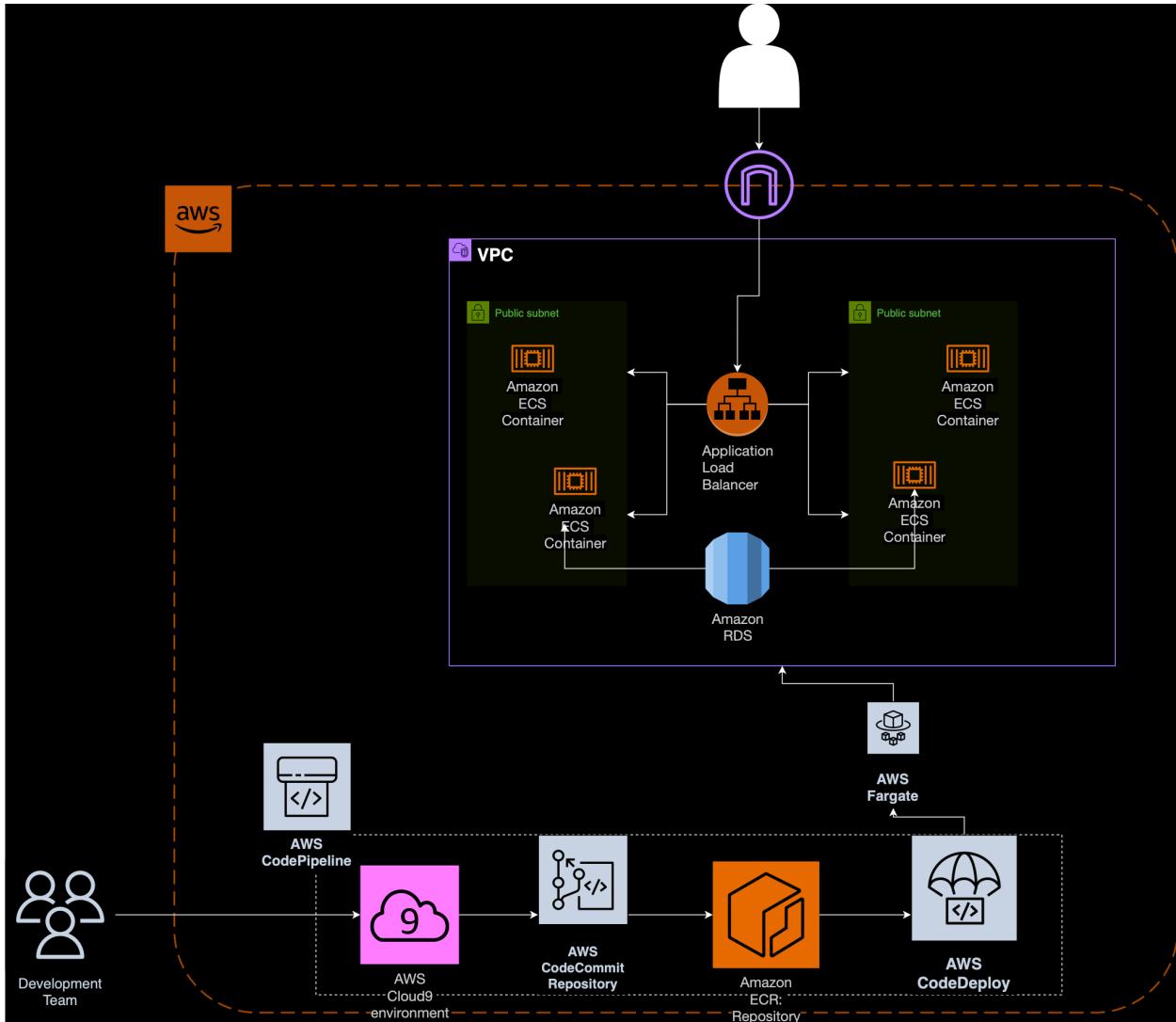


CS 524 Final Project

Part 1: Project Execution

Phase 1: Planning the design and estimating cost

Task 1.1: Create an architecture diagram



Phase 2: Analyzing the infrastructure of the monolithic application

Task 2.1: Verify that the monolithic application is available

1. Verify that the monolithic web application is accessible from the internet.

- Navigate to the Amazon EC2 console.

The screenshot shows the AWS EC2 Instances page. A single instance is listed:

- Name:** MonolithicApp...
- Instance ID:** i-02916d49615da29b3
- Instance state:** Running
- Instance type:** t2.micro
- Status check:** 2/2 checks passed
- Public IPv4 DNS:** ec2-44-222-81-196.compute-1.amazonaws.com
- Public IPv4 IP:** 44.222.81.196

- Copy the Public IPv4 address of the MonolithicAppServer instance, and load it in a new browser tab. The coffee suppliers website displays.

The screenshot shows a browser window displaying the 'Monolithic Coffee suppliers' website. The page includes:

- A header image of two cups of coffee.
- The text "Monolithic Coffee suppliers".
- A "Welcome" section.
- A note: "Use this app to keep track of your coffee suppliers".
- A link: "List of suppliers".
- A navigation bar with links: Home, Suppliers list, and other external links like Prime Video, PowerPoint Presentations, Royalty Free Stock Photos, Authentic Stock Photos, Fotolia - Sell and..., Alamy - Stock Photos, Commercial Photos, Outlook, Nice!, The creative..., Grabbit, font pairing typekit, Google Fonts, and more.

Task 2.2: Test the monolithic web application

- Choose List of suppliers. Notice that the URL path includes /Notice that the URL path includes /suppliers.

- Add a new supplier.

- On the page where you add a new supplier, notice that the URL path includes /supplier-add.
- Fill in all of the fields to create a supplier entry.

3. Edit an entry.

- On the page where you edit a supplier entry, notice that the URL path now includes supplier-update/1.

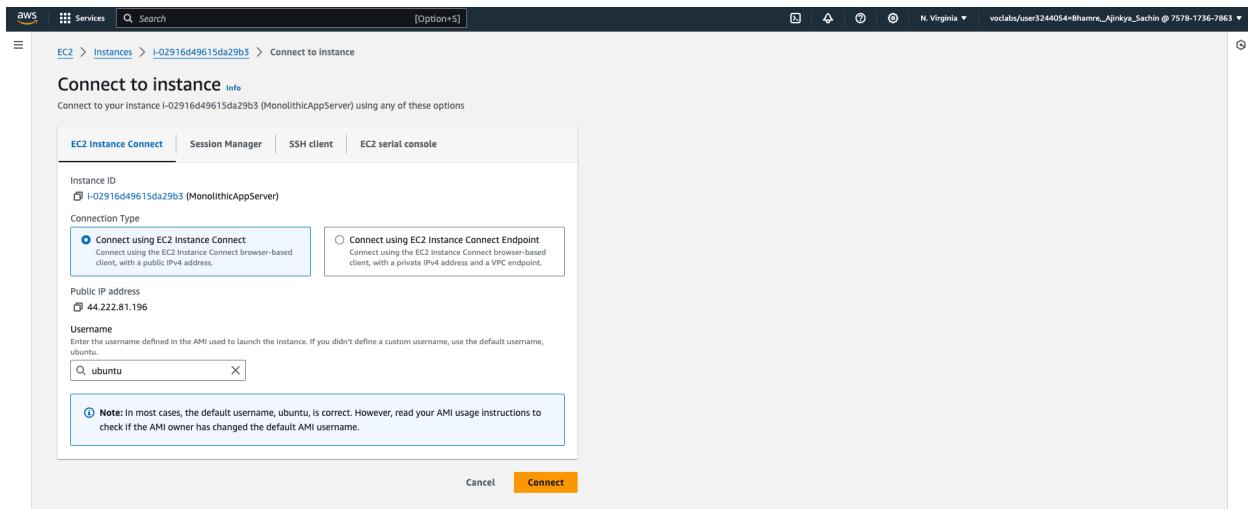
- Modify the record in some way and save the change.

Notice that the change was saved in the record.



Task 2.3: Analyze how the monolithic application runs

1. Use EC2 Instance Connect to connect to the MonolithicAppServer instance.



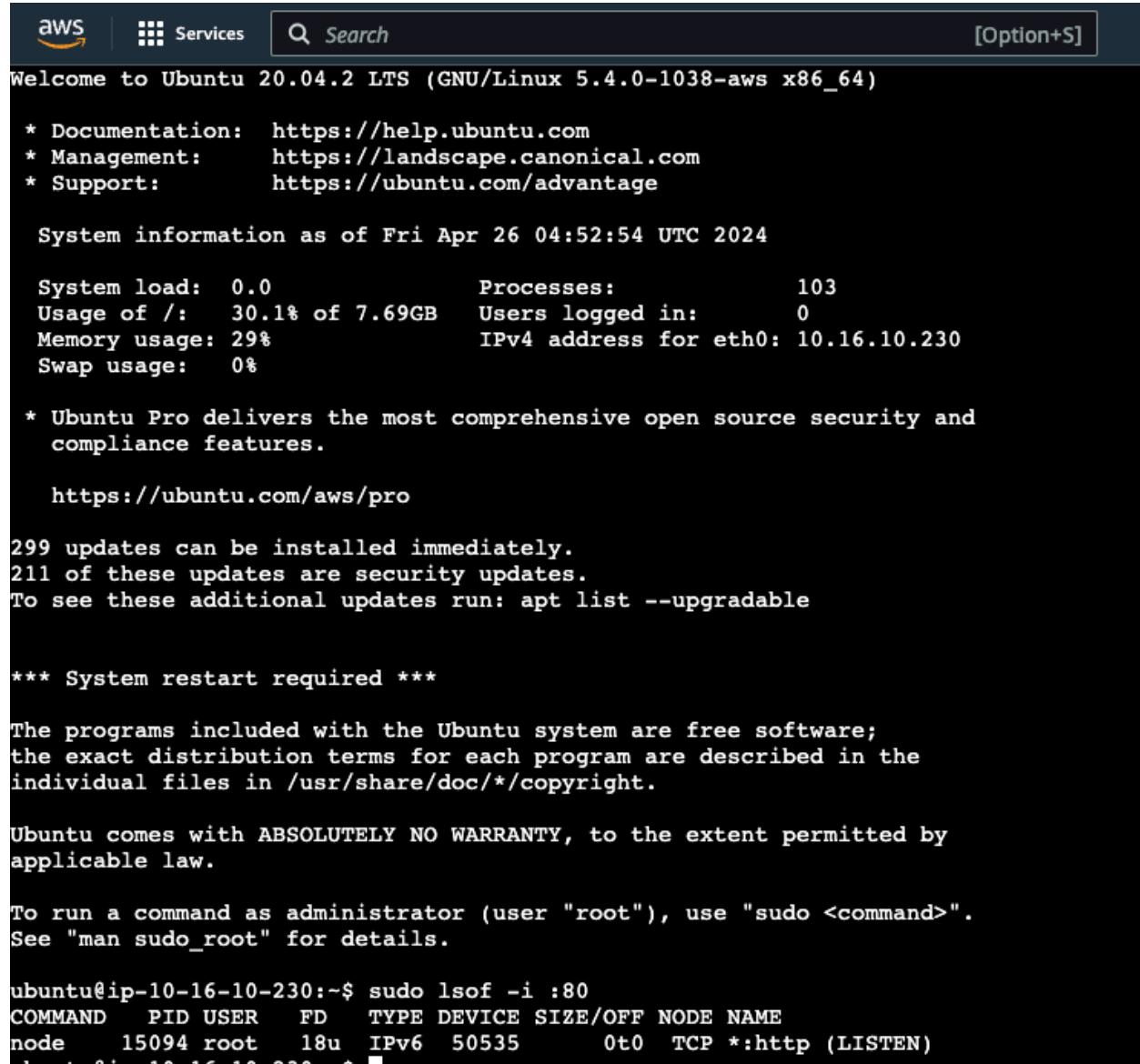
2. Analyze how the application is running.

sudo lsof -i :80 :

The command sudo lsof -i :80 is used to list all the processes currently listening on port 80 on the EC2 instance

- COMMAND: Indicates the name of the process listening on port 80. In this case, it's node.
- PID: Represents the process ID (PID) of the node process. It's 15094.
- USER: Specifies the user who owns the process. It's root.

- FD: Stands for File Descriptor and indicates the file descriptor number used by the process. It's 18u.
- TYPE: Describes the type of file or socket. Here, it's a TCP socket.
- DEVICE: Indicates the device number associated with the file. It's IPv6.
- SIZE/OFF: Represents the size of the file or offset within the file. It's 0t0.
- NODE: Specifies the node associated with the file. It's *:http (LISTEN).



```

aws | Services | Search | [Option+S]

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 Fri Apr 26 04:52:54 UTC 2024

 System load: 0.0          Processes: 103
 Usage of /: 30.1% of 7.69GB  Users logged in: 0
 Memory usage: 29%          IPv4 address for eth0: 10.16.10.230
 Swap usage: 0%

 * Ubuntu Pro delivers the most comprehensive open source security and
 compliance features.

 https://ubuntu.com/aws/pro

299 updates can be installed immediately.
211 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-10-16-10-230:~$ sudo lsof -i :80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 15094 root 18u IPv6 50535 0t0 TCP *:http (LISTEN)

```

From this output, we can see that there is a process with PID 15094 (owned by the root user) listening on port 80. The process name is node, and it's listening on all available IPv6 addresses (*) for incoming HTTP connections (http).

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

```
ubuntu@ip-10-16-10-230:~$ ps -ef | head -1; ps -ef | grep node
UID      PID  PPID  C STIME TTY          TIME CMD
root     15094     1  0 Apr25 ?        00:00:01 node index.js
ubuntu   16135  16101  0 05:02 pts/0    00:00:00 grep --color=auto node
```

There are two processes listed:

- The first process is the node process with PID 15094. It is owned by the root user and appears to be running index.js.
- The second process is the grep command itself, which is searching for the string "node" in the process list.

So, the node process is being run by the root user on this EC2 instance.

Regarding the PID matching, yes, the PID 15094 from the ps command output matches the PID 15094 from the previous output of the lsof command. Both commands indicate that the node process with PID 15094 is running on the EC2 instance.

3. To analyze the application structure, run the following commands:

```
ubuntu@ip-10-16-10-230:~$ cd ~/resources/codebase_partner
ubuntu@ip-10-16-10-230:~/resources/codebase_partner$ ls
app  index.js  node_modules  package-lock.json  package.json  public  views
```

Questions for thought: Based on what you have observed, what can you determine about how and where this node application is running? How do you think it was installed? What prerequisite libraries, if any, were required to make it run? Where does the application store data?

- The Node.js application is likely running on an Ubuntu server (ubuntu@ip-10-16-10-230) hosted on AWS EC2.
- It appears to be installed in the directory ~/resources/codebase_partner.
- It was likely installed using npm, as indicated by the presence of package-lock.json and package.json.
- Prerequisite libraries were installed via npm and are located in the node_modules directory.
- The application stores data in its directory or possibly in a database, depending on its functionality.

4. Connect a MySQL client to the RDS database that the node application stores data in.

i) Connect to the RDS database:

Use the AWS Management Console or AWS CLI to navigate to the RDS dashboard and find the endpoint of the RDS database instance where the node application stores data.

Copy the endpoint URL. (`RDSEndpoint supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com`)

ii) Verify database accessibility:

SSH into the MonolithicAppServer instance.

```
nmap -Pn supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com
```

```
aws | Services | Q Search [Option+S]
ubuntu@ip-10-16-10-230:~/resources/codebase_partner$ nmap -Pn supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-26 05:32 UTC
Nmap scan report for supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com (10.16.30.176)
Host is up (0.00056s latency).
rDNS record for 10.16.30.176: ip-10-16-30-176.ec2.internal
Not shown: 999 filtered ports
PORT      STATE SERVICE
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 6.56 seconds
ubuntu@ip-10-16-10-230:~/resources/codebase_partner$ █
```

This command checks if the RDS database is reachable from the MonolithicAppServer instance on the standard MySQL port (usually port 3306).

iii) Connect to the database:

```
mysql -u admin -p -h supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com
```

```
ubuntu@ip-10-16-10-230:~/resources/codebase_partner$ mysql -u admin -p -h supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 110
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> █
```

5. Observe the data in the database.

- From the mysql> prompt, run SQL commands as appropriate to see that a database named COFFEE contains a table named suppliers. This table contains the supplier entry or entries that you added earlier when you tested the web application.
- Exit the MySQL client and then close the EC2 Instance Connect tab. Also close the coffee suppliers web application tab.

```

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| COFFEE   |
| information_schema |
| inventory |
| mysql     |
| performance_schema |
| sys       |
+-----+
6 rows in set (0.01 sec)

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> SHOW TABLES;
+-----+
| Tables_in_COFFEE |
+-----+
| suppliers        |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM suppliers;
+----+-----+-----+-----+-----+-----+-----+
| id | name | address | city | state | email | phone |
+----+-----+-----+-----+-----+-----+-----+
| 3  | supplier-1 | supplier-address | supplier-city | supplier-city | supplier@gmail.com | +12012679334 |
+----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> EXIT;
Bye

```

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

In this phase, you will create a development environment by using AWS Cloud9. You will also check your application code into AWS CodeCommit, which is a Git-compatible repository.

Task 3.1: Create an AWS Cloud9 IDE as your work environment

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.

AWS Services Search [Option+S] N. Virginia v vclabs/user5244054=Bhamre_Ajinkya_Sachin @ 7578-1756-7863 ▾

AWS Cloud9 > Environments > Create environment

Create environment Info

Details

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

Description – optional Limit 200 characters.

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

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

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

New EC2 instance

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

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

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

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

AWS Services Search [Option+S]

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

Amazon Linux 2

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

30 minutes

Network settings Info

Connection
How your environment is accessed.

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

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

VPC settings Info
Amazon Virtual Private Cloud (VPC)

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

vpc-022ff6b5b8bdd27a5d
Name – LabVPC

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

subnet-0c59acf72ec6db8e
Name – Public Subnet1

Info You have chosen a public subnet for your Cloud9 environment, note the following:

- If accessing the EC2 instance directly through SSH, the instance can only be launched into a public subnet.
- You must attach an Internet gateway to the Amazon VPC so the SSM Agent for the instance can connect to Systems Manager.
- You must ensure that the public subnet has a route table with a minimum set of routes. [Find out more](#)

Tags – optional Info
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.

Info The following IAM resources will be created in your account

- AWSServiceRoleForAWSCloud9 – AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Find out more](#)

Cancel **Create**

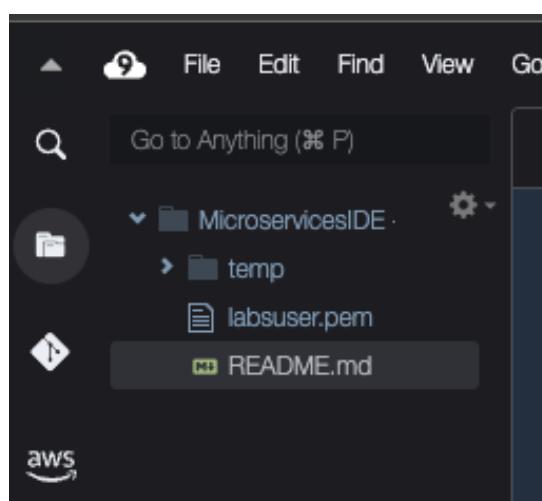
The screenshot shows the AWS Cloud9 interface. On the left, there's a sidebar with 'My environments', 'Shared with me', and 'All account environments'. The main area is titled 'Environments (1)' and shows a table with one row. The row contains the name 'MicroservicesIDE', a 'Cloud9 IDE' link, 'EC2 instance' status, 'Secure Shell (SSH)', 'Owner' permission, and an ARN. A success message at the top says 'Successfully created MicroservicesIDE. To get the most out of your environment, see Best practices for using AWS Cloud9'.

Task 3.2: Copy the application code to your IDE

- From the **AWS Details** panel on this lab instructions page, download the **labsuser.pem** file to local computer.
- Upload the .pem file to your AWS Cloud9 IDE, and use the Linux chmod command to set the proper permissions on the file so that you can use it to connect to an EC2 instance.

The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a sidebar with 'File', 'Edit', 'Find', 'Go', 'Run', 'Tools', 'Window', 'Support', and 'Preview'. The main area shows a file tree with 'MicroservicesIDE' containing 'labsuser.pem' and 'README.md'. A red box highlights this folder structure. Below the file tree, there's a 'Welcome' message, a 'Getting started' section with 'Create File', 'Upload Files...', and 'Clone from GitHub' options, and a 'Configure AWS Cloud9' section with theme and keyboard mode settings. At the bottom, there's a terminal window showing a bash session and a status bar with 'CodeWhisperer' and 'AWS: profile default'.

- Create a temp directory on the AWS Cloud9 instance at /home/ec2-user/environment/temp.

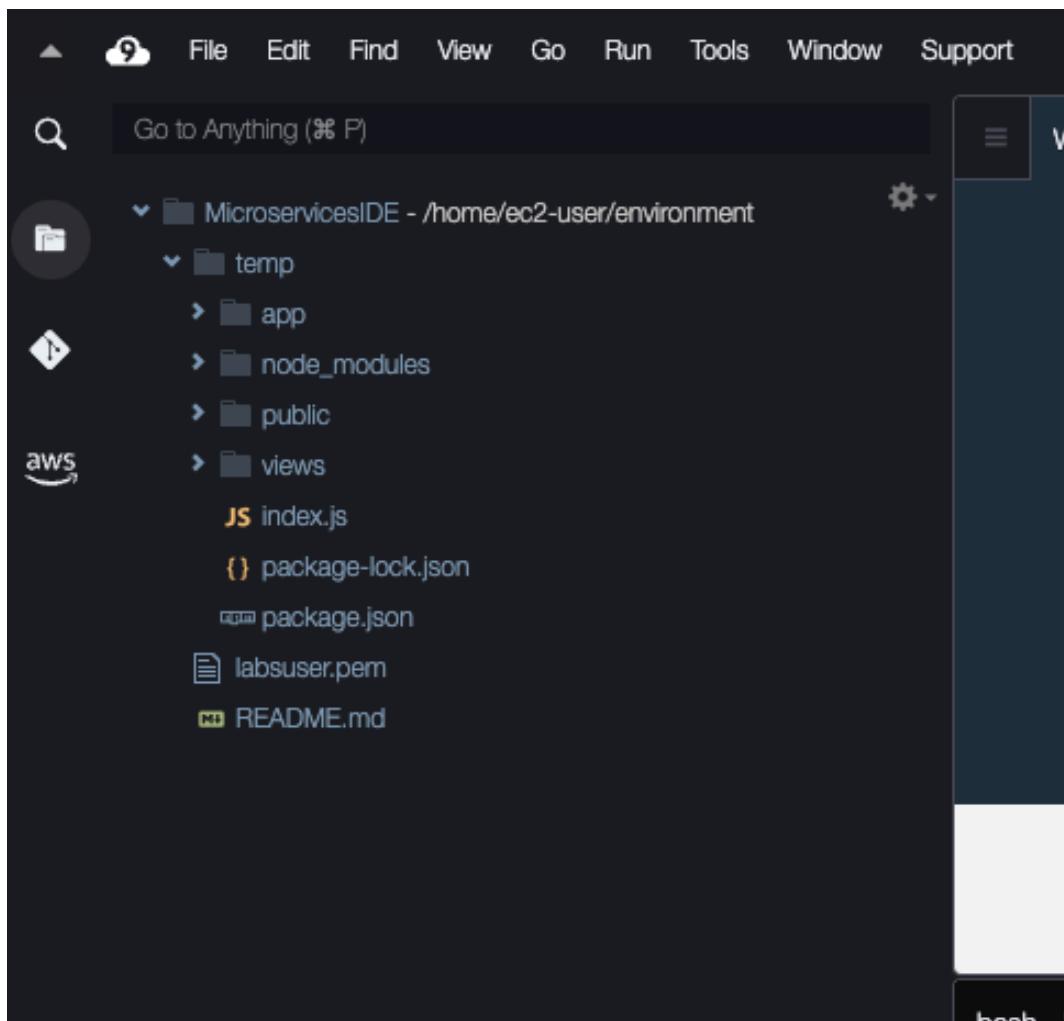


4. From the Amazon EC2 console, retrieve the private IPv4 address of the MonolithicAppServer instance.

Public IPv4 address : 44.222.81.196

5. Use the Linux scp command in the Bash terminal on the AWS Cloud9 instance to copy the source code for the node application from the MonolithicAppServer instance to the temp directory that you created on the AWS Cloud9 instance.

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



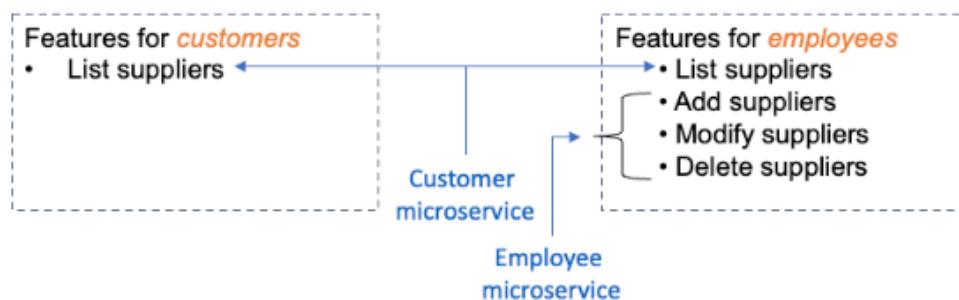
Task 3.3: Create working directories with starter code for the two microservices

In this task, you will create areas in your development environment to support separating the application logic into two different microservices.

Based on the solution requirements of this project, it makes sense to split the monolithic application into two microservices. You will name the microservices *customer* and *employee*.

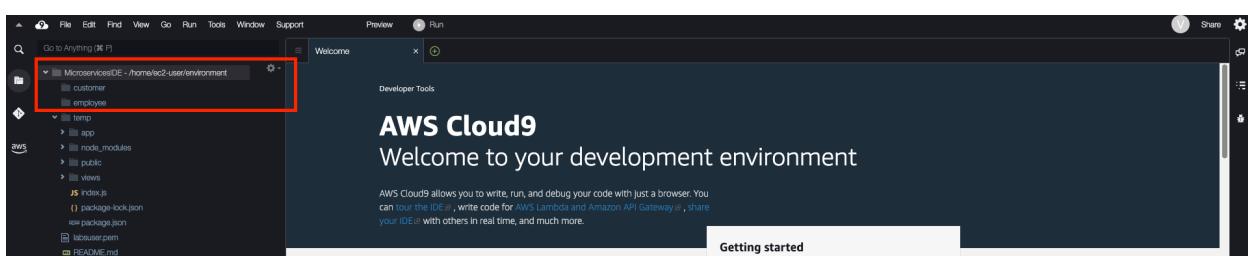
The following table explains the functionality that is needed for each microservice.

Primary User	Microservice Functionality	Access Level
Customer	The <i>customer</i> microservice will provide the functionality that customers (the café franchise location managers who want to buy coffee beans) need. The customers need a read-only view of the contact information for the suppliers to be able to buy coffee beans from them. You can think of the café franchise location managers as <i>customers</i> of the application.	Read-only
Employee	The <i>employee</i> microservice will provide the functionality that employees (the café corporate office employees) need. Employees need to add, modify, and delete suppliers who are listed in the application. Employees are responsible for keeping the listings accurate and up to date.	Read-write



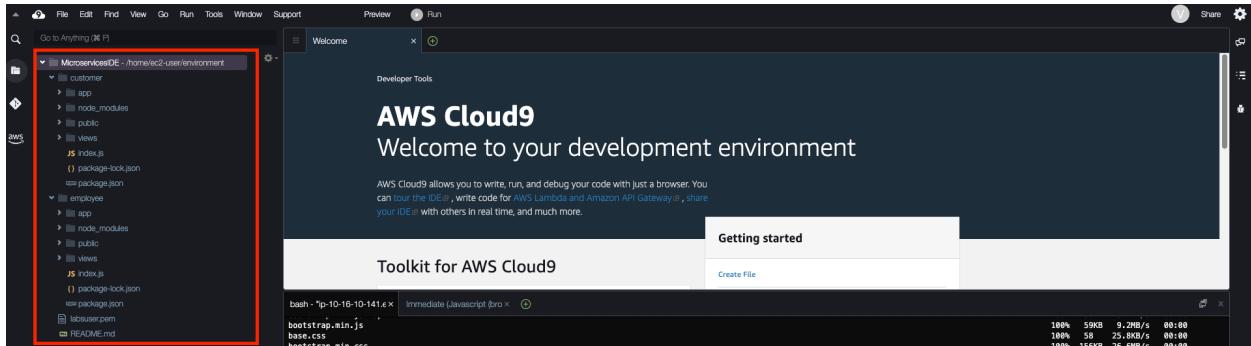
The employee microservice will eventually be made available only to employees. You will accomplish this by first encapsulating them as a separate microservice (in phases 3 and 4 of the project), and then later in phase 9 of the project, you will limit who can access the employee microservice.

1. In the microservices directory, create two new directories that are named customer and employee.



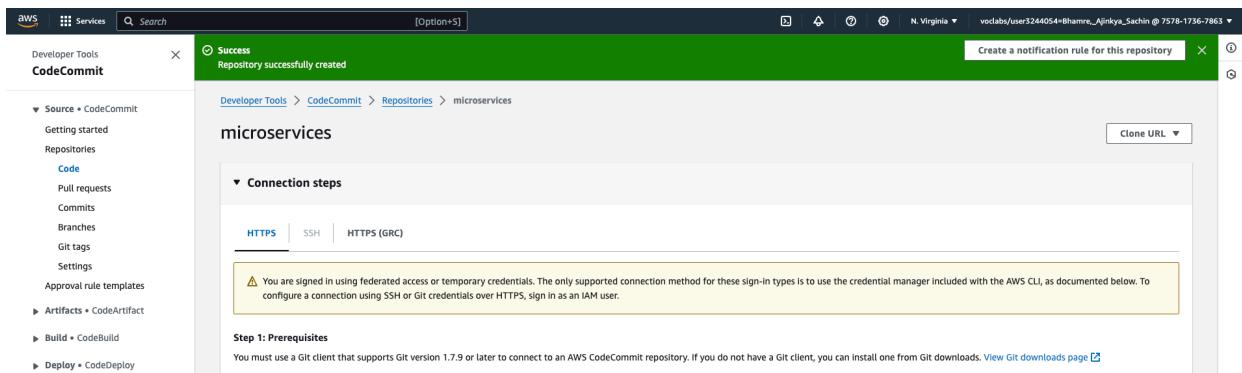
2. Place a copy of the source code for the monolithic application in each new directory, and remove the files from the temp directory.

3. Delete the empty temp directory.



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

1. Create a CodeCommit repository that is named microservices.



2. To check the unmodified application code into the microservices CodeCommit repository, run the following commands:

```
voclabs:~/environment/microservices (dev) $ git push -u origin dev
Enumerating objects: 2193, done.
Counting objects: 100% (2193/2193), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2085/2085), done.
Writing objects: 100% (2193/2193), 1.94 MiB | 5.28 MiB/s, done.
Total 2193 (delta 550), 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'.
```

3. Configure your Git client to know your username and email address.

```
vocabs:~/environment/microservices (dev) $ git config --global user.name "Ajinkya Bhamre"
vocabs:~/environment/microservices (dev) $ git config --global user.email "abhamre@stevens.edu"
```

4. In a new browser tab, browse to the CodeCommit console and observe that the code is now checked into your microservices repository.

The screenshot shows the AWS CodeCommit interface. On the left, there's a navigation sidebar with links like 'Getting started', 'Code', 'Pull requests', 'Commits', 'Branches', 'Git tags', 'Settings', 'Approval rule templates', 'Artifacts', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The main content area shows a repository named 'microservices'. It lists four files: 'customer', 'employee', 'labuser.pem', and 'README.md'. The 'README.md' file is expanded, showing its content:

```
Hi there! Welcome to AWS Cloud9!
To get started, create some files, play with the terminal, or visit https://docs.aws.amazon.com/console/cloud9/ for our documentation.
Happy coding!
```

Phase 4: Configuring the application as two microservices and testing them in Docker containers

Task 4.1: Adjust the AWS Cloud9 instance security group settings

security group associated with your AWS Cloud9 EC2 instance is configured to allow inbound network traffic on TCP ports 8080 and 8081.

Task 4.2: Modify the source code of the *customer* microservice

```

1  const Supplier = require("../models/supplier.model.js");
2  const {body, validationResult} = require("express-validator");
3
4  /*eslint */
32
33  exports.findAll = (req, res) => {
34      Supplier.getAll((err, data) => {
35          if (err)
36              res.render("500", {message: "There was a problem retrieving the list of suppliers"});
37          else res.render("supplier-list-all", {suppliers: data});
38      });
39  };
40
41  exports.findOne = (req, res) => {
42      Supplier.findById(req.params.id, (err, data) => {
43          if (err) {
44              if (err.kind === "not_found") {
45                  res.status(404).send({
46                      message: `Not found Supplier with id ${req.params.id}.`);
47                  });
48              } else {
49                  res.render("500", {message: `Error retrieving Supplier with id ${req.params.id}`});
50              }
51          } else res.render("supplier-update", {supplier: data});
52      );
53  };
54
55
56  /*eslint */

```

Code editor showing supplier.model.js:

```

1  const mysql = require("mysql2");
2  const dbConfig = require("../config/config");
3
4  const Supplier = function (supplier) {
5      /* */
6
7      Supplier.getAll = result => {
8          db_connection.query("SELECT * FROM suppliers", (err, res) => {
9              if (err) {
10                  console.log("error: ", err);
11                  result(err, null);
12                  return;
13              }
14              console.log("suppliers: ", res);
15              result(null, res);
16          });
17      };
18
19
20      Supplier.findById = (supplierId, result) => {
21          db_connection.query(`SELECT * FROM suppliers WHERE id = ${supplierId}`, (err, res) => {
22              if (err) {
23                  console.log("error: ", err);
24                  result(err, null);
25                  return;
26              }
27              if (res.length) {
28                  console.log("found supplier: ", res[0]);
29                  result(null, res[0]);
30                  return;
31              }
32              result({kind: "not_found"}, null);
33          });
34      };
35
36      /*
37          Supplier.updateById = (id, supplier, result) => { };
38
39          Supplier.deleteById = (id, result) => { };
40
41          Supplier.removeAll = result => { };
42
43      */
44
45      module.exports = Supplier;
46

```

Code editor showing nav.html:

```

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

```

A screenshot of a code editor showing a template file. The file contains HTML code with Mustache syntax for rendering a table of suppliers. The code includes a header section, a table with columns for Name, Address, City, State, Email, and Phone, and a footer section.

```
1 {{>header}}
2 <div class="container">
3   {{>nav}}
4   <h1>All suppliers</h1>
5   <table class="table table-hover">
6     <thead>
7       <tr>
8         <th scope="col">Name</th>
9         <th scope="col">Address</th>
10        <th scope="col">City</th>
11        <th scope="col">State</th>
12        <th scope="col">Email</th>
13        <th scope="col">Phone</th>
14      <!-- <th scope="col"></th> -->
15    </tr>
16  </thead>
17  <tbody>
18    {{#suppliers}}
19      <tr>
20        <th scope="row">{{name}}</th>
21        <td>{{address}}</td>
22        <td>{{city}}</td>
23        <td>{{state}}</td>
24        <td>{{email}}</td>
25        <td>{{phone}}</td>
26        <!-- <td><h4><span class="badge badge-info"><a class="text-light" href="/supplier-update/{{id}}>edit</a></span></h4></td>
27        -->
28      </tr>
29    {{/suppliers}}
30  </tbody>
31 </table>
32 <!-- <h4><a class="badge badge-success" href="/supplier-add">Add a new supplier</a></h4> -->
33
34 </div>
35 {{>footer}}
```

A screenshot of a code editor showing an Express.js application configuration file named index.js. The file sets up the view engine to use Mustache, defines routes for listing suppliers, adding suppliers, updating suppliers, and deleting suppliers, and handles 404 errors. It also specifies a port of 8080 and logs the server's port.

```
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("/", (req, res) => {
23   res.render("home", {});
24 });
25 app.get("/suppliers/", supplier.findAll);
26 // show the add supplier form
27 //app.get("/supplier-add", (req, res) => {
28 //  res.render("supplier-add", {});
29 //});
30 // receive the add supplier POST
31 //app.post("/supplier-add", supplier.create);
32 // show the update form
33 //app.get("/supplier-update/:id", supplier.findOne);
34 // receive the update POST
35 //app.post("/supplier-update", supplier.update);
36 // receive the POST to delete a supplier
37 //app.post("/supplier-remove/:id", supplier.remove);
38 // handle 404
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 || 8080
46 app.listen(app_port, () => {
47   console.log(`Server is running on port ${app_port}.`);
48 })
```

1. Edit the supplier.controller.js file:

- Removed unnecessary functions related to adding, updating, and deleting suppliers.
- Kept only the read-only functions findAll and findOne.

2. Edit the supplier.model.js file:

- Removed unnecessary functions related to adding, updating, and deleting suppliers.
- Kept only the functions getAll and findByld.

3. Edit the nav.html file:

- Changed "Monolithic Coffee suppliers" to "Coffee suppliers".
- Changed "Home" to "Customer home".
- Added a new link for administrators: Administrator link.

4. Edit the supplier-list-all.html file:

- Removed the "Add a new supplier" button.
- Removed the edit buttons next to supplier rows.

5. Delete unnecessary HTML files:

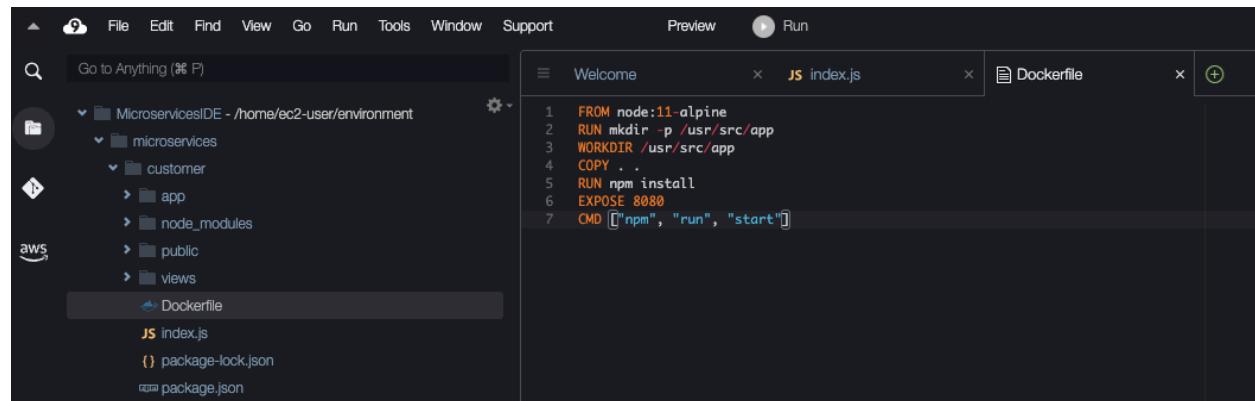
- Deleted supplier-add.html, supplier-form-fields.html, and supplier-update.html from the customer/views directory.

6. Edit the index.js file:

- Commented out unnecessary routes related to adding, updating, and deleting suppliers.
- Changed the port number to 8080 to match the Docker container configuration.

Task 4.3: Create the **customer** microservice Dockerfile and launch a test container

1. In the **customer** directory, create a new file named Dockerfile that contains the following code:



```

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

```

```
Ajin@voclabs:~/environment/microservices (dev) $ cd customer
Ajin@voclabs:~/environment/microservices/customer (dev) $ docker build --tag customer .    edu
Sending build context to Docker daemon  9.008MB
Step 1/7 : FROM node:11-alpine
11-alpine: Pulling from library/node
e7c96db7181b: Pull complete
0119aca44649: Pull complete
40df19605a18: Pull complete
82194b8b4a64: Pull complete
Digest: sha256:8bb56bab197299c8ff820f1a55462890caf08f57ffe3b91f5fa6945a4d505932
Status: Downloaded newer image for node:11-alpine
    --> f18da2f58c3d
Step 2/7 : RUN mkdir -p /usr/src/app
    --> Running in dfa84ea13147
Removing intermediate container dfa84ea13147
    --> 5613e0a99cc6
Step 3/7 : WORKDIR /usr/src/app
    --> Running in 4d7f50a1ce30
Removing intermediate container 4d7f50a1ce30
    --> 5977b0cdf719
Step 4/7 : COPY .
    --> 7a5300ced89d
Step 5/7 : RUN npm install
    --> Running in cb2b775cea69
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.858s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container cb2b775cea69
    --> 32ff33c2c09b
Step 6/7 : EXPOSE 8080
    --> Running in 50873c7b933b
Removing intermediate container 50873c7b933b
    --> 6628e59812cb
Step 7/7 : CMD ["npm", "run", "start"]
    --> Running in fba762324a8b
Removing intermediate container fba762324a8b
    --> a3bf4ccb12df
Successfully built a3bf4ccb12df
Successfully tagged customer:latest
```

The Docker build process has completed successfully. Here's a breakdown of what happened:

1. The Docker daemon pulled the Node.js 11 Alpine image from the Docker Hub repository.
2. A directory named /usr/src/app was created inside the container.
3. The working directory was set to /usr/src/app.
4. The current directory contents were copied into the container's working directory.
5. The npm install command was executed to install dependencies for the Node.js application.
6. The container was instructed to expose port 8080.
7. Finally, the container was configured to run the application using the command npm run start. The Docker image with the tag customer:latest has been successfully built.

3. Verify that the customer-labeled Docker image was created.

docker images

```
voclabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY      TAG          IMAGE ID      CREATED       SIZE
customer        latest       a3bf4ccb12df  15 minutes ago  82.7MB
node            11-alpine   f18da2f58c3d  4 years ago   75.5MB
```

i. Set the dbEndpoint variable using the provided command:

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

ii. Verify that the dbEndpoint variable is set correctly by running:

```
voclabs:~/environment/microservices/customer (dev) $ dbEndpoint=$(cat ~/environment/microservices/customer/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/customer (dev) $ echo $dbEndpoint
supplierdb.crrzlncupqd3.us-east-1.rds.amazonaws.com
```

iii. If the dbEndpoint variable is set correctly, you can launch the Docker container using the following command

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

```
voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer
45eda30c632ec2395c7eb84123a75542a6c304e079b67b367f176ddf82b6a4d5
voclabs:~/environment/microservices/customer (dev) $
```

This command will launch a Docker container named customer_1 from the customer image, expose port 8080 on AWS Cloud9 instance, and pass the database host location as an environment variable to the container. The container will run in the background (-d flag).

5. Check which Docker containers are currently running on the AWS Cloud9 instance.

docker ps

```
voclabs:~/environment/microservices/customer (dev) $ docker ps
CONTAINER ID        IMAGE           COMMAND       CREATED          STATUS          PORTS          NAMES
45eda30c632e        customer        "docker-entrypoint.s..."  5 minutes ago   Up 5 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   customer_1
voclabs:~/environment/microservices/customer (dev) $
```

6. Verify that the customer microservice is running in the container and working as intended.

The screenshot shows a web browser window with the URL <https://41674d3c53844e8ea1003292f70fb7d.vfs.cloud9.us-east-1.amazonaws.com/>. The page title is "Coffee suppliers". The header includes links for "Customer home", "Suppliers list", and "Administrator link". A banner image shows two cups of coffee on a espresso machine. Below the banner, a "Welcome" message says "Use this app to keep track of your coffee suppliers" and a "List of suppliers" button.

The screenshot shows the "All suppliers" page with the URL <https://41674d3c53844e8ea1003292f70fb7d.vfs.cloud9.us-east-1.amazonaws.com/suppliers>. The page title is "Coffee suppliers". The header includes links for "Customer home", "Suppliers list", and "Administrator link". A banner image shows two cups of coffee on a espresso machine. Below the banner, the heading "All suppliers" is displayed, followed by a table with one row:

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334

The screenshot shows the "admin/suppliers" page with the URL <https://41674d3c53844e8ea1003292f70fb7d.vfs.cloud9.us-east-1.amazonaws.com/admin/suppliers>. The page title is "Coffee suppliers". The header includes links for "Customer home", "Suppliers list", and "Administrator link". A banner image shows two cups of coffee on a espresso machine. Below the banner, the heading "404" is displayed with a small globe icon, followed by the message "Sorry, we don't seem to have that page in stock" and a "Return to home page" button.

7. Commit and push your source code changes into CodeCommit.
8. Optional: Observe the commit details in the CodeCommit console.

Ajinkya Bhamre - 20017896

Email : abhamre@stevens.edu

```
voclabs:~/environment/microservices/customer (dev) $ git log  
commit 340b8d3f7954f586629466265678437d82a479c8 (HEAD -> dev)  
Author: Ajinkya Bhamre <abhamre@stevens.edu>  
Date:   Fri Apr 26 22:36:55 2024 +0000
```

customer-container-hosted

```
commit f33356aa82e6bc27741d19f496e5856f7c27d439 (origin/dev)  
Author: EC2 Default User <ec2-user@ip-10-16-10-141.ec2.internal>  
Date:   Fri Apr 26 07:34:52 2024 +0000
```

two unmodified copies of the application code

```
voclabs:~/environment/microservices/customer (dev) $ git push origin dev  
Enumerating objects: 24, done.  
Counting objects: 100% (24/24), done.  
Delta compression using up to 2 threads  
Compressing objects: 100% (12/12), done.  
Writing objects: 100% (13/13), 1.35 KiB | 1.35 MiB/s, done.  
Total 13 (delta 7), reused 0 (delta 0), pack-reused 0  
remote: Validating objects: 100%  
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices  
  f33356a..340b8d3  dev -> dev
```

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

1. supplier.controller.js:

Prepend "/admin" to all redirect calls.

2. index.js:

Prepend "/admin" to all app.get and app.post calls.

Change the port number to 8081.

3. supplier-add.html and supplier-update.html:

Prepend "/admin" to the form action paths.

4. supplier-list-all.html and home.html:

Prepend "/admin" to the HTML paths.

5. header.html:

Modify the title to "Manage coffee suppliers".

6. nav.html:

Change "Monolithic Coffee suppliers" to "Manage coffee suppliers" in the navbar title.

Replace the existing line of code with Administrator home.

Add a new line with Customer home after line 8.

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

```

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

```

```

voclabs:~/environment/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.014MB
Step 1/7 : FROM node:11-alpine
Step 1/7 : WORKDIR /usr/src/app
Step 2/7 : COPY . .
Step 3/7 : RUN npm install
Step 4/7 : EXPOSE 8081
Step 5/7 : CMD [ "npm", "run", "start" ]
Step 6/7 : 
Step 7/7 : 
Successfully built f6ed3dabf8
Successfully tagged employee:latest
voclabs:~/environment/employee (dev) $ dbEndpoint=$(cat ~/environment/customer/app/config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
voclabs:~/environment/employee (dev) $ echo $dbEndpoint
supplierdb.czr1cupcqd.us-east-1.rds.amazonaws.com
voclabs:~/environment/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 --env APP_DB_HOST=$dbEndpoint employee
0b5c7741d1caem12cf5810d4914c33163ffbad08748e6583dc687134275ff9d
voclabs:~/environment/employee (dev) $ curl "Employee microservice test container running at http://$(curl ifconfig.me):8081/admin/suppliers"
Total % Received % Xferd Average Speed   Time  Current
          Dload Upload Total  Left Speed
100  12 100 12  0    0  298  0 --:--:--:--:--:-- 307
Employee microservice test container running at http://3.238.89.129:8081/admin/suppliers
voclabs:~/environment/employee (dev) $

```

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334

All fields are required

Name
newSupplier

Name of this supplier

Address
newSupplier-address

Address for this supplier

City
newSupplier-city

City for this supplier

State
newSupplier-State

State for this supplier

Email
newSupplier@gmail.com

Email for this supplier

Phone
+12012657889

Phone number for this supplier

Submit

Not Secure 3.238.89.129:8081/admin/suppliers

Return Confirmation Prime Video PowerPoint Presentations Royalty Free Stock Authentic Stock Photo Alamy – Stock Photography Commercial Photography Outlook Nicely Creative Gradients font pairing typekit Google Fonts

Administrator home
Suppliers list
Customer home

All suppliers

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334 edit
newSupplier	newSupplier-address	newSupplier-city	newSupplier-State	newSupplier@gmail.com	+12012657889 edit

Add a new supplier

Links that should take to the customer microservice will not work. For example, if you choose Customer home or Suppliers list, the pages won't be found because the link assumes that the customer microservice also runs on port 8081 (but it doesn't). You can ignore this issue—these links should work as intended when you deploy the microservices to Amazon ECS later.

Test editing an existing supplier.

Not Secure 3.238.89.129:8081/admin/suppliers

Return Confirmation Prime Video PowerPoint Presentations Royalty Free Stock Photos Authentic Stock Photo Fotolia - Sell and... Alamy - Stock Photos Commercial Photos Outlook Nice | The creative... Gradients font pairing typekit Google Fonts

The screenshot shows a web application titled "Manage coffee suppliers". At the top, there's a navigation bar with links to "Administrator home", "Suppliers list", and "Customer home". Below the header, a section titled "All suppliers" displays a table with two rows of supplier data. Each row includes columns for Name, Address, City, State, Email, and Phone, along with an "edit" button. A green "Add a new supplier" button is located at the bottom of the table.

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334
newSupplier-edited	newSupplier-address-edited	newSupplier-city-edited	newSupplier-State-edited	newSupplieredited@gmail.com	+12012657889

Add a new supplier

Test deleting an existing supplier:

Not Secure 3.238.89.129:8081/admin/supplier-update/2

Return Confirmation Prime Video PowerPoint Presentations Royalty Free Stock Photos Authentic Stock Photo Fotolia - Sell and... Alamy - Stock Photos Commercial Photos Outlook Nice | The creative... Gradients font pairing typekit Google Fonts

A modal dialog box titled "Delete supplier" is centered on the screen. It contains the question "Are you sure you want to delete supplier newSupplier-edited?". Below the question are two buttons: "Close" and "Delete this supplier".

All fields are required

Name
newSupplier-edited
Name of this supplier

Address
newSupplier-address-edited
Address for this supplier

City
newSupplier-city-edited
City for this supplier

State
newSupplier-State-edited
State for this supplier

Email
newSupplieredited@gmail.com
Email for this supplier

Phone
+12012657889
Phone number for this supplier

Submit

Delete this supplier

Not Secure 3.238.89.129:8081/admin/suppliers

Return Confirmation Prime Video PowerPoint Presentations Royalty Free Stock Photos Authentic Stock Photo Fotolia - Sell and... Alamy - Stock Photos Commercial Photos Outlook Nice | The creative... Gradients font pairing typekit Google Fonts

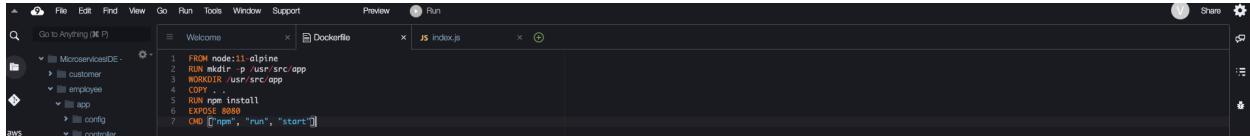
The screenshot shows the "Manage coffee suppliers" page again. The "All suppliers" section now only contains one row for "supplier-1". The "edit" button for this supplier is visible. A green "Add a new supplier" button is at the bottom.

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334

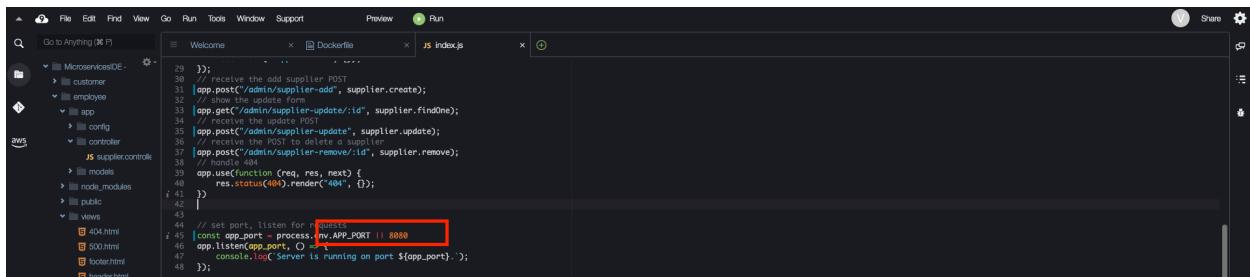
Add a new supplier

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

1. Edit the employee/index.js and employee/Dockerfile files to change the port from 8081 to 8080
2. Rebuild the Docker image for the employee microservice.



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



```
...  
42 // receive the add supplier POST  
43 app.post('/admin/supplier-add', supplier.create);  
44 // show the update form  
45 app.get('/admin/supplier-update/:id', supplier.findOne);  
46 app.post('/admin/supplier-update', supplier.update);  
47 // receive the POST to delete a supplier  
48 app.post('/admin/supplier-remove/:id', supplier.remove);  
49  
50 app.use(function (req, res, next) {  
51   res.status(404).render('404');  
52 })  
53 // set port. Listen for requests  
54 const app_port = process.env.APP_PORT || 8080  
55 app.listen(app_port, () => {  
56   console.log(`Server is running on port ${app_port}.`);  
57 });  
58 };
```

```
voclabs:~/environment/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.014MB
Step 1/7 : FROM node:11-alpine
--> f18da2f58c3d
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> be763737fb95
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> f9013de15f16
Step 4/7 : COPY .
--> fec59d2b0729
Step 5/7 : RUN npm install
--> Running in eff62845b51e
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.7s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container eff62845b51e
--> ddd3b9c9446a
Step 6/7 : EXPOSE 8080
--> Running in 61758935e2c1
Removing intermediate container 61758935e2c1
--> e8b5eb74b089
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in 039faeeb1c65
Removing intermediate container 039faeeb1c65
--> 06808d35c3e8
Successfully built 06808d35c3e8
Successfully tagged employee:latest
```

```
voclabs:~/environment/employee (dev) $ cd ..
voclabs:~/environment (dev) $ git add .
voclabs:~/environment (dev) $ git commit -m "emp-docker-index-changed"
[dev eb429fa] emp-docker-index-changed
25 files changed, 44 insertions(+), 21 deletions(-)
create mode 100644 .c9/metadata/environment/customer/Dockerfile
create mode 100644 .c9/metadata/environment/customer/app/controller/supplier.controller.js
create mode 100644 .c9/metadata/environment/customer/app/models/supplier.model.js
create mode 100644 .c9/metadata/environment/customer/index.js
create mode 100644 .c9/metadata/environment/customer/views/nav.html
create mode 100644 .c9/metadata/environment/customer/views/supplier-list-all.html
create mode 100644 .c9/metadata/environment/employee/Dockerfile
create mode 100644 .c9/metadata/environment/employee/app/controller/supplier.controller.js
create mode 100644 .c9/metadata/environment/employee/index.js
create mode 100644 .c9/metadata/environment/employee/views/header.html
create mode 100644 .c9/metadata/environment/employee/views/home.html
create mode 100644 .c9/metadata/environment/employee/views/nav.html
create mode 100644 .c9/metadata/environment/employee/views/supplier-add.html
create mode 100644 .c9/metadata/environment/employee/views/supplier-list-all.html
create mode 100644 .c9/metadata/environment/employee/views/supplier-update.html
create mode 100644 employee/Dockerfile
voclabs:~/environment (dev) $ git push origin dev
Enumerating objects: 50, done.
Counting objects: 100% (50/50), done.
Delta compression using up to 2 threads
Compressing objects: 100% (41/41), done.
Writing objects: 100% (43/43), 10.35 KiB | 2.07 MiB/s, done.
Total 43 (delta 15), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
  ffaa69a..eb429fa  dev -> dev
```

Task 4.7: Check code into CodeCommit

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

Task 5.1: Create ECR repositories and upload the Docker images

1. To authorize your 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

2. Create a separate private ECR repository for each microservice.
3. Set permissions on the customer ECR repository.
4. Use the same approach to set the same permissions on the employee ECR repository.
5. Tag the Docker images with your unique registryId (account ID) value to make it

The screenshot shows the Amazon ECR interface. In the top navigation bar, 'Amazon ECR' > 'Private registry' > 'Repositories'. Below this, the heading 'Private repositories' is displayed. A blue banner at the top indicates 'Upgraded basic scanning' with a link to 'Find out more'. On the right, there are buttons for 'switch' and 'X'. The main area is titled 'Repositories (2)' with a search bar labeled 'Filter status'. Below is a table with columns: Repository name, URI, Created at, Tag immutability, Scan frequency, and Encryption type. Two entries are listed:

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	757817367863.dkr.ecr.us-east-1.amazonaws.com/customer	27 April 2024, 18:56:02 (UTC-04)	Disabled	Manual	AES-256
employee	757817367863.dkr.ecr.us-east-1.amazonaws.com/employee	27 April 2024, 18:56:17 (UTC-04)	Disabled	Manual	AES-256

easier to manage and keep track of these images.

The screenshot shows a modal dialog titled 'Edit JSON'. The content area contains a JSON policy document:

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

At the bottom of the dialog, there are three buttons: 'Reset', 'Close', and a prominent orange 'Save' button.

```
voclabs:~/environment (dev) $ account_id=$(aws sts get-caller-identity | grep Account|cut -d '"' -f4)
voclabs:~/environment (dev) $ echo $account_id
757817367863
voclabs:~/environment (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
voclabs:~/environment (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
voclabs:~/environment (dev) $ docker images
REPOSITORY                                     TAG      IMAGE ID      CREATED        SIZE
757817367863.dkr.ecr.us-east-1.amazonaws.com/employee   latest    06808d35c3e8  2 hours ago  82.7MB
757817367863.dkr.ecr.us-east-1.amazonaws.com/customer   latest    dc7b5c34ee66  3 hours ago  82.7MB
customer                                         latest    dc7b5c34ee66  3 hours ago  82.7MB
node                                              11-alpine f18da2f58c3d  4 years ago  75.5MB
```

6. Run the appropriate docker command to push each of the Docker images to Amazon ECR.

Set account_id variable

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

Push customer image to Amazon ECR

```
docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
```

```
voclabs:~/environment (dev) $ account_id=$(aws sts get-caller-identity | grep Account | cut -d '"' -f4)
voclabs:~/environment (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
The push refers to repository [757817367863.dkr.ecr.us-east-1.amazonaws.com/customer]
7dbc3c445ef4: Pushed
5dd1c0aeceb5: Pushed
49777b716811: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:0082b7027da55e639e1fe79b3e7fcf824d29f9669ac4df74af9b9cacbd2c15e5 size: 1783
voclabs:~/environment (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [757817367863.dkr.ecr.us-east-1.amazonaws.com/employee]
ebe5c42bc739: Pushed
df7ba9d67f17: Pushed
49777b716811: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:1266f26b0c6bcd597705ca903ecdee863110fc5186bad1cb9de23ab63096ed7b size: 1783
```

customer

Images (1)						View push commands	Edit	
<input type="text"/> Search artefacts						Delete	Details	Scan
Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest			
latest	Image	27 April 2024, 19:11:51 (UTC-04)	27.70	Copy URI	sha256:0082b7027da55e639e1fe79b3e7fcf824d29f9669ac4df74af9b9cacbd2c15e5...			

Push employee image to Amazon ECR

```
docker push
$account_id.dkr.ecr.us-
east-1.amazonaws.com/
employee:latest
```

employee

Images (1)						View push commands	Edit	
<input type="text"/> Search artefacts						Delete	Details	Scan
Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest			
latest	Image	27 April 2024, 19:12:09 (UTC-04)	27.70	Copy URI	sha256:1266f26b0c6bcd597705ca903ecdee...			

7. Confirm that the two

images are now stored in Amazon ECR and that each has the latest label applied.

Timestamp	Logical ID	Status	Detailed status	Status reason
2024-04-27 19:35:22 UTC-0400	Infra-ECS-Cluster-microservices-serverlesscluster-c6f1a25a	CREATE_COMPLETE	-	-
2024-04-27 19:35:21 UTC-0400	ECSCluster	CREATE_COMPLETE	-	-
2024-04-27 19:34:43 UTC-0400	Infra-ECS-Cluster-microservices-serverlesscluster-c6f1a25a	CREATE_IN_PROGRESS	CONFIGURATION_COMPLETE	Eventual consistency check initiated
2024-04-27 19:34:43 UTC-0400	ECSCluster	CREATE_IN_PROGRESS	CONFIGURATION_COMPLETE	Eventual consistency check initiated
2024-04-27 19:34:42 UTC-0400	ECSCluster	CREATE_IN_PROGRESS	-	Resource creation initiated
2024-04-27 19:34:37 UTC-0400	ECSCluster	CREATE_IN_PROGRESS	-	-
2024-04-27 19:34:35 UTC-0400	Infra-ECS-Cluster-microservices-serverlesscluster-c6f1a25a	CREATE_IN_PROGRESS	-	User Initiated

Task 5.2: Create an ECS cluster

Success
Repository successfully created

Developer Tools > CodeCommit > Repositories > deployment

deployment

Clone URL

Connection steps

HTTPS SSH HTTPS (GRC)

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.

Step 1: Prerequisites
You must use a Git client that supports Git version 1.7.9 or later to connect to an AWS CodeCommit repository. If you do not have a Git client, you can install one from Git downloads page.

Step 2: Set up the AWS CLI Credential Helper
Set up your connection to AWS CodeCommit repositories using the credential helper included in the AWS CLI. This is the only connection method for AWS CodeCommit repositories that does not require an IAM user, so it is the only method that supports root access, federated access, and temporary credentials.

Additional details
You can find more detailed instructions in the documentation. View documentation

deployment info

Add file

Name

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

Task 5.3: Create a CodeCommit repository to store deployment file

Now you have a CodeCommit repository named "deployment" initialized with a branch named "dev". You can use this repository to store deployment configuration files for Amazon ECS and AppSpec specification files for CodeDeploy.

Task 5.4: Create task definition files for each microservice and register them with Amazon ECS

1. In the new deployment directory, create an empty file named taskdef-customer.json
2. Edit the taskdef-customer.json file.
- 3 .To 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"
```

```
{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:us-east-1:757817367863:task-definition/customer-microservice:2",
    "containerDefinitions": [
      {
        "name": "customer",
        "image": "customer",
        "cpu": 0,
        "portMappings": [
          {
            "containerPort": 8080,
            "hostPort": 8080,
            "protocol": "tcp"
          }
        ],
        "essential": true,
        "environment": [
          {
            "name": "APP_DB_HOST",
            "value": "supplierdb.crrzlnocupqd3.us-east-1.rds.amazonaws.com"
          }
        ],
        "mountPoints": []
      }
    ]
  }
}
```

The screenshot shows the AWS Elastic Container Service (ECS) Task definitions page. On the left, there's a sidebar with options like Clusters, Namespaces, Task definitions, and Account settings. The main area displays 'Task definitions (2) info'. It includes a search bar, a filter by status dropdown set to 'Active', and a table with two rows. Each row represents a task definition: 'Customer-microservice' and 'employee-microservice', both marked as 'ACTIVE'.

Task 5.5: Create AppSpec files for CodeDeploy for each microservice

1. Create an AppSpec file for the customer microservice.
2. In the same directory, create an AppSpec file for the employee microservice.

```

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

```

```

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

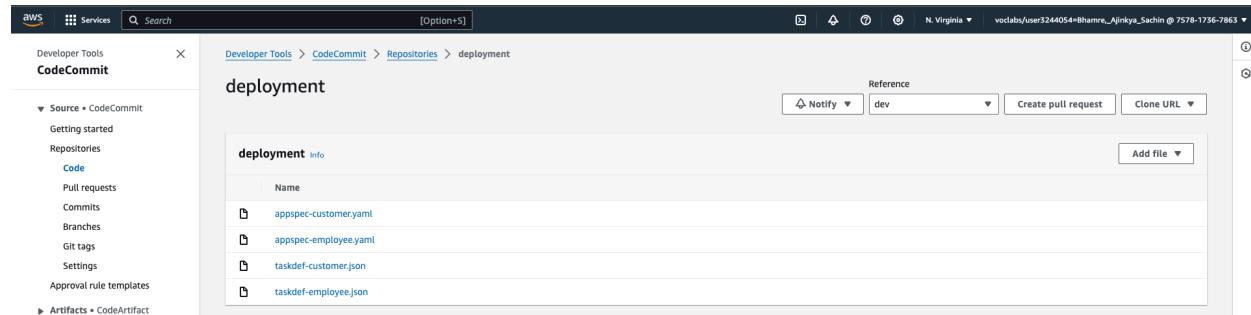
```

Task 5.6: Update files and check them into CodeCommit

1. Edit the taskdef-customer.json file.
2. Edit the taskdef-employee.json file.
3. Push all four files to CodeCommit.

```
vclabs:~/environment/deployment (dev) $ git add .
vclabs:~/environment/deployment (dev) $ git commit -m "Updated task definition files and added AppSpec files"
[dev (root-commit) 381aff8] Updated task definition files and added AppSpec files
 4 files changed, 97 insertions(+)
  create mode 100644 appspec-customer.yaml
  create mode 100644 appspec-employee.yaml
  create mode 100644 taskdef-customer.json
  create mode 100644 taskdef-employee.json
```

```
vclabs:~/environment/deployment (dev) $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
vclabs:~/environment/deployment (dev) $ git push -u origin dev
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.07 KiB | 1.07 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
 * [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
```



Phase 6: Creating target groups and an Application Load Balancer

1. Create the first target group for the customer microservice.
2. Create a second target group for the customer microservice. Use the same settings as the first target group except use customer-tg-two as the target group name.
3. Create a target group for the employee microservice.
4. Create a second target group for the employee microservice.

The screenshot shows the AWS EC2 Target Groups page. The left sidebar includes links for EC2 Dashboard, Global View, Events, Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity, and Reservations. The main content area displays a table titled 'Target groups (4) Info' with columns for Name, ARN, Port, Protocol, Target type, Load balancer, and VPC ID. The table lists four entries:

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
employee-tg-two	arn:aws:elasticloadbalancing:us-east-1:08af8f41959658745:targetgroup/employee-tg-two/sg-0c23632c7cdd680fa	8080	HTTP	IP	None associated	vpc-08af8f41959658745
employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:08af8f41959658745:targetgroup/employee-tg-one/sg-0c23632c7cdd680fa	8080	HTTP	IP	None associated	vpc-08af8f41959658745
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:08af8f41959658745:targetgroup/customer-tg-two/sg-0c23632c7cdd680fa	8080	HTTP	IP	None associated	vpc-08af8f41959658745
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:08af8f41959658745:targetgroup/customer-tg-one/sg-0c23632c7cdd680fa	8080	HTTP	IP	None associated	vpc-08af8f41959658745

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

1. Create a new EC2 security group:

The screenshot shows the AWS EC2 Security Groups page. The left sidebar includes links for EC2 Dashboard, Global View, Events, Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity, and Reservations. The main content area shows a success message: 'Security group (sg-0c23632c7cdd680fa | microservices-sg) was created successfully'. Below it, the 'Details' section for the security group 'sg-0c23632c7cdd680fa - microservices-sg' is displayed. The 'Inbound rules' tab is selected, showing two entries:

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0601b64104ed89...	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-080eedc3defb0e0e0	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

2. Create an Application Load Balancer:

1. Create a new EC2 security group named microservices-sg to use in LabVPC. Add inbound rules that allow TCP traffic from any IPv4 address on ports 80 and 8080.
2. In the Amazon EC2 console, create an Application Load Balancer named microservicesLB.
3. Add a second rule for the HTTP:80 listener.
4. Add a second rule for the HTTP:8080 listener.

Inbound security group rules successfully modified on security group sg-0c23632c7cdd680fa | microservices-sg

Details

Security group name	sg-0c23632c7cdd680fa	Description	vpc-08af8f41959658743
Owner	757817367863	Inbound rules count	2 Permission entries
		Outbound rules count	1 Permission entry

Inbound rules (2)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-02465941bbc6ecb...	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-059c32069b7a4da4c	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

microservicesLB

Details

Load balancer type	Status	VPC	IP address type
Application	Active	vpc-08af8f41959658743	IPv4
Scheme	Hosted zone	Availability Zones	Date created
Internet-facing	Z35SX0DTRQ7X7K	subnet-07a2dd2c5b21476e1 us-east-1b (use1-az2)	April 27, 2024, 22:28 (UTC-04:00)
		subnet-04035a31fc28c8f64 us-east-1a (use1-az1)	
Load balancer ARN	DNS name		
arn:aws:elasticloadbalancing:us-east-1:757817367863:loadbalancer/app/microservicesLB/07ea8024847e322c	Info	microservicesLB-1978094058.us-east-1.elb.amazonaws.com (A Record)	

Listeners and rules (2) Info

Protocol/Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	mTLS
HTTP:80	Forward to target group • customer-tg-two 1 (100%) • Group-level stickiness: Off	2 rules	ARN	Not applicable	Not applicable	Not applicable
HTTP:8080	Forward to target group • customer-tg-one 1 (100%) • Group-level stickiness: Off	2 rules	ARN	Not applicable	Not applicable	Not applicable

The screenshot shows the AWS Cloud9 interface with the following details:

- File Path:** customer/create-customer-microservice-tg-two.json
- Content:**

```
1  {
2      "count": 1
3  }
```
- File Type:** JSON
- File Size:** 1.1 KB
- Last Modified:** 10 minutes ago

The screenshot shows the AWS Cloud9 interface with the following details:

- File Path:** customer/create-customer-microservice-tg-one.json
- Content:**

```
1  {
2      "count": 1
3  }
```
- File Type:** JSON
- File Size:** 1.1 KB
- Last Modified:** 10 minutes ago

Phase 7: Creating two Amazon ECS services

1. In AWS Cloud9, create a new file named `create-customer-microservice-tg-two.json` in the deployment directory.
2. Pasted the provided JSON code into the file
3. Edited the `create-customer-microservice-tg-two.json` file
4. created the Amazon ECS service for the customer microservice

```

{
  "taskDefinition": "customer-microservice:3",
  "cluster": "microservices-serverlesscluster",
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/customer-tg-two/5fc764482bf2df8f",
      "containerName": "customer",
      "containerPort": 8080
    }
  ],
  "desiredCount": 1,
  "launchType": "CLUSTER",
  "schedulingStrategy": "REPLICAS",
  "deploymentController": {
    "type": "CODE_DEPLOY"
  },
  "networkConfiguration": [
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-04833a31fc28cf64",
        "subnet-07a2d2c5b2476e1"
      ],
      "securityGroups": [
        "sg-0c23632c7dd580fa"
      ],
      "assignPublicIp": "ENABLED"
    }
  ]
}

```

```

aws -v ip-10-10-10-222.exe
{
  "service": {
    "serviceArn": "arn:aws:ecs:us-east-1:757817367863:service/microservices-serverlesscluster/customer-microservice",
    "serviceName": "customer-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      {
        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/customer-tg-two/5fc764482bf2df8f",
        "containerName": "customer",
        "containerPort": 8080
      }
    ],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 1,
    "runningCount": 0,
    "pendingCount": 0,
    "launchType": "FARGATE",
    "platformVersion": "1.4.0",
    "platformFamily": "Linux",
    "voclabs:/environment/deployment (dev) $ "
}

```

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

1. Created an Amazon ECS service for the employee microservice.
2. Ran the appropriate AWS CLI command to create the service in Amazon ECS.

```

{
  "taskDefinition": "employee-microservice:1",
  "cluster": "microservices-serverlesscluster",
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/employee-tg-two/59128ca43a8e789",
      "containerName": "employee",
      "containerPort": 8080
    }
  ],
  "desiredCount": 1,
  "launchType": "FARGATE",
  "schedulingStrategy": "REPLICAS",
  "deploymentController": {
    "type": "CODE_DEPLOY"
  },
  "networkConfiguration": [
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-04833a31fc28cf64",
        "subnet-07a2d2c5b2476e1"
      ],
      "securityGroups": [
        "sg-0c23632c7dd580fa"
      ],
      "assignPublicIp": "ENABLED"
    }
  ]
}

```

```

aws -v ip-10-10-10-225.exe
{
  "service": {
    "serviceArn": "arn:aws:ecs:us-east-1:757817367863:service/microservices-serverlesscluster/employee-microservice",
    "serviceName": "employee-microservice",
    "clusterArn": "arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster",
    "loadBalancers": [
      {
        "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/employee-tg-two/59128ca43a8e789",
        "containerName": "employee",
        "containerPort": 8080
      }
    ],
    "serviceRegistries": [],
    "status": "ACTIVE",
    "desiredCount": 1,
    "runningCount": 0,
    "pendingCount": 0,
    "launchType": "FARGATE",
    "platformVersion": "1.4.0",
    "platformFamily": "Linux",
    "voclabs:/environment/deployment (dev) $ "
}

```

The screenshot shows the AWS ECS Cluster Overview page. The cluster 'microservices-serverlesscluster' is active. It displays two services: 'customer-microservice' and 'employee-microservice', both of which are active and running tasks. The customer service has 0/1 tasks running, and the employee service has 0/1 tasks running.

Phase 8: Configuring CodeDeploy and CodePipeline

Task 8.1: Create a CodeDeploy application and deployment groups

The screenshot shows the AWS CodeDeploy Application Details page for 'microservices-customer'. It displays deployment group details, environment configuration, and load balancing settings. Deployment group details include name 'microservices-customer', service role ARN, and compute platform 'Amazon ECS'. Environment configuration includes ECS cluster name 'microservices-serverlesscluster' and service name 'customer-microservice'. Load balancing includes target groups 'customer-tg-one' and 'customer-tg-two' with their respective ARNs.

Deployment group details

- Deployment group name: microservices-employee
- Application name: microservices
- Compute platform: Amazon ECS
- Service role ARN: arn:aws:iam::757817367863:role/DeployRole
- Deployment configuration: CodeDeployDefault.ECSAllAtOnce

Environment configuration

- ECS cluster name: microservices-serverlesscluster
- ECS service name: employee-microservice

Load Balancing

- Target group 1 name: employee-tg-two
- Target group 2 name: employee-tg-one
- Production listener ARN: arn:aws:elasticloadbalancing:us-east-1:757817367863:listener/app/microservicesLB/07ea8024847e322c/d4f7a925896de734
- Test listener ARN: arn:aws:elasticloadbalancing:us-east-1:757817367863:listener/app/microservicesLB/07ea8024847e322c/db4d88b7b138e92a

Task 8.2: Create a pipeline for the *customer* microservice

Success
Pipeline was saved successfully.

update-customer-microservice

Pipeline type: V2 Execution mode: QUEUED

Source Succeeded
Pipeline execution ID: 6aebd72c-9277-4691-80ba-59ddaa58061

Source	Image
AWS CodeCommit	Amazon LCR
Succeeded - 11 minutes ago	Didn't Run
381aff86	No executions yet

Deploy Failed
Pipeline execution ID: 6aebd72c-9277-4691-80ba-59ddaa58061

Deploy
Amazon ECS (Blue/Green)
Failed - 11 minutes ago
View details

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

1. Launched a Deployment
2. Observe Progress in CodeDeploy:
3. Test the Customer Microservice
4. Observe Running Tasks in Amazon ECS

The screenshot shows the AWS CodePipeline console with the pipeline named "update-customer-microservice". The pipeline type is V2 and the execution mode is QUEUED. The pipeline has two stages: Source and Deploy. The Source stage is completed successfully, and the Deploy stage is also completed successfully. The URL for the screenshot is https://us-east-1.console.aws.amazon.com/console/home?region=us-east-1.

The screenshot shows the AWS CodeDeploy console with the deployment configuration "CodeDeployDefault.ECSAllAtOnce" and the deployment group "microservices-customer". The revision details show an application revision registered by Deployment ID: d-YBLUWOT5. The task set activity table lists two task sets: "ecs-svc/7573483859554758279" (Original, ACTIVE, 0 traffic, 1 desired count, 0 running count, 0 pending count) and "ecs-svc/882827722128573000" (Replacement, PRIMARY, 100% traffic, 1 desired count, 1 running count, 0 pending count). The deployment lifecycle events table shows various events like BeforeInstall, Install, AfterInstall, AllowTestTraffic, etc., all succeeded with start and end times around April 28, 2024, at UTC-4:00.

The screenshot shows a web browser window with the URL <https://microserviceslb-1978094058.us-east-1.elb.amazonaws.com>. The page title is "Coffee suppliers". The main content area features a photograph of two cups of coffee on a espresso machine. Below the image, the text "Welcome" is displayed, followed by the sub-instruction "Use this app to keep track of your coffee suppliers". A blue link "List of suppliers" is present. In the top right corner, there are links for "Customer home", "Suppliers list", and "Administrator link". The browser's address bar and various icons are visible at the top.

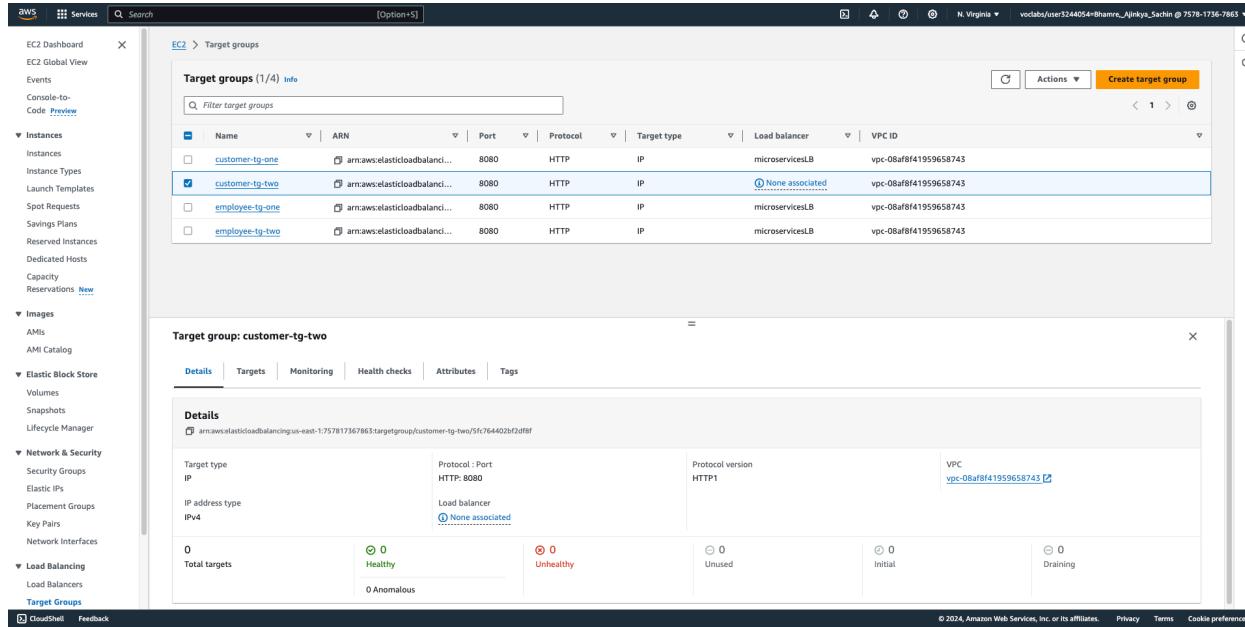
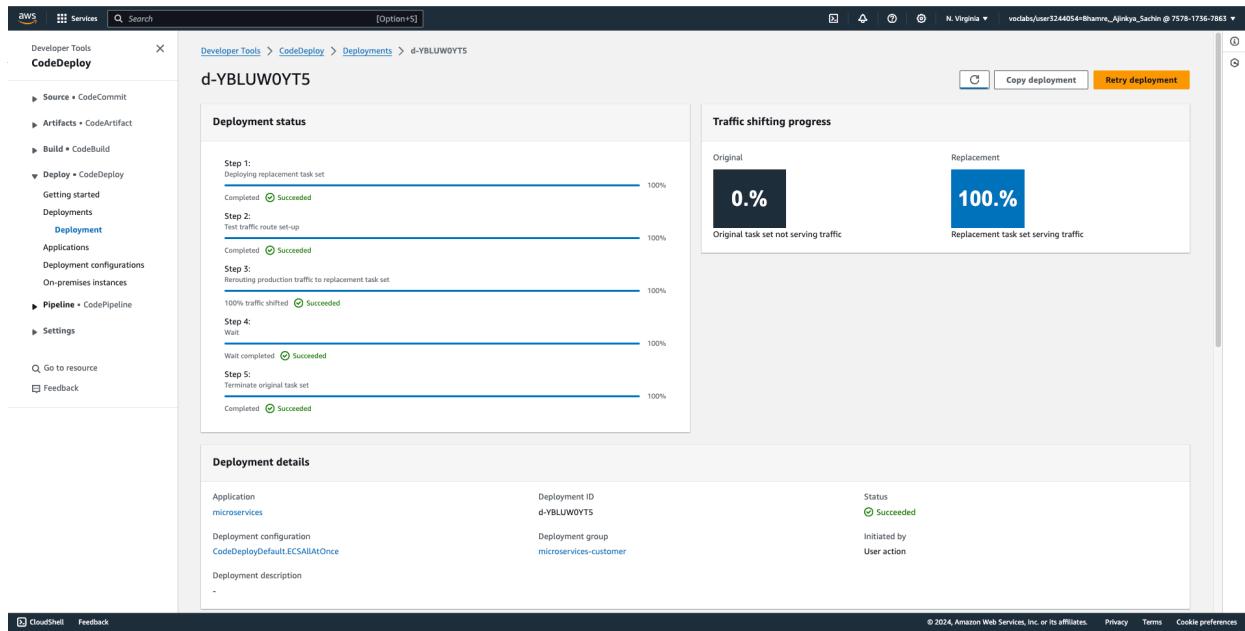
5. Return to CodeDeploy Page

The screenshot shows the same web application as above, but the main content area is titled "All suppliers". It displays a table with one row of data:

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334

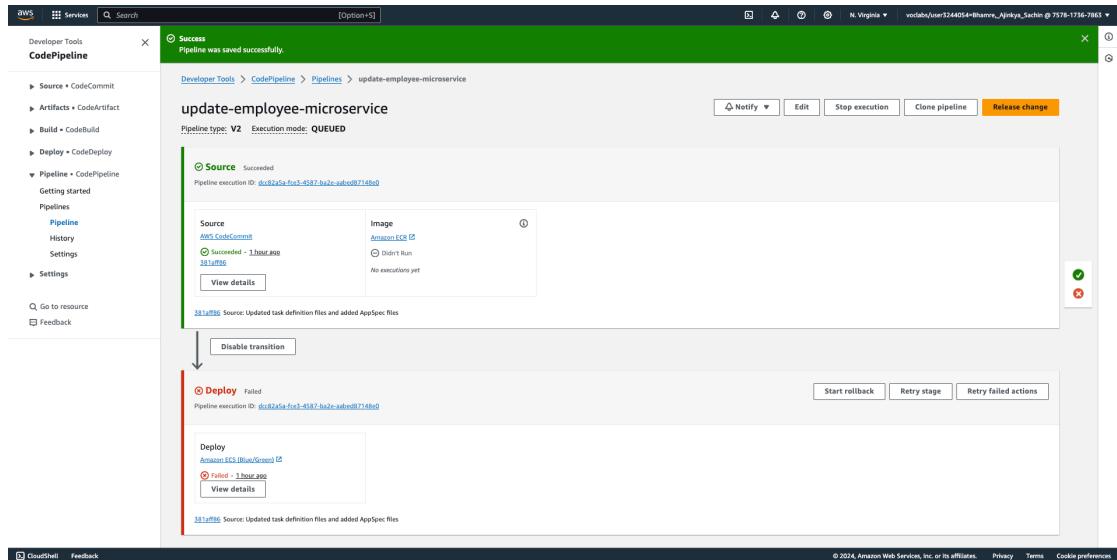
The screenshot shows the AWS Elastic Container Service (ECS) Cluster overview for the cluster "microservices-serverlesscluster". The "Services" tab is selected, showing two services: "customer-microservice" and "employee-microservice". Both services are active and have 1 task running each. The "Tasks" tab shows 1 pending task for the "customer-microservice". The "CloudWatch monitoring" section indicates "Default" monitoring is applied. The "ARN" of the cluster is listed as `arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster`.

The screenshot shows the AWS Elastic Container Service (ECS) Task overview for the cluster "microservices-serverlesscluster". The "Tasks" tab is selected, showing one task named "6576a01b..." which is currently running. The "CloudWatch monitoring" section indicates "Default" monitoring is applied. The "ARN" of the cluster is listed as `arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster`.



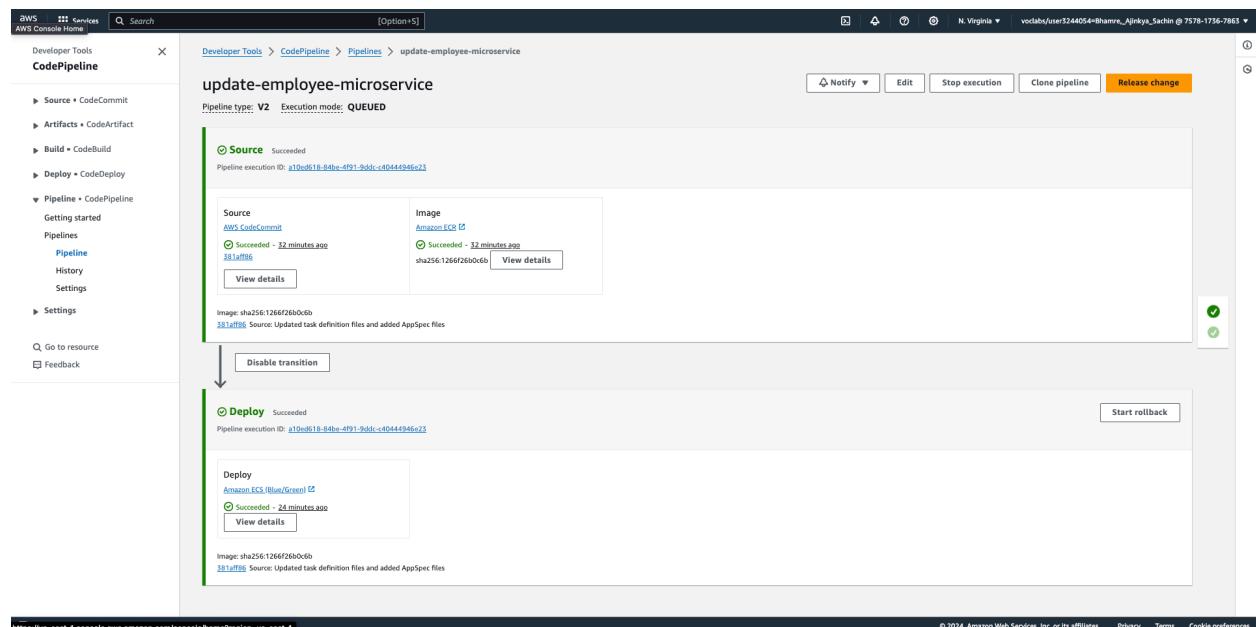
6. Observe Load Balancer and Target Group Settings

Task 8.4: Create a pipeline for the *employee* microservice



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

1. Launched Deployment:
2. Monitored CodeDeploy Progress:
3. Loaded Employee Microservice:
4. Tested Employee Microservice:
5. Observed Running Tasks:
6. Confirmed Deployment Success:



The screenshot shows a web browser window with the URL <https://microserviceslib-1978094058.us-east-1.elb.amazonaws.com/admin/suppliers>. The page title is "Manage coffee suppliers". On the left, there's a thumbnail image of two cups of coffee. On the right, there are three navigation links: "Administrator home", "Suppliers list", and "Customer home". Below the title, it says "All suppliers". A table lists one supplier entry:

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334

A green button at the bottom left says "Add a new supplier!".

The screenshot shows a web browser window with the URL <https://microserviceslib-1978094058.us-east-1.elb.amazonaws.com/admin/supplier-add>. The page title is "Manage coffee suppliers". On the left, there's a thumbnail image of two cups of coffee. On the right, there are three navigation links: "Administrator home", "Suppliers list", and "Customer home". A blue banner at the top says "All fields are required". The form has fields for Name, Address, City, State, Email, and Phone, each with a placeholder value. A blue "Submit" button is at the bottom.

The screenshot shows a web browser window with the URL <https://microserviceslib-1978094058.us-east-1.elb.amazonaws.com/admin/suppliers>. The page title is "Manage coffee suppliers". On the left, there's a thumbnail image of two cups of coffee. On the right, there are three navigation links: "Administrator home", "Suppliers list", and "Customer home". Below the title, it says "All suppliers". The table now shows two supplier entries:

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334
supplier-2	supplier-2-address	supplier-2-city	supplier-2-state	supplier2@gmail.com	2012567334

A green button at the bottom left says "Add a new supplier!".

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-1:757817367863:cluster/microservices-serverlesscluster	Status	Active
		CloudWatch monitoring	Default
		Registered container instances	-
Tasks			
Draining	Active	Pending	Running
-	2	-	2

Services (2) Info

Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition	Launch...
customer-microservice	arn:aws:sec...	Active	REPLICA	1/1 tasks running	-	customer-microservice:4	FARGATE
employee-microservice	arn:aws:sec...	Active	REPLICA	1/1 tasks running	-	employee-microservice:2	FARGATE

Services | Tasks | Infrastructure | Metrics | Scheduled tasks | Tags

Create

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-1:757817367863:cluster/microservices-serverlesscluster	Status	Active
		CloudWatch monitoring	Default
		Registered container instances	-
Tasks			
Draining	Active	Pending	Running
-	2	-	2

Tasks (2)

Task	Last status	Desired status	Task definition	Health status	Started at	Container instant...	Launch type	Platform version	CPU	Memory
6576a01b...	Running	Running	customer-micr...	Unknown	3 hours ago	-	FARGATE	1.4.0	.5 vCPU	1 GB
8794dbde...	Running	Running	employee-micr...	Unknown	42 minutes ago	-	FARGATE	1.4.0	.5 vCPU	1 GB

Services | Tasks | Infrastructure | Metrics | Scheduled tasks | Tags

Stop | **Run new task**

Developer Tools

CodeDeploy

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

Q Go to resource

Feedback

d-N07F4AOUS

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	0.%	Original task set not serving traffic
Replacement	100.%	Replacement task set serving traffic

Copy deployment | **Retry deployment**

Deployment details

Application	Deployment ID	Status
microservices	d-N07F4AOUS	Succeeded
Deployment configuration	Deployment group	Initiated by
CodeDeployDefault.ECSAtOnce	microservices-employee	User action
Deployment description	-	

CloudShell | **Feedback**

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

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

Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/customer-tg-one/07ea024847e322c/d4f7a925896de734	8080	HTTP	IP	microservicesLB	vpc-08af8f41959658743
customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/customer-tg-two/07ea024847e322c/d4f7a925896de734	8080	HTTP	IP	None associated	vpc-08af8f41959658743
employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/employee-tg-one/07ea024847e322c/d4f7a925896de734	8080	HTTP	IP	microservicesLB	vpc-08af8f41959658743
employee-tg-two	arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/employee-tg-two/07ea024847e322c/d4f7a925896de734	8080	HTTP	IP	None associated	vpc-08af8f41959658743

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

Task 9.1: Limit access to the employee microservice

Details

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

Protocol:Port	Load balancer	Default actions
HTTP:80	microservicesLB	Forward to target group • customer-tg-one (100%) • Group-level stickiness: Off

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 73.160.209.154/32, AND • Path Pattern is /admin/*	Forward to target group • employee-tg-one (100%) • Group-level stickiness: Off	arn:aws:elasticloadbalancing:us-east-1:757817367863:listener/app/microservicesLB/07ea024847e322c/d4f7a925896de734	0 tags
Default	Last (default)	If no other rule applies	Forward to target group • customer-tg-two (100%) • Group-level stickiness: Off	arn:aws:elasticloadbalancing:us-east-1:757817367863:listener/app/microservicesLB/07ea024847e322c/d4f7a925896de734	0 tags

The screenshot shows the AWS CloudWatch Metrics interface. A metric named "EmployeeCount" is selected. The value is 100, and it is shown across all dimensions. The metric is plotted against time, with data points at 2023-09-01T00:00:00Z and 2023-09-01T01:00:00Z.

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

The screenshot shows a terminal window with a file browser on the left. The terminal is running a Docker container with the command `docker run -it dkr.ecr.us-east-1.amazonaws.com/employee:latest`. The output shows the deployment process:

```

$ docker run -it dkr.ecr.us-east-1.amazonaws.com/employee:latest
[...]
Login Succeeded
volumes:/environment/employee (dev) $ account_id=$aws sts get-caller-identity | grep Account | cut -d '-' -f4
volumes:/environment/employee (dev) $ echo $account_id
75393f7e-974a-49a2-93a2-864397873298
volumes:/environment/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
volumes:/environment/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
volumes:/environment/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
volumes:/environment/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push successfully to repository [75393f7e-974a-49a2-93a2-864397873298].dkr.ecr.us-east-1.amazonaws.com/employee
b99298c9fa22 Pushed
864397873298 Pushed
43f1113380b7 Layer already exists
d8d7f53380b7 Layer already exists
1dc7f53380b7 Layer already exists
d0cc53380b7 Layer already exists
f10593f7e4b5 Layer already exists
latest: digest: sha256:583ed084ecf73cecca1bd834637cd8ad22de278af85d851ca438ff9f20 size: 1783
volumes:/environment/employee (dev) $

```

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

update-employee-microservice

Pipeline type: V2 Execution mode: QUEUED

Source Succeeded Pipeline execution ID: 41708150-8d15-43c4-952-df7475a6f7fb

Deploy Succeeded Pipeline execution ID: 41708150-8d15-43c4-952-df7475a6f7fb

employee

Images (2)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	28 April 2024, 21:53:02 (UTC-04)	27.70	Copy URI	sha256:585edad8a1ecf33cecca10d5834637...
-	Image	27 April 2024, 19:12:09 (UTC-04)	27.70	Copy URI	sha256:1266f26b0c6bcd597705ca903ecdee...

Deployment history

Deployment ID	Status	Deployment type	Compute platform	Application	Deployment group	Revision location	Initiating event	Start time	End time
d-CN86VQ3US	Succeeded	Blue/green	Amazon ECS	microservices	microservices-emp...	806d3d9672e5c2...	User action	Apr 28, 2024 10:00...	Apr 28, 2024 10:00...
d-JBAN052US	Failed	Blue/green	Amazon ECS	microservices	microservices-emp...	77cd2dbaa357025...	User action	Apr 28, 2024 9:57...	Apr 28, 2024 9:57...
d-3NDQRGZUS	Failed	Blue/green	Amazon ECS	microservices	microservices-emp...	b5908a6921dd0f08...	User action	Apr 28, 2024 9:53...	Apr 28, 2024 9:53...
d-N07F4AOUS	Succeeded	Blue/green	Amazon ECS	microservices	microservices-emp...	64e3ecd5c01f081...	User action	Apr 28, 2024 7:31...	Apr 28, 2024 7:31...
d-YBLUW0YTS	Succeeded	Blue/green	Amazon ECS	microservices	microservices-cust...	ad7358b18b6164...	User action	Apr 28, 2024 5:11...	Apr 28, 2024 5:11...

Task 9.4: Test access to the *employee* microservice

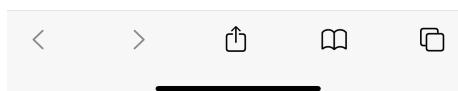
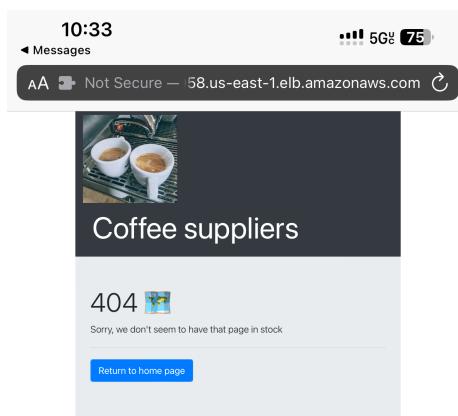
1. Tested from the Same Device: (<http://microserviceslb-1978094058.us-east-1.elb.amazonaws.com/admin/suppliers>)

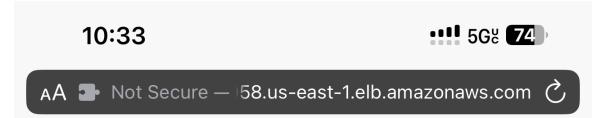
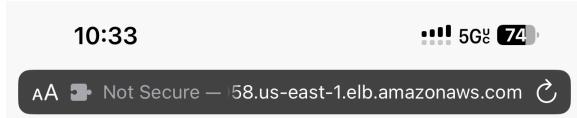
Administrator home
Suppliers list
Customer home

Name	Address	City	State	Email	Phone
supplier-1	supplier-address	supplier-city	supplier-city	supplier@gmail.com	+12012679334
supplier-2	supplier-2-address	supplier-2-city	supplier-2-state	supplier2@gmail.com	2012567334

Add a new supplier

2. Verified that the banner with the page title is now a light color due to the change made to the nav.html file.
3. Tested from a Different Device:





Used a different device (e.g., smartphone) / connected from a different network.

Visited the same URLs (`http://<alb-endpoint>/admin/suppliers` and `http://<alb-endpoint>:8080/admin/suppliers`) from this device.

Verified that I receive a 404 error on any page that loads.

The page should display "Coffee suppliers" instead of "Manage coffee suppliers."

This indicates that access to the employee microservice is restricted based on the IP address.

Task 9.5: Scale the *customer* microservice

- Updated the Customer Service in Amazon ECS: aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3

```
aws — aws ecs update-service --cluster microservices-serverlesscluster --service — aws — less + aws ecs update-service --...  
{  
    "service": {  
        "serviceArn": "arn:aws:ecs:us-east-1:757817367863:service/microservices-serverlesscluster/customer-microservice",  
        "serviceName": "customer-microservice",  
        "clusterArn": "arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster",  
        "loadBalancers": [  
            {  
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:757817367863:targetgroup/customer-tg-one/22e8eb4270696a5b",  
                "containerName": "customer",  
                "containerPort": 8080  
            }  
        ],  
        "serviceRegistries": [],  
        "status": "ACTIVE",  
        "desiredCount": 3,  
        "runningCount": 1,  
        "pendingCount": 0,  
        "launchType": "FARGATE",  
        "platformVersion": "1.4.0",  
        "taskDefinition": "arn:aws:ecs:us-east-1:757817367863:task-definition/cu...  
:  
:
```

This command increases the desired count of containers for the customer-microservice service to 3.

- View the Customer Service in Amazon ECS:

The screenshot shows the AWS ECS console interface. On the left, there's a navigation sidebar with links for Services, Search, Tell us what you think, Amazon Elastic Container Service, Clusters, Namespaces, Task definitions, Account settings, Install AWS Copilot, Amazon ECR, Repositories, AWS Batch, Documentation, Discover products, and Subscriptions. The main area displays the 'microservices-serverlesscluster' cluster overview. It includes sections for Cluster overview (ARN: arn:aws:ecs:us-east-1:757817367863:cluster/microservices-serverlesscluster, Status: Active, CloudWatch monitoring: Default, Registered container instances: 4), Services (Draining: -, Active: 2, Pending: -, Running: 4), and Tasks (Pending: -, Running: 4). Below this, there's a 'Services (2) Info' table with columns for Service name, ARN, Status, Service type, Deployments and tasks, Last deployment, Task definition, and Launch type. Two services are listed: 'customer-microservice' (Status: Active, Service type: REPLICA, Deployments and tasks: 3/3 tasks running, Last deployment: -, Task definition: customer-microservice:4, Launch type: FARGATE) and 'employee-microservice' (Status: Active, Service type: REPLICA, Deployments and tasks: 1/1 tasks running, Last deployment: -, Task definition: employee-microservice:5, Launch type: FARGATE).

Service	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition	Launch type
customer-microservice	arn:aws:ecs:us-east-1:757817367863:task-definition/customer-microservice:4	Active	REPLICA	3/3 tasks running	-	customer-microservice:4	FARGATE
employee-microservice	arn:aws:ecs:us-east-1:757817367863:task-definition/employee-microservice:5	Active	REPLICA	1/1 tasks running	-	employee-microservice:5	FARGATE

Part 2: Analysis

In **Phase 1**, I meticulously planned the solution's design and estimated its cost using the AWS Pricing Calculator. Creating an architectural diagram helped visualize the solution's components and their interactions, laying a solid foundation for implementation. The cost estimate breakdown provided insights into the financial implications, aiding in budgeting for the project's duration.

Phase 2 involved analyzing the existing monolithic application's infrastructure and functionality. By verifying its availability and interacting with its features, I gained a comprehensive understanding of its behavior and structure. Exploring the application's code, running processes, and database connections deepened my insights, paving the way for a smooth transition to a microservices architecture.

In **Phase 3**, I established a development environment using AWS Cloud9 and integrated the application code into AWS CodeCommit. Setting up the Cloud9 instance with the required configurations and copying the source code facilitated seamless development. Initializing Git repositories, creating branches, and pushing changes to CodeCommit ensured version control and collaboration efficiency.

Transitioning to a microservices architecture in **Phase 4** involved several crucial steps. Adjusting security group configurations, modifying microservices functionalities, creating Docker images, and testing containerized applications laid the groundwork for scalability and deployment automation. Version control with CodeCommit ensured code integrity and traceability, fostering a collaborative development environment.

Phase 5 focused on preparing microservices for deployment on Amazon ECS and integrating them into the CI/CD pipeline. Establishing ECR repositories, creating ECS clusters, defining task definitions, and authoring AppSpec files ensured smooth deployment orchestration. Updates to deployment files and code check-ins streamlined the integration process, setting the stage for continuous delivery.

In **Phase 6**, I configured an Application Load Balancer to serve as the HTTPS entry point for the application. Setting up target groups and routing rules enabled efficient traffic distribution, supporting scalability and reliability for the microservices.

Phase 7 involved creating ECS services for the microservices, defining task definitions, and utilizing AWS CLI commands for service creation. These services ensure the seamless deployment and management of microservices within the ECS cluster, enhancing scalability and reliability.

Phase 8 focused on configuring CodeDeploy and CodePipeline for microservice deployment. Creating CodeDeploy applications and deployment groups, setting up pipelines, and employing blue/green deployment strategies ensured smooth updates. Testing pipelines and observing CodeDeploy's modifications to load balancer rules confirmed successful deployments.

In Phase 9, I adjusted microservice code, observed CI/CD pipelines in action, and tested access restrictions. Limiting access to the employee microservice, modifying UI, and triggering pipeline runs through Docker image updates demonstrated the agility and automation of the deployment process. Scaling the customer microservice showcased independent microservice management, enhancing scalability and efficiency.