups | EC2 us-east-1 | ChatGPT

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#TargetGroups:

aws Services Search [Option+S]

EC2 > Target groups

Target groups (4) Info

Filter target groups

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
employee-tg-two	arn:aws:elasticloadbalancing:us-east-1:866914524751:targetgroup/employee-tg-two/8080	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb
employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:866914524751:targetgroup/employee-tg-one/8080	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:866914524751:targetgroup/customer-tg-one/8080	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:866914524751:targetgroup/customer-tg-two/8080	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb

0 target groups selected

Select a target group above.

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.

Create security group

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

Basic details

Security group name [Info](#)
microservices-sg
Name cannot be edited after creation.

Description [Info](#)
Security group for microservices, allowing traffic on ports 80 and 8080.

VPC [Info](#)
vpc-089ece87b4fc2f9bb (LabVPC)

Inbound rules

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

Add rule

Description

A description to help you identify the security group.

Constraint

The description can be up to 255 characters long. Valid characters for EC2-Classic security groups include all ASCII characters. Valid characters for VPC security groups include a-z, A-Z, 0-9, spaces, and _~!@#\$%^&{}+=;,.{}!\$^&{}

Learn more [Info](#)

Create a security group
Security group rules

Outbound rules

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

Add rule

⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags

Create security group

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS EC2 Security Groups console. A modal window displays a success message: "Security group (sg-0f00ae540db3769c8 | microservices-sg) was created successfully". The main page shows the newly created security group "sg-0f00ae540db3769c8 - microservices-sg". The "Details" section provides basic information: Security group name is "microservices-sg", Security group ID is "sg-0f00ae540db3769c8", Description is "Security group for microservices, allowing traffic on ports 80 and 8080.", Owner is "866964334160", Inbound rules count is "2 Permission entries", and Outbound rules count is "1 Permission entry". The "Inbound rules" tab is selected, showing two rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0539e7bc0d31ae87a	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-
-	sgr-07e4d5d05e068a2...	IPv4	HTTP	TCP	80	0.0.0.0/0	-

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.

The screenshot shows the "Create Application Load Balancer" wizard. The current step is "Basic configuration". The form fields are as follows:

- Load balancer name:** microservicesLB
- Scheme:** Internet-facing (radio button selected)
- IP address type:** IPv4 (radio button selected)

Below the form, there is a "Network mapping" section with a note: "This load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings."

Lab Instructions: Building Microservices with AWS Step 1: Create an Application Load Balancer

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateALBWizard:

aws Services Search [Option+S]

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-089ce87b4fc2f9bb
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-0f81de1a195dc5d31 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](#).

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 7:07 AM

Lab Instructions: Building Microservices with AWS Step 1: Create an Application Load Balancer

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateALBWizard:

aws Services Search [Option+S]

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-02782b56d0d5c8e907 VPC: vpc-089ce87b4fc2f9bb

microservices-sg
sg-0f00aa540d51769c8 VPC: vpc-089ce87b4fc2f9bb

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

Protocol	Port	Default action
HTTP	: 80	Forward to customer-tg-two Target type: IP IPv4

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.

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 7:07 AM

Lab Instructions: Building Microservices with AWS Step 1: Create an Application Load Balancer

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateALBWizard:

aws Services Search [Option+S]

You can add up to 50 more tags.

▼ Listener HTTP:8080

Protocol	Port	Default action
HTTP	: 8080	Forward to customer-tg-one Target type: IP IPv4

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

Include WAF security protections behind the load balancer

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 7:07 AM

Origin

Origin Protocol Policy: HTTP Only

Default Cache Behavior

- Forwarded Headers: None

HTTP Status	HTTP Response
404	/error/404.html
502	/error/502.html

Next Step

Origin

Origin Protocol Policy: HTTP Only

Default Cache Behavior

- Forwarded Headers: None

HTTP Status	HTTP Response
404	/error/404.html
502	/error/502.html

Next Step

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.

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS CloudWatch Metrics Insights interface. A search query 'aws.ec2.*' has been entered. The results list various metrics from different regions and instance types, such as 'cpu_utilization', 'mem_utilization', and 'net_outbound_bandwidth'. Each result includes a preview of the metric data and a 'View details' link.

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.

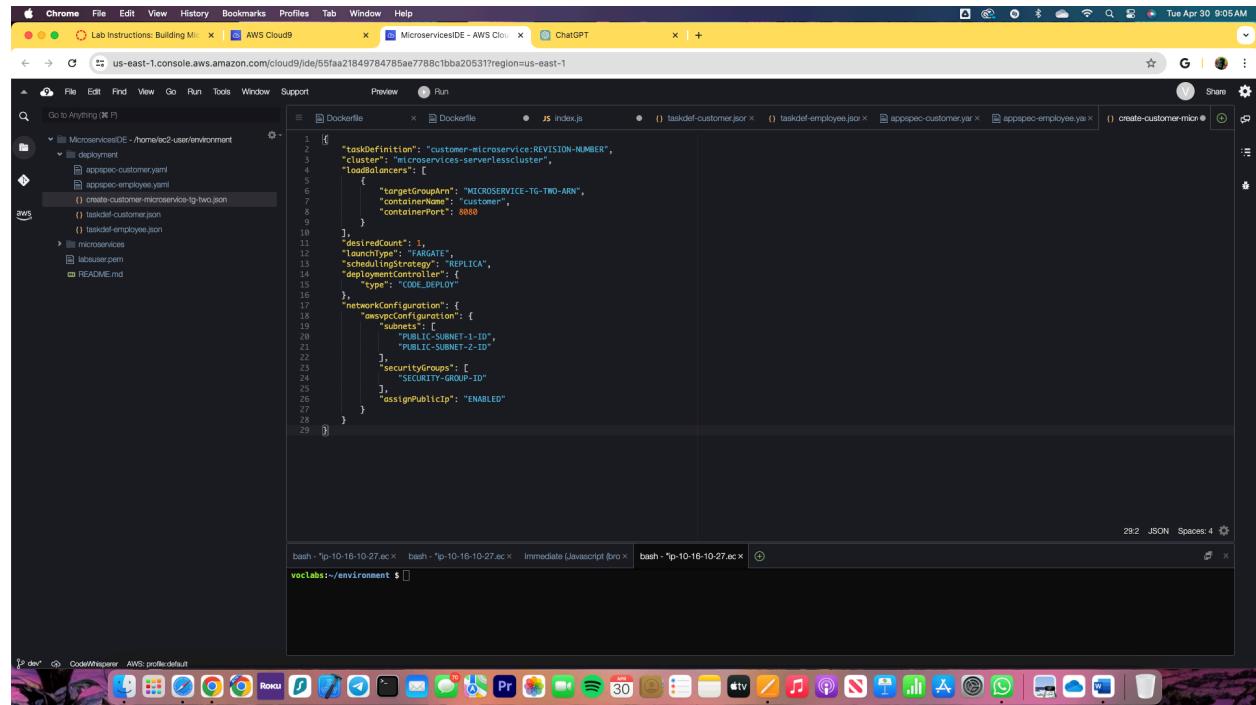
The screenshot shows the AWS CloudWatch Metrics Insights interface. A search query 'aws.ec2.*' has been entered. The results list various metrics from different regions and instance types, such as 'cpu_utilization', 'mem_utilization', and 'net_outbound_bandwidth'. Each result includes a preview of the metric data and a 'View details' link.

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:

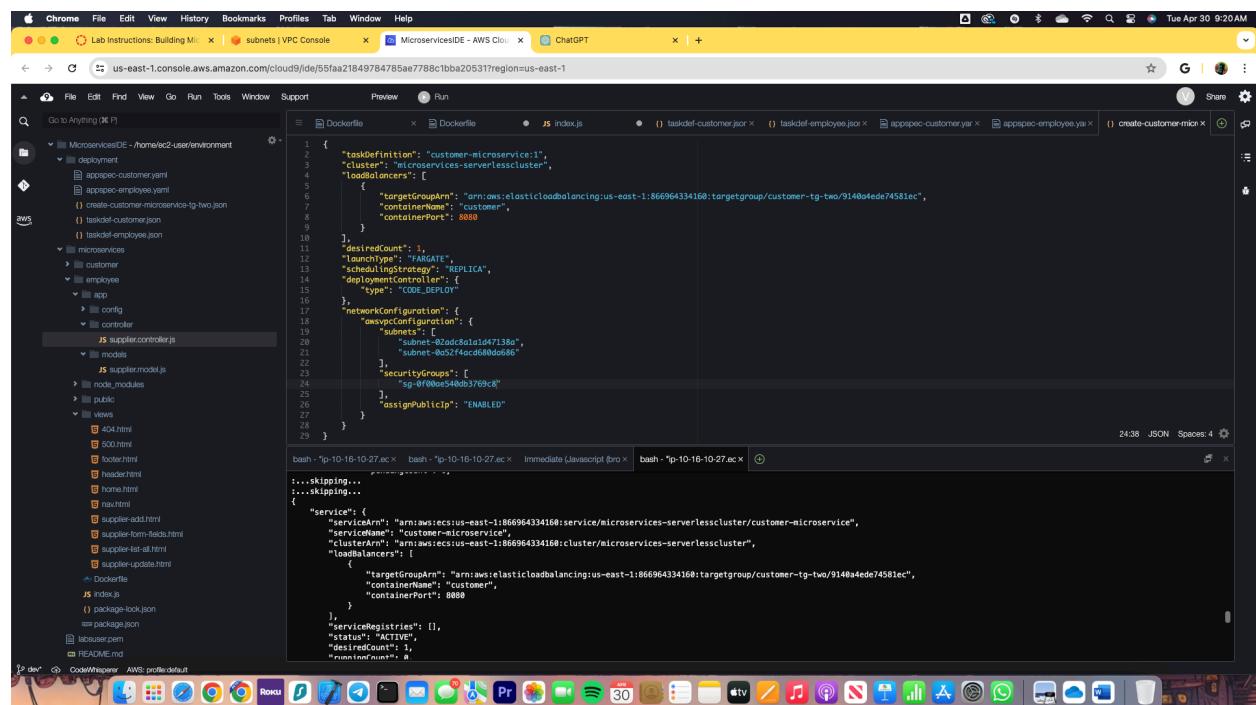


```

1  {
2    "taskDefinition": "customer-microservice:REVISION-NUMBER",
3    "cluster": "microservices-serverlesscluster",
4    "loadBalancers": [
5      {
6        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-two/914ba4ede74581ec",
7        "containerName": "CUSTOMER",
8        "containerPort": 8088
9      }
10   ],
11   "desiredCount": 1,
12   "launchType": "FARGATE",
13   "schedulingStrategy": "REPLICAS",
14   "deploymentController": {
15     "type": "CODE_DEPLOY"
16   },
17   "networkConfiguration": {
18     "awsvpcConfiguration": {
19       "subnets": [
20         "PUBLIC-SUBNET-1-ID",
21         "PUBLIC-SUBNET-2-ID"
22       ],
23       "securityGroups": [
24         "SECURITY-GROUP-ID"
25       ],
26       "assignPublicIp": "ENABLED"
27     }
28   }
29 }

```

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



```

1  {
2    "taskDefinition": "customer-microservice:1",
3    "cluster": "microservices-serverlesscluster",
4    "loadBalancers": [
5      {
6        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-two/914ba4ede74581ec",
7        "containerName": "CUSTOMER",
8        "containerPort": 8088
9      }
10   ],
11   "desiredCount": 1,
12   "launchType": "FARGATE",
13   "schedulingStrategy": "REPLICAS",
14   "deploymentController": {
15     "type": "CODE_DEPLOY"
16   },
17   "networkConfiguration": {
18     "awsvpcConfiguration": {
19       "subnets": [
20         "subnet-02d0c101d47138a",
21         "subnet-0552f4cd680d0686",
22       ],
23       "securityGroups": [
24         "sg-0f00ae54db379cc",
25       ],
26       "assignPublicIp": "ENABLED"
27     }
28   }
29 }

...skipping...
...skipping...

{
  "service": {
    "serviceArn": "arn:aws:secrets:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceName": "Customer-Microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      {
        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/customer-tg-two/914ba4ede74581ec",
        "containerName": "CUSTOMER",
        "containerPort": 8088
      }
    ],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 1,
    "runningCount": 1
  }
}

```

```

bash -> p-10-18-10-27.ec <- bash -> p-10-18-10-27.ec <- Immediate Javascript Bro <- bash -> p-10-18-10-27.ec <- vclabs:-/environment/deployment (dev) $ aws ecs create-service --service-name customer-microservice --cli-input-json file://create-employee-microservice-tg-two.json
{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceArn": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceArn": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceArn": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceArn": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "serviceName": "arn:aws:ecs:us-east-1:866964334160:service/microservices-serverlesscluster/customer-microservice",
    "serviceArn": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

```

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.

```

bash -> p-10-18-10-27.ec <- bash -> p-10-18-10-27.ec <- Immediate Javascript Bro <- bash -> p-10-18-10-27.ec <- vclabs:-/environment/deployment (dev) $ aws ecs create-service --service-name employee-microservice --cli-input-json file://create-employee-microservice-tg-two.json
{
  "service": {
    "taskDefinition": "employee-microservice",
    "cluster": "microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "taskDefinition": "employee-microservice",
    "cluster": "microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "taskDefinition": "employee-microservice",
    "cluster": "microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "taskDefinition": "employee-microservice",
    "cluster": "microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

...skipping...
...skipping...

{
  "service": {
    "taskDefinition": "employee-microservice",
    "cluster": "microservices-serverlesscluster",
    "loadBalancers": [
      ...
    ],
    ...
  }
}

```

```

create-employee-microservice-tg-two.json
{
    "service": {
        "serviceName": "arn:aws:secretsmanager:us-east-1:866964334160:service/microservices-serverlesscluster/employee-microservice",
        "serviceArn": "arn:aws:lambda:us-east-1:866964334160:layer/microservices-serverlesscluster",
        "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-two/33fe164b9332bd13",
                "containerName": "employee",
                "containerPort": 8888
            }
        ],
        "serviceRegistries": [],
        "...skipping..."
    },
    "...skipping..."
}
{
    "service": {
        "serviceName": "arn:aws:secretsmanager:us-east-1:866964334160:service/microservices-serverlesscluster/employee-microservice",
        "serviceArn": "arn:aws:lambda:us-east-1:866964334160:layer/microservices-serverlesscluster",
        "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-two/33fe164b9332bd13",
                "containerName": "employee",
                "containerPort": 8888
            }
        ],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "...skipping..."
    },
    "...skipping..."
}
{
    "service": {
        "serviceName": "arn:aws:secretsmanager:us-east-1:866964334160:service/microservices-serverlesscluster/employee-microservice",
        "serviceArn": "arn:aws:lambda:us-east-1:866964334160:layer/microservices-serverlesscluster",
        "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-two/33fe164b9332bd13",
                "containerName": "employee",
                "containerPort": 8888
            }
        ],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 1,
        "...skipping..."
    },
    "...skipping..."
}
{
    "service": {
        "serviceName": "arn:aws:secretsmanager:us-east-1:866964334160:service/microservices-serverlesscluster/employee-microservice",
        "serviceArn": "arn:aws:lambda:us-east-1:866964334160:layer/microservices-serverlesscluster",
        "clusterArn": "arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334160:targetgroup/employee-tg-two/33fe164b9332bd13",
                "containerName": "employee",
                "containerPort": 8888
            }
        ],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 1,
        "...skipping..."
    },
    "...skipping..."
}

```

Amazon Elastic Container Service

Clusters

- NAMESPACES
- Task definitions
- Account settings

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

microservices-serverlesscluster

Cluster overview

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster	Active	Default	-

Services

Draining	Active	Pending	Running
-	2	-	-

Services (2) Info

Filter launch type	Filter service type					
Any launch type	Any service type					
Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition
customer-microservice	arn:aws:...	Active	REPLICA	0/1 tasks running	-	customer-microservice...
employee-microservice	arn:aws:...	Active	REPLICA	0/1 tasks running	-	employee-microservice...

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.

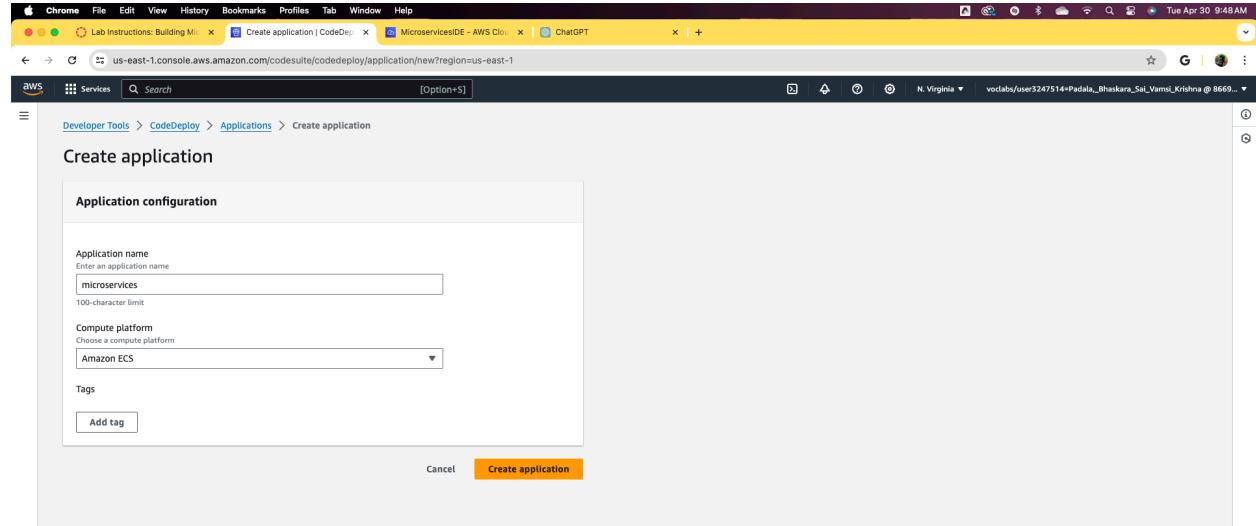
In the ECS context, it's akin to laying out the groundwork before actual tasks start running. This phase involves setting up task definitions, creating task sets, and eventually launching tasks. Once tasks are initiated, you can monitor their status and manage them through the ECS console.

Phase 8: Configuring CodeDeploy and CodePipeline

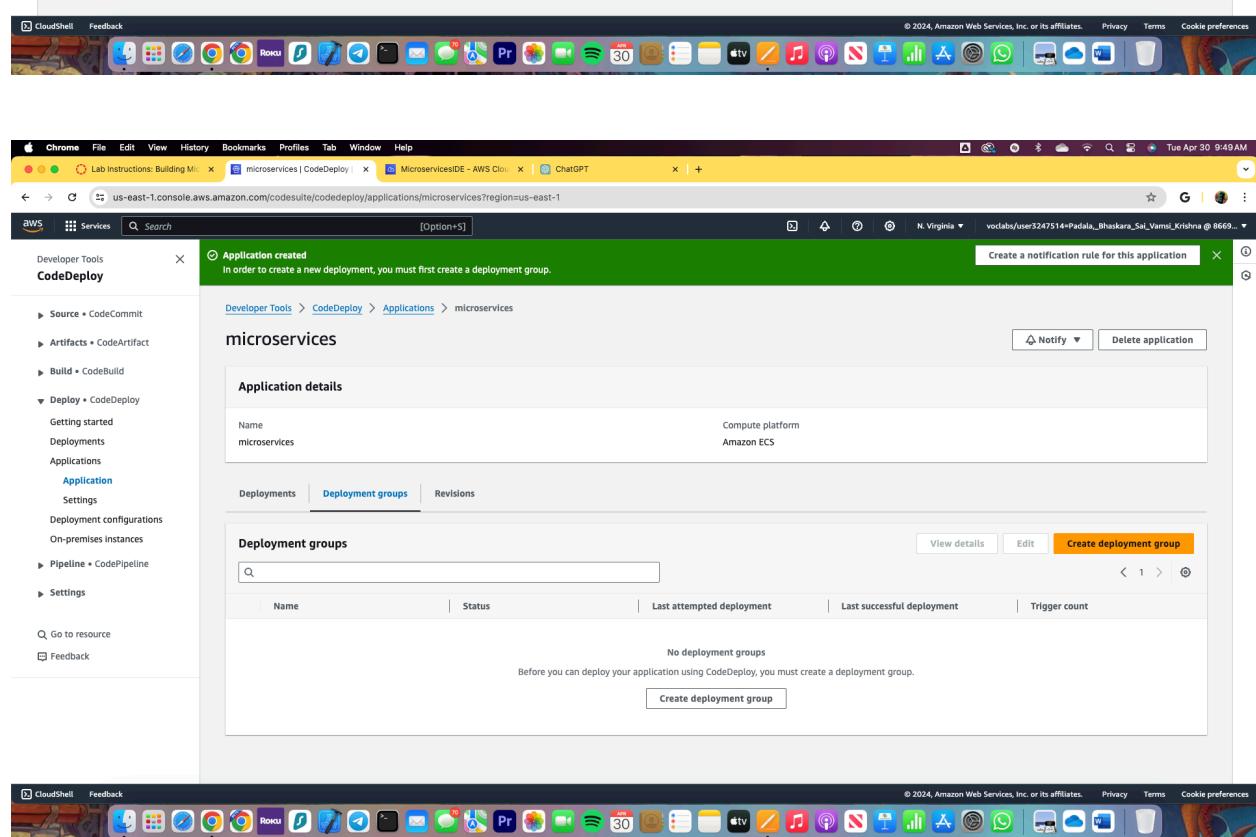
Now that you've laid out the Application Load Balancer, target groups, and the Amazon ECS services that form the infrastructure for deploying your microservices, the subsequent phase involves setting up the CI/CD pipeline to facilitate application deployment.

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



The screenshot shows the 'Create application' wizard in the AWS CodeDeploy console. The 'Application configuration' step is displayed. The 'Application name' field contains 'microservices'. The 'Compute platform' dropdown is set to 'Amazon ECS'. A 'Tags' section with an 'Add tag' button is present. At the bottom right are 'Cancel' and 'Create application' buttons, with 'Create application' being highlighted.



The screenshot shows the 'microservices' application details page. A green banner at the top states 'Application created' and 'In order to create a new deployment, you must first create a deployment group.' The 'Deployment groups' tab is active. Below it is a table for deployment groups with no entries. A 'Create deployment group' button is located at the bottom of the table area. The left sidebar shows navigation options like Source, Artifacts, Build, Deploy, Getting started, Deployments, Applications, Application settings, Deployment configurations, On-premises instances, Pipeline, and Settings.

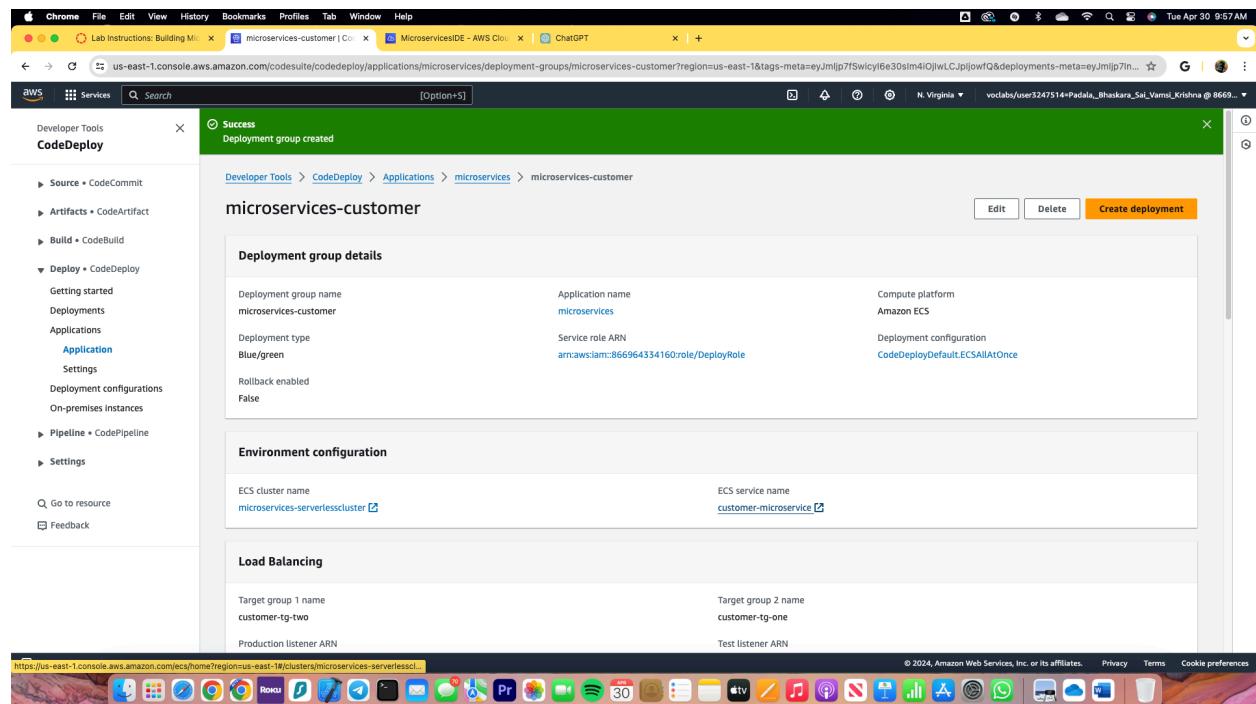
To create a CodeDeploy deployment group for the customer microservice, follow these steps:

Go to the microservices application detail page in the AWS Management Console.

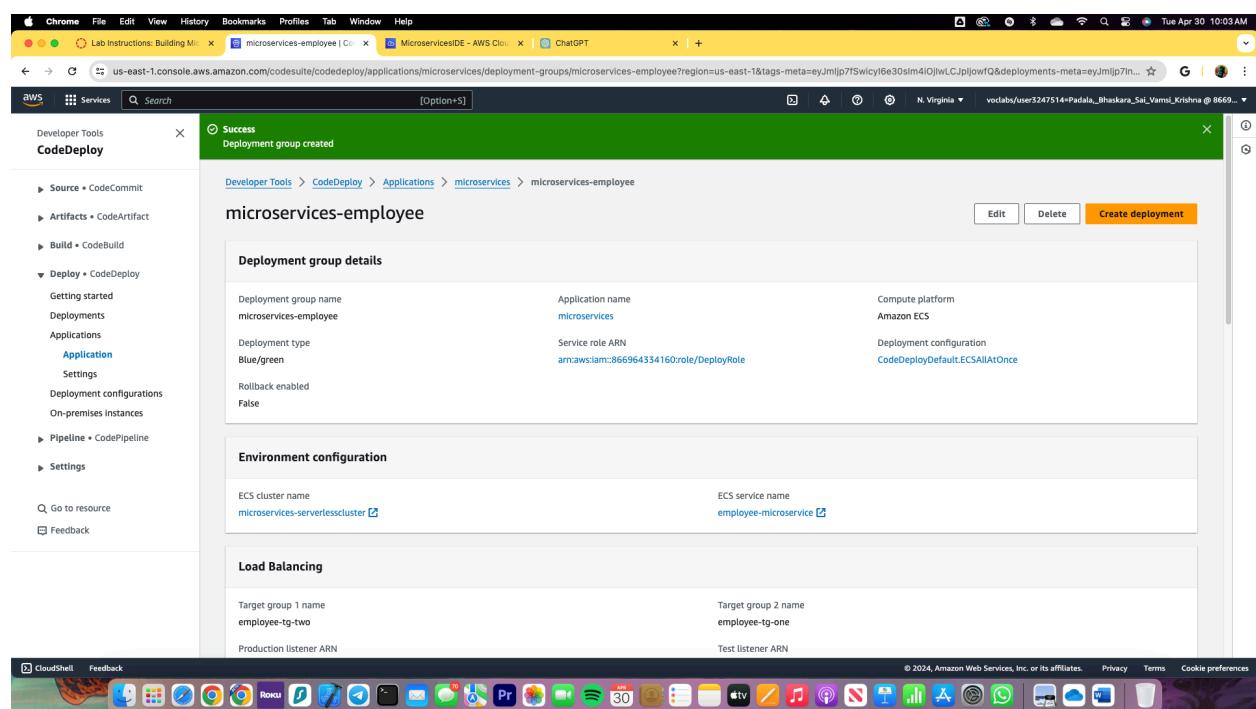
Choose the "Deployment groups" tab.

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

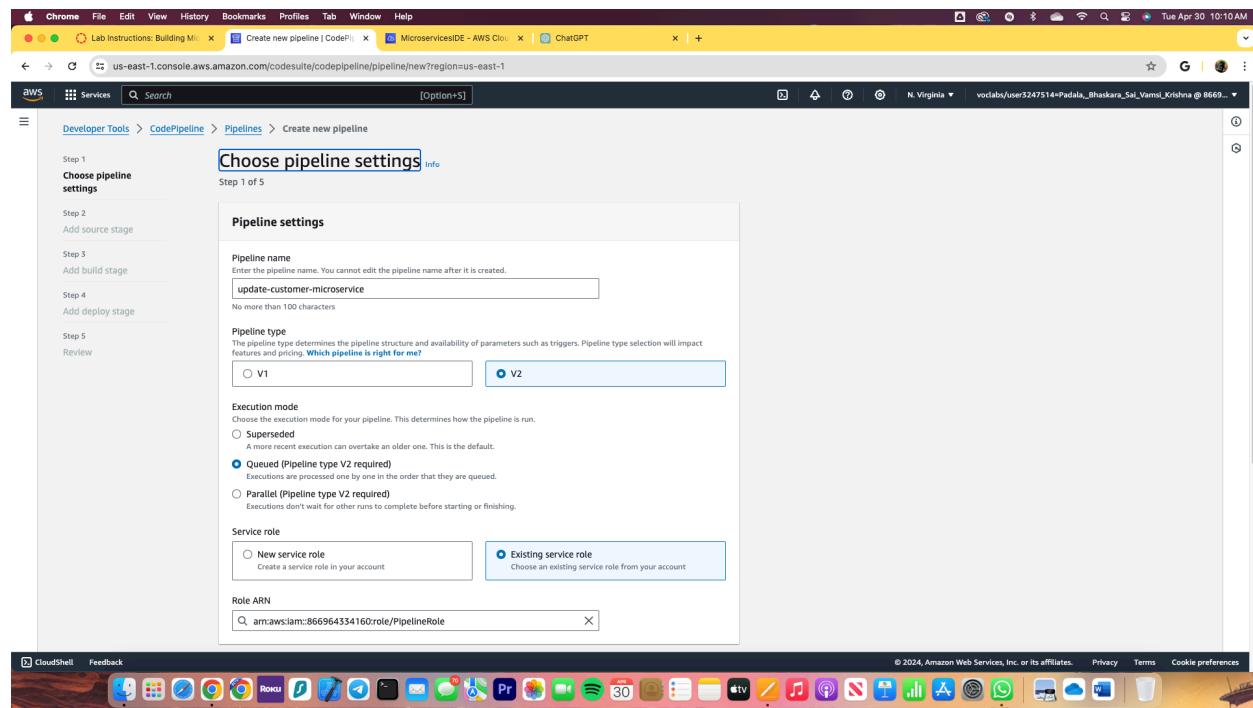


customer microservice. Configure the load balancer settings as needed, specifying the Application Load Balancer and target group associated with the customer microservice.

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.

The screenshot shows the 'Change detection options' section of the pipeline configuration. It offers two choices: 'Amazon CloudWatch Events (recommended)' and 'AWS CodePipeline'. The 'Amazon CloudWatch Events' option is selected, with a note explaining it uses CloudWatch Events to automatically start the pipeline when changes occur.

Output artifact format

The screenshot shows the 'Output artifact format' section with two options: 'Codepipeline default' (selected) and 'Full clone'. The 'Codepipeline default' option is described as using the default zip format for artifacts in the pipeline and not including Git metadata about the repository.

The screenshot shows the 'Deploy provider' section of the pipeline configuration. It lists 'Amazon ECS (Blue/Green)' as the selected provider. Other options include 'Lambda' and 'AWS Lambda'. Below the provider selection, fields for 'Region' (set to 'US East (N. Virginia)'), 'AWS CodeDeploy application name' ('microservices'), and 'AWS CodeDeploy deployment group' ('microservices-customer') are visible.

Amazon ECS task definition

The screenshot shows the 'Amazon ECS task definition' section. It specifies the source artifact as 'SourceArtifact' and the task definition file as 'taskdef-customer.json'.

AWS CodeDeploy AppSpec file

The screenshot shows the 'AWS CodeDeploy AppSpec file' section. It specifies the source artifact as 'SourceArtifact' and the appspec file as 'appspec-customer.yaml'.

Dynamically update task definition image - optional

The screenshot shows the 'Dynamically update task definition image - optional' section, which allows specifying an input artifact and a placeholder name for the container definition image.

Input artifact with image details

The screenshot shows the 'Input artifact with image details' section, where an input artifact named 'Select Input artifact' is chosen, and the placeholder text is set to 'IMAGE'.

Configure automatic rollback on stage failure

The screenshot shows the 'Configure automatic rollback on stage failure' checkbox, which is currently unchecked.

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



us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1

aws Services Search [Option+S]

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

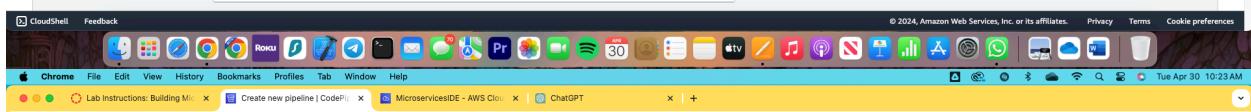
Artifact location: A new Amazon S3 bucket will be created as the default artifact store for your pipeline

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

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1

aws Services Search [Option+S]

Step 2: Add source stage

Source action provider

Source action provider: AWS CodeCommit

RepositoryName: deployment

Default branch: dev

PollForSourceChanges: false

OutputArtifactFormat: CODE_ZIP

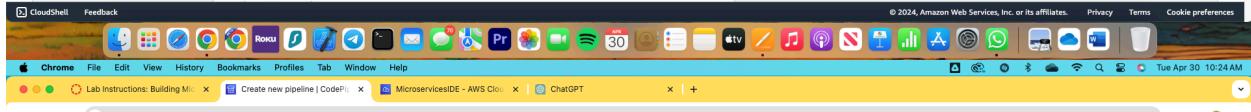
Step 3: Add build stage

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

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1

aws Services Search [Option+S]

Step 4: Add deploy stage

Deploy action provider

Deploy action provider: Amazon ECS (Blue/Green)

ApplicationName: microservices

DeploymentGroupName: microservices-customer

TaskDefinitionTemplateArtifact: taskdef-customer.json

SourceArtifact: AppSpecTemplateArtifact

SourceArtifact: AppSpecTemplatePath: appspec-customer.yaml

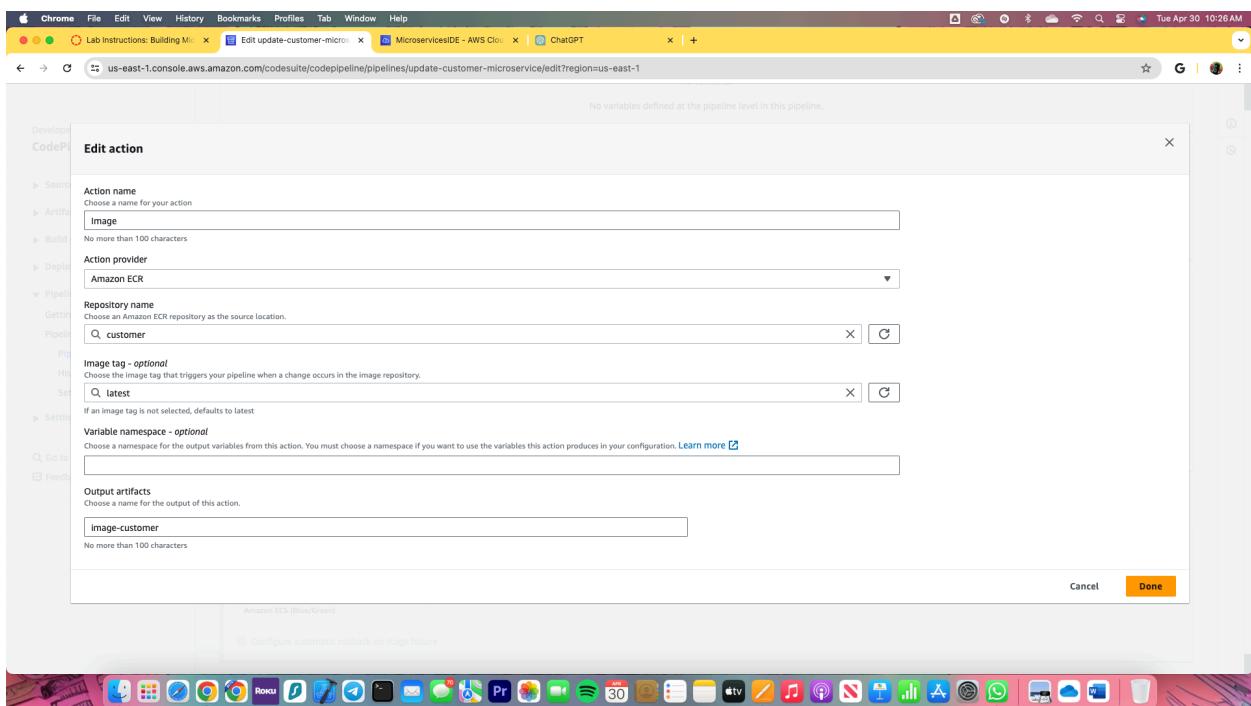
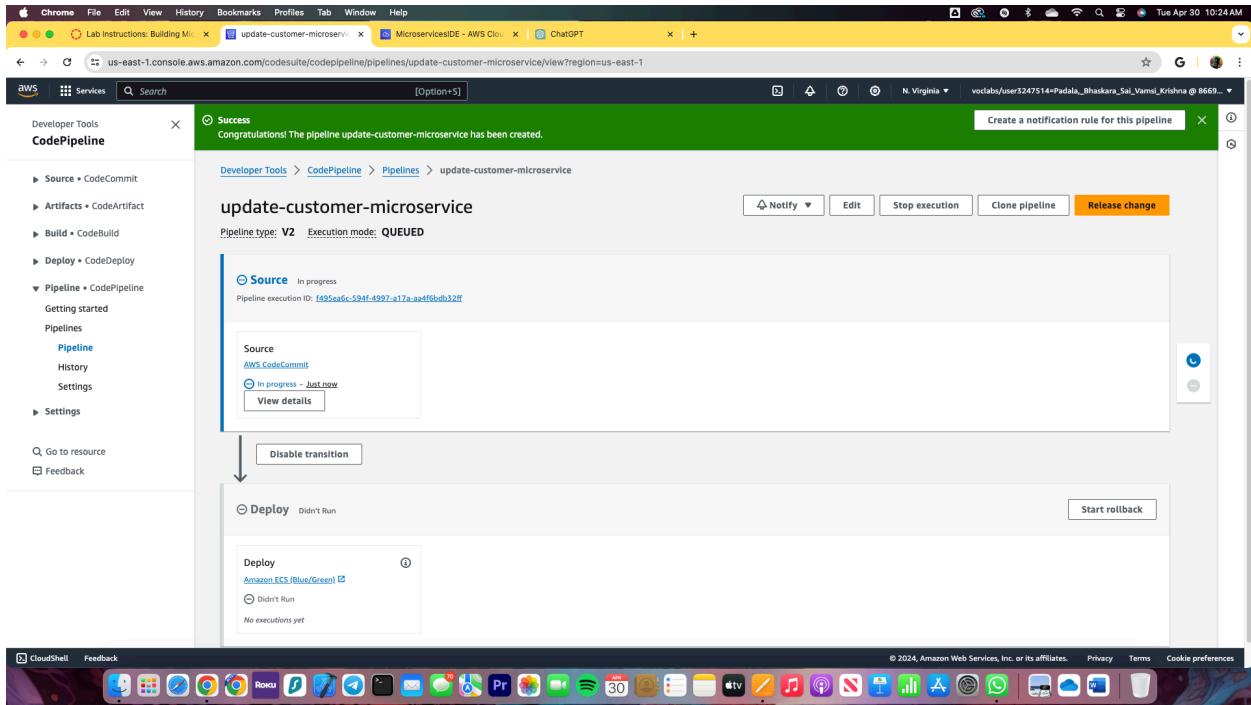
Configure automatic rollback on stage failure: Disabled

Create pipeline

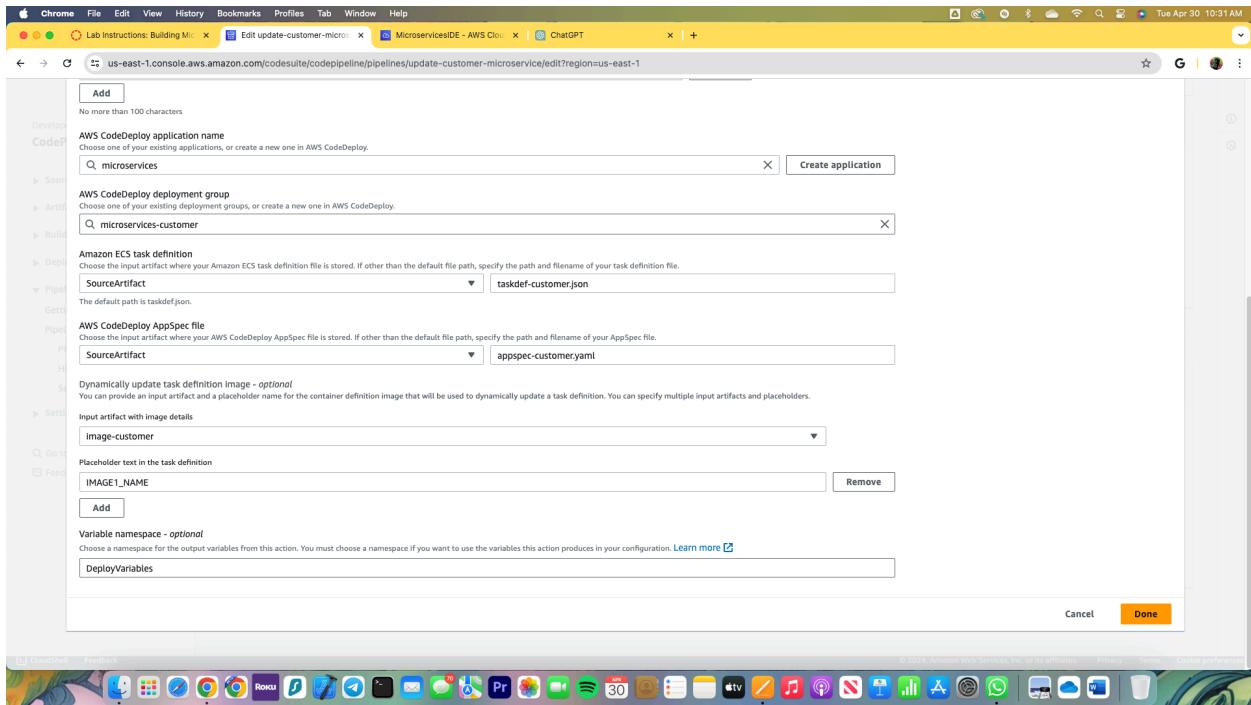
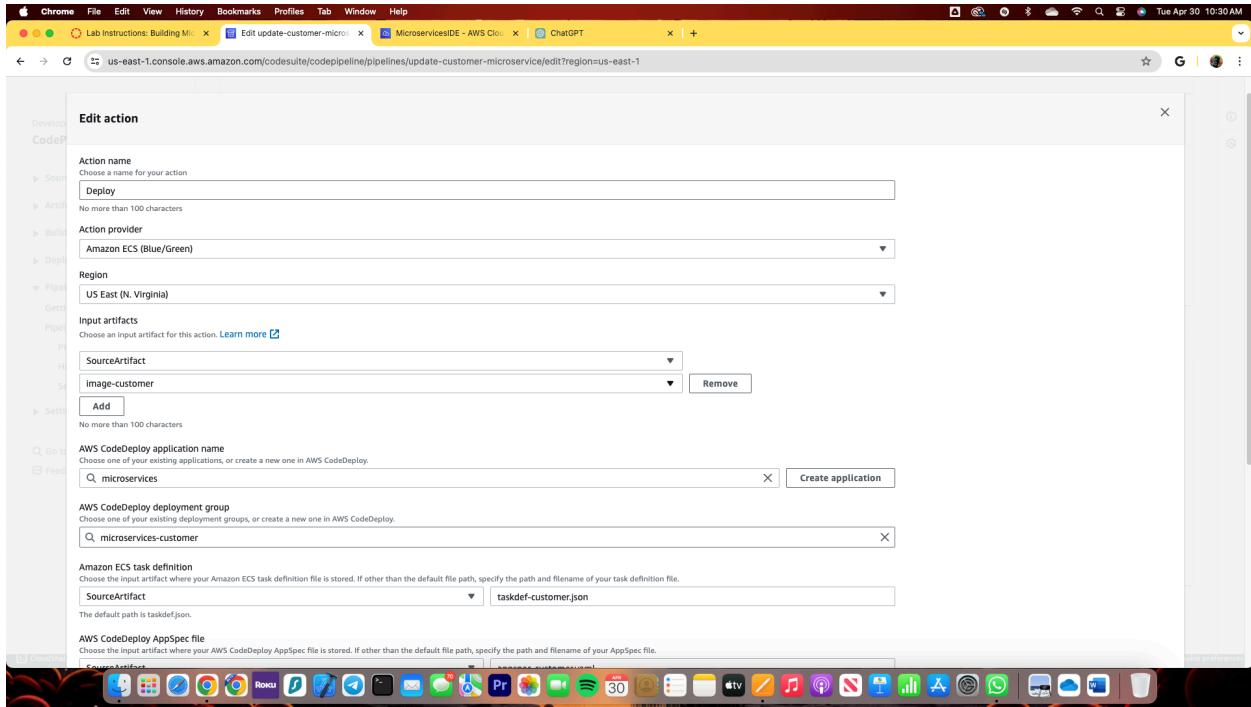
20025924

Bhaskara Sai Vamsi Krishna Padala

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.

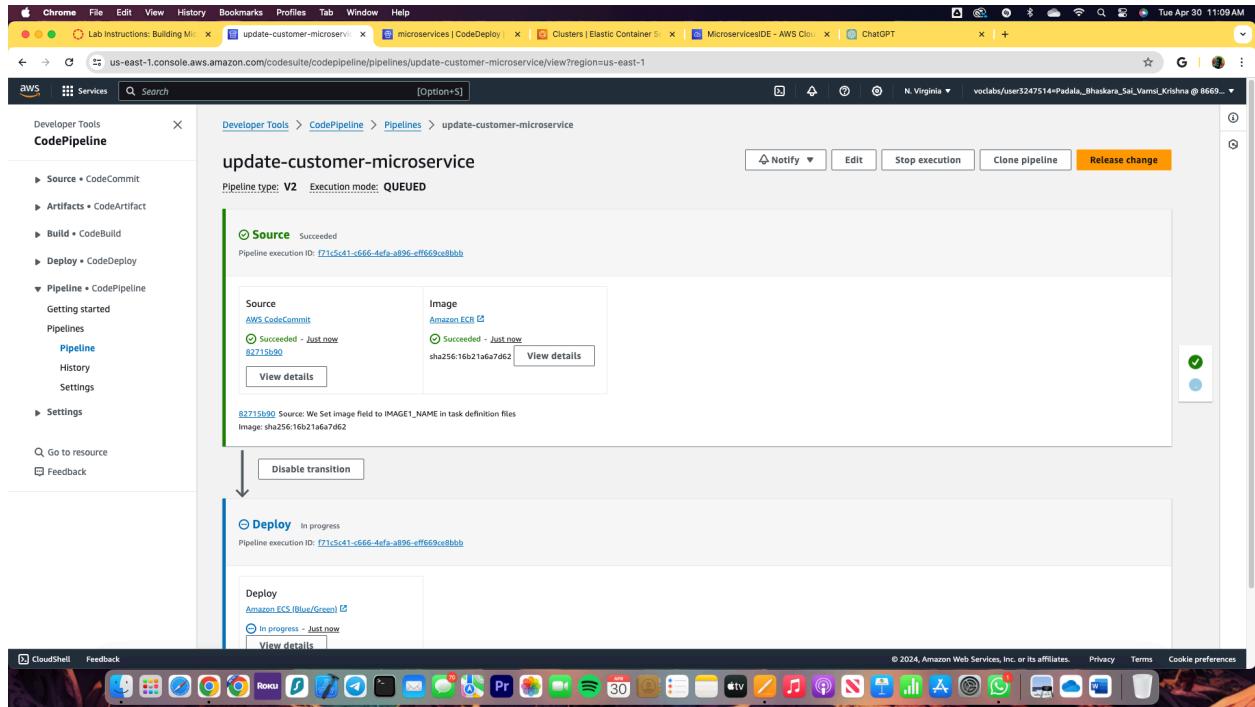
Navigate to the CodePipeline Console.

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.

20025924

Bhaskara Sai Vamsi Krishna Padala



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.

Name	Status	Last attempted deployment	Last successful deployment	Trigger count
microservices-customer	-	-	-	0
microservices-employee	-	-	-	0

Revision details

Revision location: 5b24932900b4ad509a8767efac5fb1dd2945651c83feae2e7b9e9642bf55b790

Revision created: 5 minutes ago

Revision description: Application revision registered by Deployment ID: d-OIZC6Q3V5

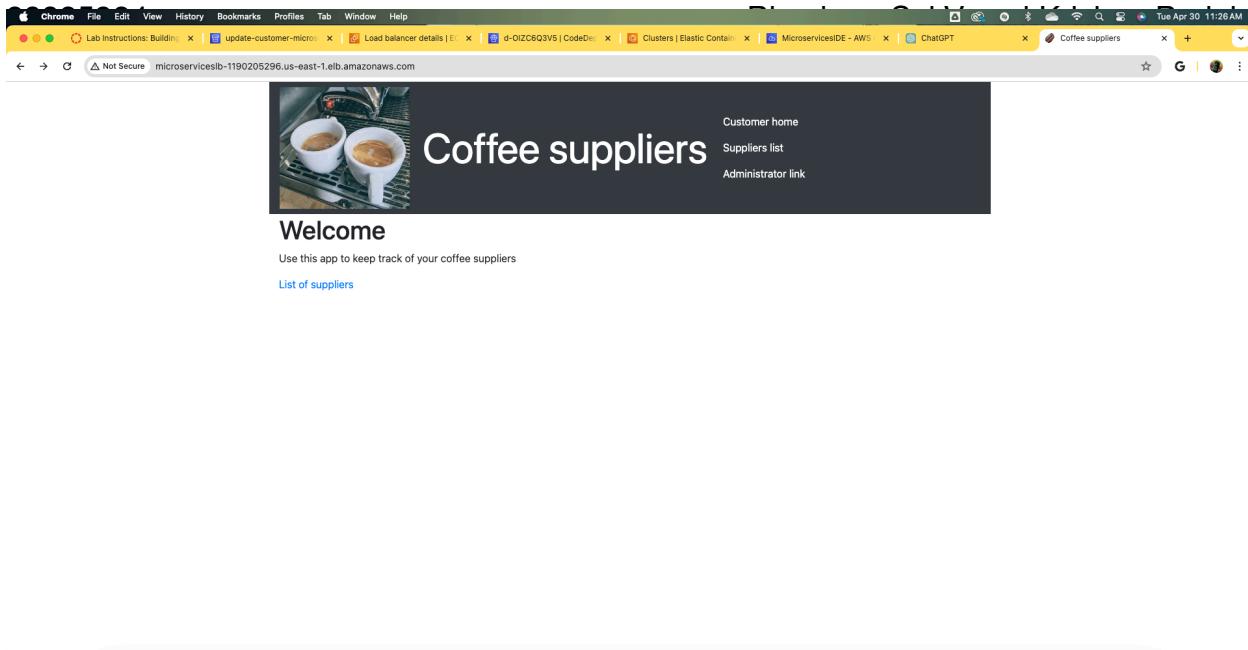
Task set activity

Task set ID	Environment	Task set status	Traffic	Desired count	Running count	Pending count
ecs-svc/6921003347052609220	Original	ACTIVE	0	1	0	0
ecs-svc/0036528239739575530	Replacement	PRIMARY	100%	1	1	0

Deployment lifecycle events

Event	Duration	Status	Start time	End time
BeforeInstall	less than one second	Succeeded	Apr 30, 2024 11:08 AM (UTC-4:00)	Apr 30, 2024 11:08 AM (UTC-4:00)
Install	2 minutes 3 seconds	Succeeded	Apr 30, 2024 11:08 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
AfterInstall	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
AllowTestTraffic	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
AfterAllowTestTraffic	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
BeforeAllowTraffic	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
AllowTraffic	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:10 AM (UTC-4:00)
AfterAllowTraffic	less than one second	Succeeded	Apr 30, 2024 11:10 AM (UTC-4:00)	Apr 30, 2024 11:11 AM (UTC-4:00)

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.

A screenshot of a web browser window showing the "All suppliers" page of the "Coffee suppliers" application. The header is identical to the home page. The main content area displays a table with one row of data:

Name	Address	City	State	Email	Phone
Bhaskara Sai Vamsi Krishna Padala	3293 JFK Blvd, Apt#2	Jersey City	New Jersey	bpadala@stevens.edu	5512221824

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.

Lab Instructions: Building a Serverless Microservices Application using AWS Lambda, API Gateway, and Amazon ElastiContainer Service (ECS)

us-east-1.console.aws.amazon.com/ecs/v2/clusters/microservices-serverlesscluster/services?region=us-east-1

Amazon Elastic Container Service

Clusters

- Namespaces
- Task definitions
- Account settings

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

microservices-serverlesscluster

Cluster overview

ARN arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster	Status Active	CloudWatch monitoring <input type="radio"/> Default	Registered container instances -
Services	Tasks		
Draining -	Active 2	Pending -	Running 1

Services (2) Info

Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition
customer-microservice	arn:aws:ec...	Active	REPLICA	<div style="width: 100%;">1/1 tasks running</div>	-	employee-micro...
employee-microservice	arn:aws:ec...	Active	REPLICA	<div style="width: 0%;">0/1 tasks running</div>	-	employee-micro...

CloudShell Feedback

us-east-1.console.aws.amazon.com/ecs/v2/clusters/microservices-serverlesscluster/tasks?region=us-east-1

Amazon Elastic Container Service

Clusters

- Namespaces
- Task definitions
- Account settings

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

microservices-serverlesscluster

Cluster overview

ARN arn:aws:ecs:us-east-1:866964334160:cluster/microservices-serverlesscluster	Status Active	CloudWatch monitoring <input type="radio"/> Default	Registered container instances -
Services	Tasks		
Draining -	Active 2	Pending -	Running 1

Tasks (1)

Task	Last status	Desired state	Task defn.	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

CloudShell Feedback

us-east-1.console.aws.amazon.com/codesuite/codedeploy/deployments/d-OIZC6Q3V5?region=us-east-1

Developer Tools

CodeDeploy

- Source + CodeCommit
- Artifacts + CodeArtifact
- Build + CodeBuild
- Deploy + CodeDeploy
- Getting started
- Deployments
- Deployment
- Applications
- Deployment configurations
- On-premises instances
- Pipeline + CodePipeline
- Settings

Go to resource

Feedback

d-OIZC6Q3V5

Deployment status

Step 1: Deploying replacement task set	Completed Succeeded	100%
Step 2: Test traffic route set-up	Completed Succeeded	100%
Step 3: Rerouting production traffic to replacement task set	Completed Succeeded	100%
Step 4: Wait	Wait completed Succeeded	100%
Step 5: Terminate original task set	Completed Succeeded	100%

Traffic shifting progress

Original <div style="background-color: black; width: 0%;">0.0%</div>	Replacement <div style="background-color: blue; width: 100%;">100.0%</div>
Original task set not serving traffic	Replacement task set serving traffic

Deployment details

Application microservices	Deployment ID d-OIZC6Q3V5	Status Succeeded
------------------------------	------------------------------	--

The screenshot shows the AWS CloudWatch Metrics Insights page for the 'customer-tg-two' target group. The interface includes a navigation bar at the top, followed by a sidebar with various AWS services like EC2, S3, Lambda, etc. The main content area displays a table of metrics with columns for Metric Name, Unit, and Value. A chart below the table shows the trend of the 'Latency' metric over time, with a red shaded area indicating an alert or threshold. The bottom of the page features a summary section and links for further actions.

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.

The screenshot shows the AWS CloudWatch Metrics Insights page for the 'customer-tg-one' target group. The interface is similar to the previous one, with a sidebar and a main content area displaying a table of metrics and a trend chart for the 'Latency' metric. The bottom of the page includes a summary section and action links.

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.

HTTP:80 Info

Details

Protocol:Port: HTTP:80

Load balancer: microservicesLB

Default actions:

- Forward to target group: customer-tg-one (100%)
- Group-level stickiness: Off

Listener ARN: arn:aws:elasticloadbalancing:us-east-1:866964334160:listener/app/microservicesLB/9a40ff3974bd3a3d/e75c01d7f6805ddf

Rules

Listener rules (2) Info

Name tag	Priority	Conditions (If)	Actions (Then)	ARN	Tags
-	1	Path Pattern is /admin/*	Forward to target group employee-tg-one (100%) Group-level stickiness: Off	ARN	0 tags
Default	Last (default)	If no other rule applies	Forward to target group customer-tg-one (100%) Group-level stickiness: Off	ARN	0 tags

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

HTTP:8080 Info

Details

Protocol:Port: HTTP:8080

Load balancer: microservicesLB

Default actions:

- Forward to target group: customer-tg-one (100%)
- Group-level stickiness: Off

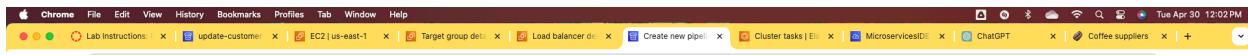
Listener ARN: arn:aws:elasticloadbalancing:us-east-1:866964334160:listener/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2

Rules

Listener rules (2) Info

Name tag	Priority	Conditions (If)	Actions (Then)	ARN	Tags
-	1	Path Pattern is /admin/*	Forward to target group employee-tg-one (100%) Group-level stickiness: Off	ARN	0 tags
Default	Last (default)	If no other rule applies	Forward to target group customer-tg-one (100%) Group-level stickiness: Off	ARN	0 tags

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



aws Services Search [Option+S]

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Step 1 Choose pipeline settings Step 5 of 5

Review info

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.



CloudShell Feedback

Chrome File Edit View History Bookmarks Profiles Tab Window Help

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 12:02PM

Lab Instructions | update-custome... | EC2 us-east-1 | Target group det... | Load balancer de... | Create new pipe... | Cluster tasks | El... | MicroservicesIDF | ChatGPT | Coffee suppliers

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1

aws Services Search [Option+S]

Step 2: Add source stage

Source action provider

Source action provider: AWS CodeCommit

RepositoryName: deployment

Default branch: dev

PollForSourceChanges: false

OutputArtifactFormat: CODE_ZIP

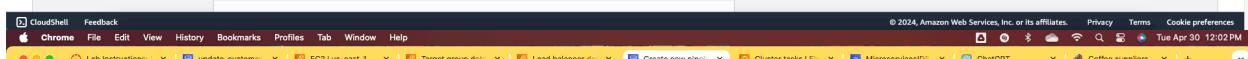
Step 3: Add build stage

Build action provider

Build stage: No build

Step 4: Add deploy stage

Deploy action provider



CloudShell Feedback

Chrome File Edit View History Bookmarks Profiles Tab Window Help

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Tue Apr 30 12:02PM

Lab Instructions | update-custome... | EC2 us-east-1 | Target group det... | Load balancer de... | Create new pipe... | Cluster tasks | El... | MicroservicesIDF | ChatGPT | Coffee suppliers

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1

aws Services Search [Option+S]

Step 4: Add deploy stage

Deploy action provider

Deploy action provider: Amazon ECS (Blue/Green)

ApplicationName: microservices

DeploymentGroupName: microservices-employee

TaskDefinitionTemplateArtifact: taskdef-employee.json

SourceArtifact: AppSpecTemplateArtifact

TaskDefinitionTemplatePath: taskdef-employee.json

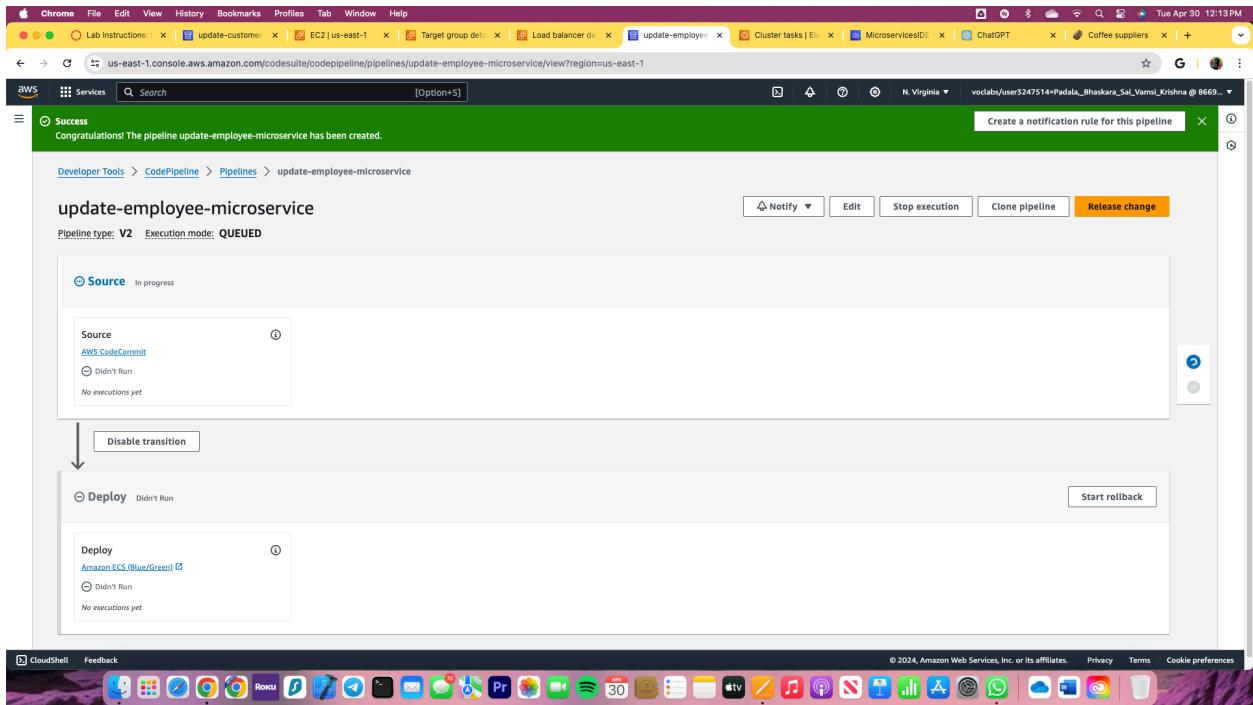
Configure automatic rollback on stage failure: Disabled

Cancel Previous Create pipeline

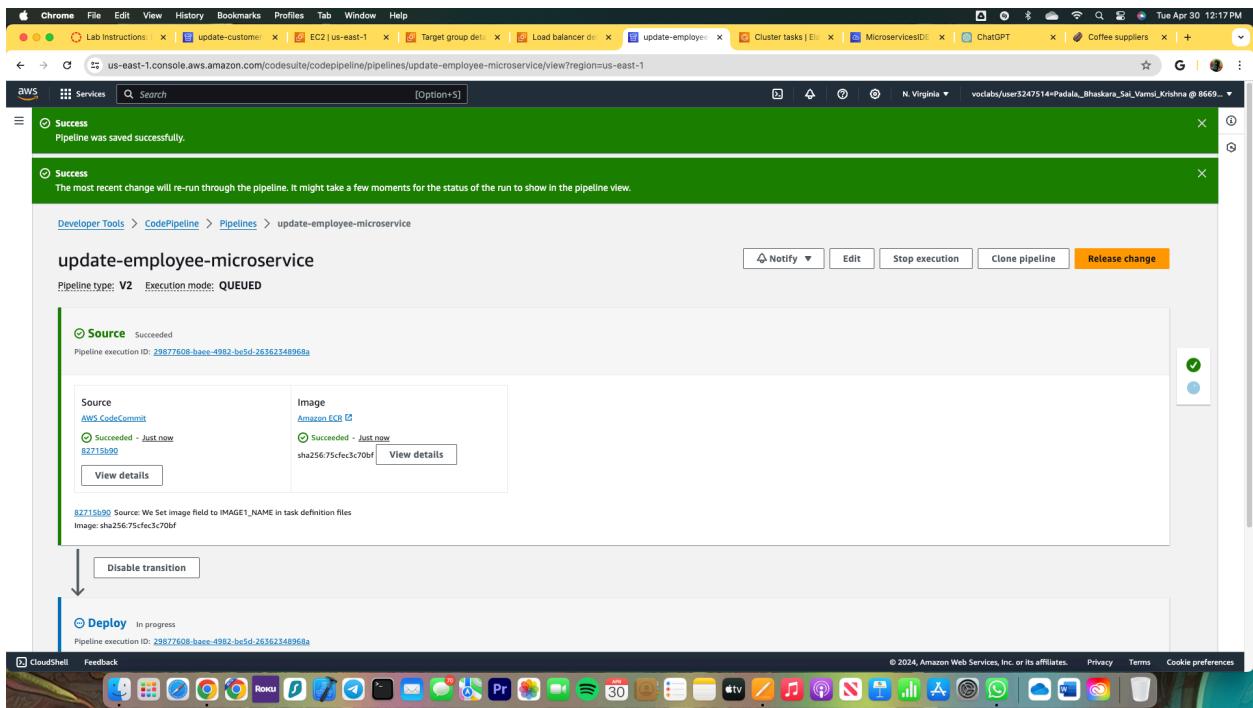


20025924

Bhaskara Sai Vamsi Krishna Padala



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.

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:8669.../targetgroup/customer-tg-one/54134...	8080	HTTP	IP	microservicesLB	vpc-089ece87b4fc2f9bb
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:8669.../targetgroup/customer-tg-two/54134...	8080	HTTP	IP	None associated	vpc-089ece87b4fc2f9bb
employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:8669.../targetgroup/employee-tg-one/54134...	8080	HTTP	IP	microservicesLB	vpc-089ece87b4fc2f9bb

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

The screenshot shows the AWS CloudWatch Metrics interface. On the left, a sidebar lists various AWS services like EC2 Dashboard, Instances, and Network & Security. The main area displays four time-series charts under the heading "4 target groups selected". Each chart has four panels: "Unhealthy Hosts (Maximum)", "Healthy Hosts (Minimum)", "Unhealthy Hosts (Average)", and "Healthy Hosts (Average)". The charts show data for "customer-tg-one" and "customer-tg-two" over the period from April 25 to April 29. The "Healthy Hosts (Average)" chart for "customer-tg-one" shows values of 1.16 and 0.578, while for "customer-tg-two" it shows 0.5.

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.

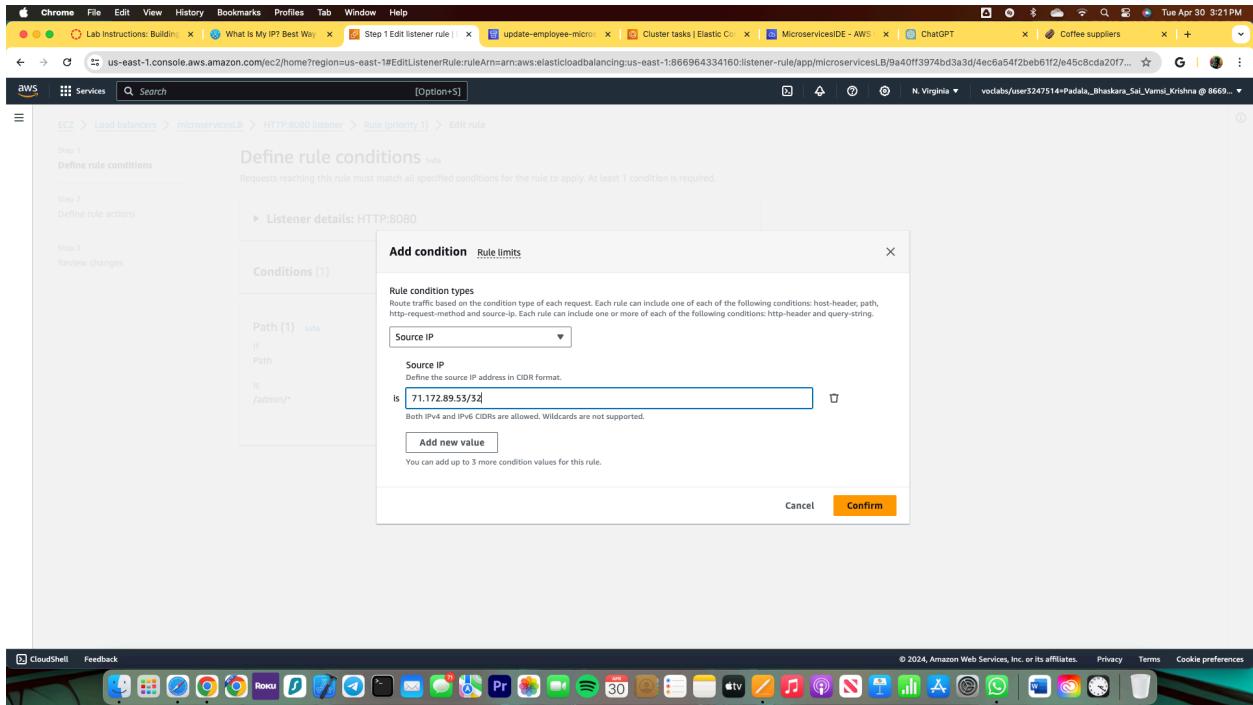
The screenshot shows a web browser displaying the "WhatIsMyIP.com" website. The main content area shows the following information:

- What Is My IP? (Large green header)
- My Public IPv4: [71.172.89.53](#)
- My Public IPv6: Not Detected
- My IP Location: Newark, NJ US
- My ISP: Verizon Business

Below this, there are two expandable sections:

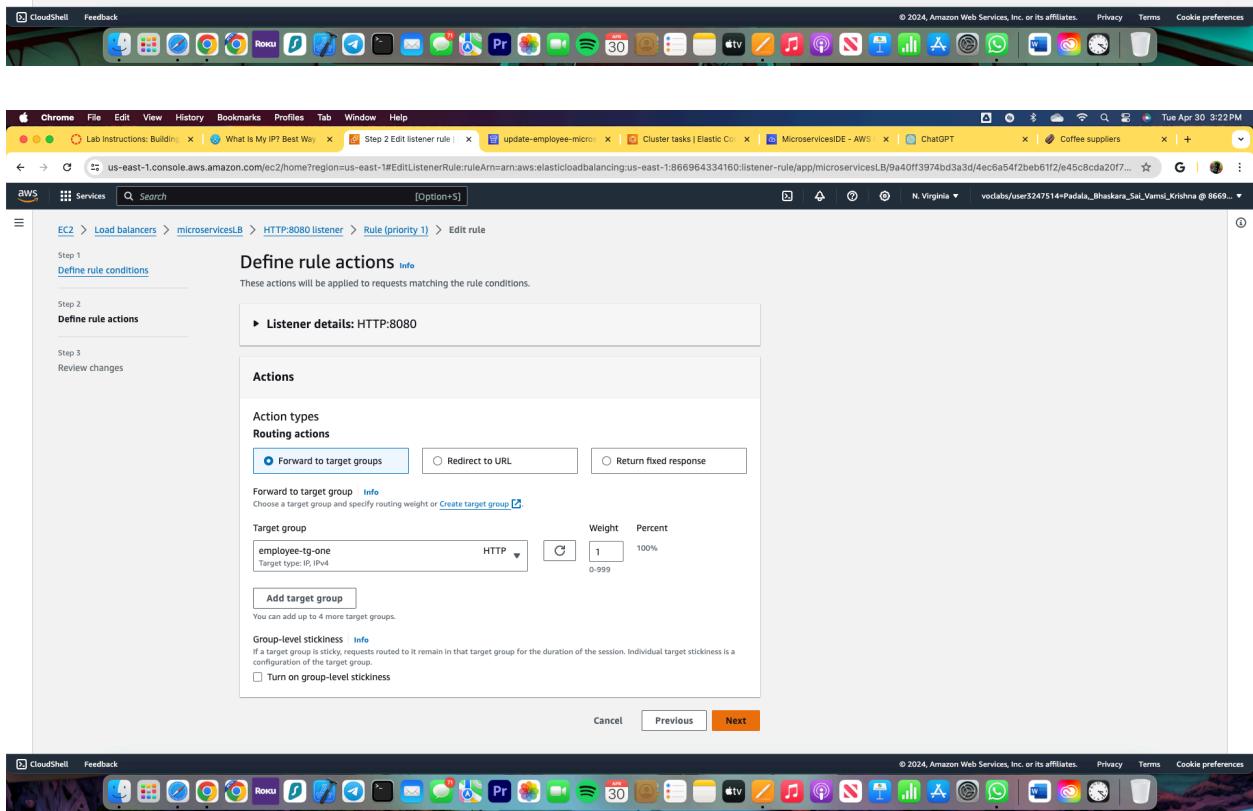
- What is an IP address?
- What is a private IP address?

Screenshot of the AWS CloudFront console showing the "Define rule conditions" step. A modal dialog titled "Add condition" is open, showing a "Source IP" condition with the value "71.172.89.53/32".



The screenshot shows the AWS CloudFront console with the URL `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#EditListenerRule:ruleArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:listener-rule/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2/e45c8cda2017...`. The page title is "Step 1 Edit listener rule | Step 1 Edit listener rule | What Is My IP? Best Web...".

Screenshot of the AWS CloudFront console showing the "Define rule actions" step. A modal dialog titled "Actions" is open, showing a "Forward to target groups" action selected. It lists a target group named "employee-tg-one" with a weight of 1 and a percent of 100%.



The screenshot shows the AWS CloudFront console with the URL `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#EditListenerRule:ruleArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:listener-rule/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2/e45c8cda2017...`. The page title is "Step 1 Edit listener rule | Step 1 Edit listener rule | What Is My IP? Best Web...".

Lab Instructions: Building a Microservices Application | What Is My IP? Best Way | Step 3 Edit listener rule | update-employee-microservice | Cluster tasks | Elastic Cache | MicroservicesIDE - AWS | ChatGPT | Coffee suppliers

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#EditListenerRule:ruleArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:listener-rule/app/microservicesLB/9a40ff3974bd3a3d/4ec6a54f2beb61f2/e45c8cd20f7832a

N. Virginia vodabs/user3247514#Padala_Bhaskara_Sai_Vamsi_Krishna @ 8669...

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

Review changes

Listener details: HTTP:8080

Rule details

Priority	Conditions (If)	Actions (Then)
1	If request matches all: • Path Pattern Is /admin/*, AND • Source IP Is 71.172.89.53/32	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/e45c8cd20f7832a

Cancel Previous Save changes



CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lab Instructions: Building a Microservices Application | What Is My IP? Best Way | Listener details | EC2 | update-employee-microservice | Cluster tasks | Elastic Cache | MicroservicesIDE - AWS | ChatGPT | Coffee suppliers

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ELBLListenerV2:loadBalancerArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:loadbalancer/app/microservicesLB/9a40ff3974bd3a3d/listenerPort=8080?action=...

N. Virginia vodabs/user3247514#Padala_Bhaskara_Sai_Vamsi_Krishna @ 8669...

EC2 Dashboard > Services Search [Option+S]

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 AM Catalog

Elastic Block Store Volumes Snapshots Lifecycle Manager

Network & Security Security Groups Elastic IPs Placement Groups Key Pairs

Successfully updated rule on listener HTTP:8080.

HTTP:8080 Info

Details

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

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

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

Name tag	Priority	Conditions (If)	Actions (Then)	ARN	Tags
-	1	• 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	ARN	0 tags
Default	Last	If no other rule applies	Forward to target group • customer-tg-one 1 (100%)	ARN	0 tags



CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lab Instructions: Building a Microservices Application | What Is My IP? Best Way | Step 1 Edit listener rule | update-employee-microservice | Cluster tasks | Elastic Cache | MicroservicesIDE - AWS | ChatGPT | Coffee suppliers

us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#EditListenerRule:ruleArn=arn:aws:elasticloadbalancing:us-east-1:866964334160:listener-rule/app/microservicesLB/9a40ff3974bd3a3d/e75c01d776805df/57d69de6d48...

N. Virginia vodabs/user3247514#Padala_Bhaskara_Sai_Vamsi_Krishna @ 8669...

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 conditions

Requests reaching this rule must match all specified conditions for this rule to apply. At least 1 condition is required.

Listener details: HTTP:8080

Add condition Rule limits

Conditions (1)

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.

Path (1)

Path
Is /admin/*

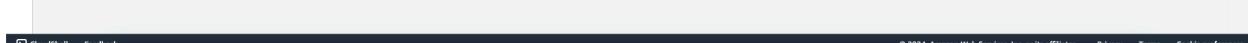
Source IP

Define the source IP address in CIDR format.
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



20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS CloudShell interface with multiple tabs open. The main window displays the configuration of an Application Load Balancer rule named 'HTTP:80 listener'. The rule has a priority of 1 and a condition: 'If request matches all: Path Pattern is /admin/*, AND Source IP is 71.172.89.53/32'. The action is 'Forward to target group' with a weight of 100 (100%). Group-level stickiness is set to 'Off'. The ARN of the rule is listed as well. At the bottom, there are 'Cancel', 'Previous', and 'Save changes' buttons.

The sidebar on the left shows the AWS navigation menu with sections like EC2 Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, Network & Security, and more. The 'Console-to-Code Preview' section is currently selected.

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.

20025924

Bhaskara Sai Vamsi Krishna Padala

A screenshot of a Mac desktop. At the top, there's a browser window with several tabs open, including "Lab Instructions: Building a", "What Is My IP? Best Way", "Listener details | EC2", "update-employee-microservice", "Cluster tasks | Elastic Container Service", "MicroservicesDE - AWS", and "ChatGPT". Below the browser is the AWS Cloud9 IDE interface. On the left, a file tree shows a directory structure for "MicroservicesDE - home/ec2-user/environment" with files like "nav.html", "index.js", "package-lock.json", "package.json", "labuser.pem", and "README.md". In the main editor area, the "nav.html" file is open, displaying HTML code for a navigation bar. To the right of the editor is a terminal window titled "bash - *ip-10-16-10-27.ec2.us-east-1.amazonaws.com" with the command "ls" running. The terminal output shows a single file named "employees.json". The status bar at the bottom indicates "13:7 HTML Spaces:4". The desktop dock at the bottom has icons for various applications like Finder, Mail, and Safari.

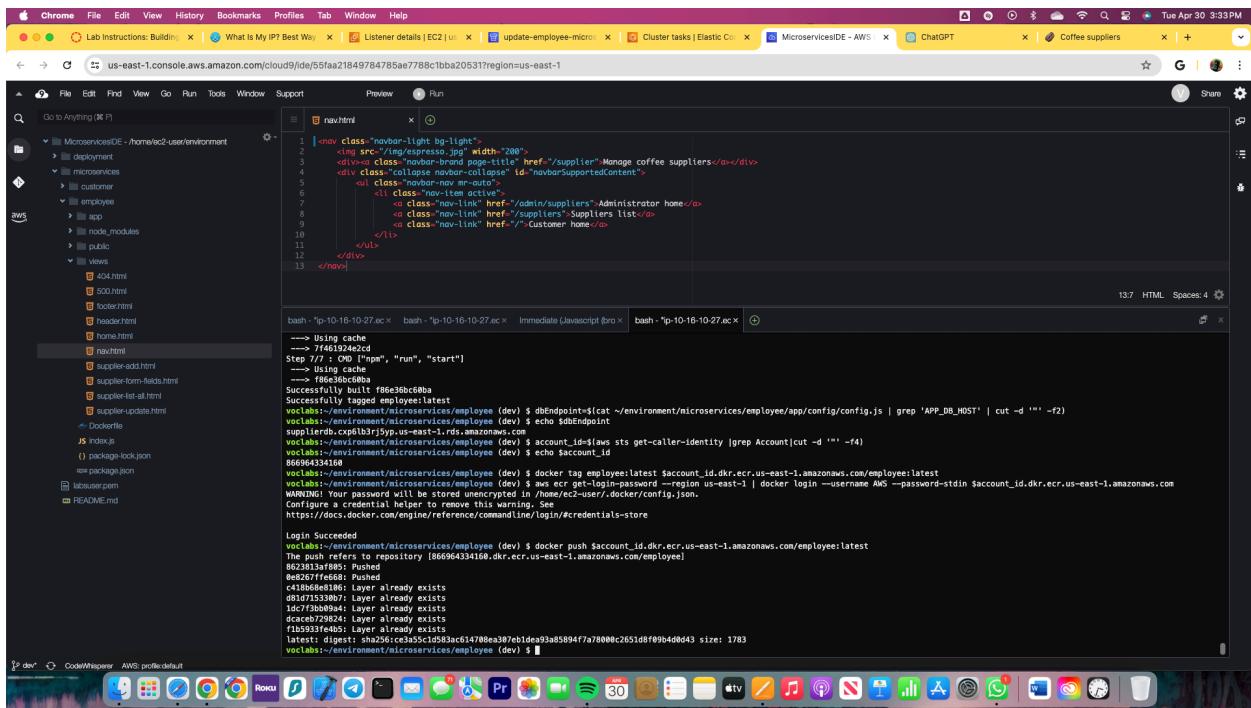
Save the updated file.

A screenshot of a Mac desktop, similar to the previous one but with different terminal output. The terminal window now shows the command "cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST'" being run, followed by the output "8669434168". The rest of the terminal output is identical to the previous screenshot, showing the creation of an ECR image and pushing it to the repository.

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:



```

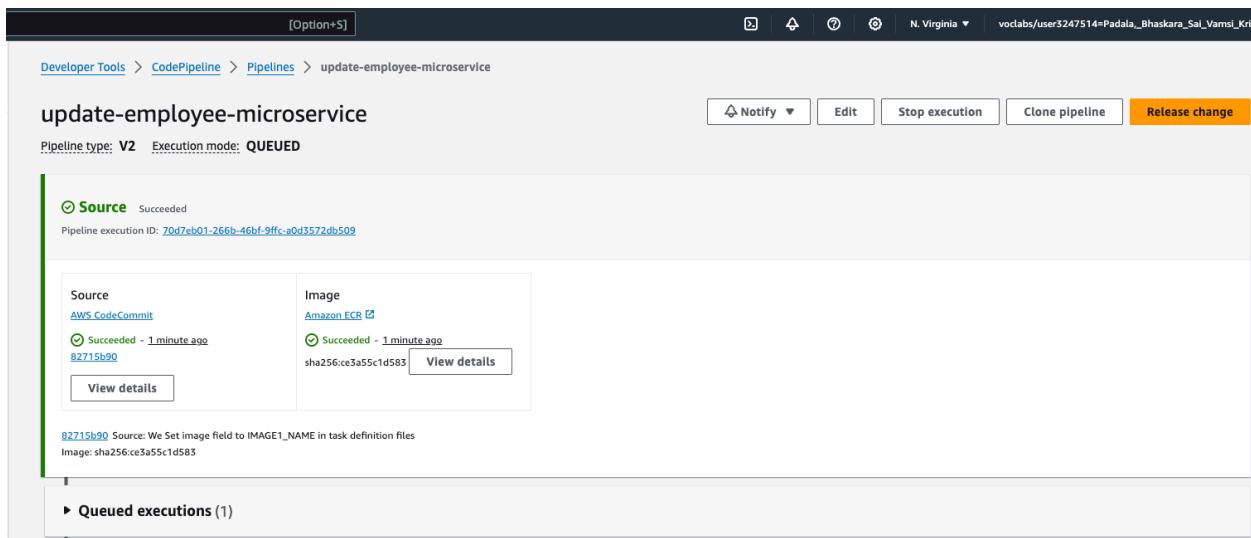
Go Anything (F)
MicroservicesIDE - home/ec2-user/environment
  deployment
  microservices
    customer
      employee
        app
        node_modules
        public
      views
        404.html
        500.html
        footer.html
        header.html
        home.html
        nav.html
        supplier-add.html
        supplier-form-fields.html
        supplier-list-all.html
        supplier-update.html
      Dockerfile
      index.js
      package-lock.json
      package.json
      labuser.pem
      README.md

bash - *ip-10-16-10-27.ac ~ bash - *ip-10-16-10-27.ac ~ Immediate Javascript (two x) bash - *ip-10-16-10-27.ac ~
---- Using cache
---- 7f44f192e4cd
Step 7/7 : CMD ["npm", "run", "start"]
--> Using cache
--> f98e3d0b00
Successfully built f98e3d0b00
Successfully tagged employee:latest
vocelabs:/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
vocelabs:/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.ebp013rjywu-east-1.rds.amazonaws.com
vocelabs:/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity | grep Account | cut -d '-' -f4)
vocelabs:/environment/microservices/employee (dev) $ echo $account_id
8623813a9f885
vocelabs:/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vocelabs:/environment/microservices/employee (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING: Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configuring a credential helper to remove this warning.
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
vocelabs:/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [8623813a9f885.dkr.ecr.us-east-1.amazonaws.com/employee]
8623813a9f885: Pushed
8623813a9f885: Pushed
8623813a9f885: Layer already exists
db1d71533807: Layer already exists
1d7f3ba0984: Layer already exists
dcace0751490: Layer already exists
f1a03314a051: Layer already exists
latest: digest: sha256:cce3a55c1d583 size: 1783
vocelabs:/environment/microservices/employee (dev) $

```

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



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

update-employee-microservice

Pipeline type: V2 Execution mode: QUEUED

Source	Image
AWS CodeCommit Succeeded Pipeline execution ID: 70d7eb01-266b-46bf-9ffc-a0d3572db509	Amazon ECR Succeeded - 1 minute ago sha256:cce3a55c1d583

82715b90 Source: We Set image field to IMAGE1_NAME in task definition files
Image: sha256:cce3a55c1d583

Queued executions (1)

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



Manage Coffee suppliers

All suppliers

Name	Address	City	State	Email	Phone
Bhaskara Sai Vamsi Krishna Padala	3293 JFK Blvd, Apt#2	Jersey City	New Jersey	bpadala@stevens.edu	5512221824

[Add a new supplier](#)

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 `<p class="lead">Sorry, we don't seem to have that page in stock</p>`.

Rheekara Sai Venki Krishna Padela

```

Go to Anything (⌘ F)                                preview run
MicroservicesIDE - home/ec2-user/environment
└── deployment
    └── microservices
        ├── customer
        └── employee
            └── app
                └── views
                    ├── 404.html
                    ├── 500.html
                    ├── footer.html
                    ├── header.html
                    ├── home.html
                    └── nav.html
aws
└── Dockerfile
└── index.js
├── package-lock.json
└── package.json
lsources.txt
└── README.md

bash -i p-10-16-10-27.ec.x bash -i p-10-16-10-27.ec.x Immediate (Javascript bro x) bash -i p-10-16-10-27.ec.x
vclabs:~/environment/microservices/employee (dev) $ curl http://microservicesLB-1190285296.us-east-1.elb.amazonaws.com/admin/suppliers
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <link rel="stylesheet" href="/css/base.css">
    <title>Coffee suppliers</title>
</head>
<body>
<div class="container">
    <nav class="navbar-expand-lg navbar-dark bg-dark">
        <img alt="img/espresso.jpg" width="200">
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a href="#" class="nav-link" href="#">Administrator home</a>
                <li class="nav-item">
                    <a href="#" class="nav-link" href="#">Customer home</a>
                <li class="nav-item">
                    <a href="#" class="nav-link" href="#">Suppliers list</a>
                <li class="nav-item">
                    <a href="#" class="nav-link" href="#">Customer link</a>
                </ul>
            </div>
        </nav>
        <div class="jumbotron">
            <h1>Sorry, we don't seem to have that page in stock</h1>
            <p>Return to home</p>
            <p><a href="#" class="btn btn-primary btn-lg" href="#" role="button">Return to home</a></p>
        </div>
    </div>
    <script src="/js/jquery-3.6.0.min.js"></script>
    <script src="/js/bootstrap.min.js"></script>
</body>
vclabs:~/environment/microservices/employee (dev) $ 

```

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:

```

Go to Anything (⌘ F)                                preview run
MicroservicesIDE - home/ec2-user/environment
└── deployment
    └── microservices
        ├── customer
        └── employee
            └── app
                └── views
                    ├── 404.html
                    ├── 500.html
                    ├── footer.html
                    ├── header.html
                    ├── home.html
                    └── nav.html
aws
└── Dockerfile
└── index.js
├── package-lock.json
└── package.json
lsources.txt
└── README.md

bash -i p-10-16-10-27.ec.x bash -i p-10-16-10-27.ec.x Immediate (Javascript bro x) aws -i p-10-16-10-27.ec.x
vclabs:~/environment/microservices/employee (dev) $ aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3
{
    "service": {
        "serviceName": "arn:aws:ecs:us-east-1:86696434168:service/microservices-serverlesscluster/customer-microservice",
        "serviceName": "customer-microservice",
        "clusterArn": "arn:aws:ecs:us-east-1:866964334168:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:866964334168:targetgroup/customer-tg-one/257cl83ed64b77",
                "containerName": "customer",
                "containerPort": 8088
            }
        ],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 3,
        "runningCount": 1,
        "pendingTasks": 0,
        "launchType": "FARGATE",
        "platformVersion": "1.4.8",
        "platformFamily": "LINUX",
        "taskDefinition": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:2",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "taskSets": [
            {
                "id": "ecs-svc-003652823973957538",
                "taskSetArn": "arn:aws:ecs:us-east-1:866964334168:task-set/microservices-serverlesscluster/customer-microservice/ecs-svc/003652823973957538",
                "serviceArn": "arn:aws:ecs:us-east-1:866964334168:service/microservices-serverlesscluster/customer-microservice",
                "clusterArn": "arn:aws:ecs:us-east-1:866964334168:cluster/microservices-serverlesscluster",
                "status": "PRIMARY",
                "externalId": "0-0ZCQD9V5",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:us-east-1:866964334168:task-definition/employee-microservice:2",
                "containerOverrides": []
            }
        ]
    }
}
vclabs:~/environment/microservices/employee (dev) $ 

```

20025924

Bhaskara Sai Vamsi Krishna Padala

The screenshot shows the AWS CloudWatch Metrics interface. At the top, there are tabs for 'Metrics' and 'Logs'. Below that, a search bar and a date range selector ('Last 1 hour') are visible. The main area displays three time-series metrics for a Lambda function named 'HelloWorld':

- Latency:** Represented by a blue line, it shows a single sharp peak reaching approximately 100 ms at around 10 seconds.
- Throughput:** Represented by an orange line, it shows a steady increase from 0 to about 1000 requests per second over the 1-hour period.
- ErrorRate:** Represented by a green line, it remains near zero throughout the duration.

The X-axis represents time from 0 to 1 hour, and the Y-axis represents metric values.

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.