

Verify that the supplier no longer appears on the suppliers page.

All suppliers

Name	Address	City	State	Email	Phone
Nancy_Supplier1	1 Castle Point Terrace	Hoboken	New Jersey	ngupta19@stevens.edu	5512098571

Add a new supplier

Troubleshooting tip: If any of the functionality mentioned above isn't working as intended, follow these steps: (1) Stop and delete the running container, (2) modify the microservice source code as appropriate, (3) create an updated Docker image from the source code, (4) launch a new test container, and (5) verify whether the functionality is now available. For a list of commands to run to accomplish these steps, see [Updating a test container running on Cloud9](#) in the appendix of this file.

5. To observe details about both running test containers, run the following command:

`docker ps`

```
Welcome
Dockerfile
FROM node:11-alpine
bash -cp-10-16-10-131.e x Immediate
--> Running in 5ff3cbef073b
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.753s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  --> use "npm audit fix" to fix them, or "npm audit" for details
Removing intermediate container 5ff3cbef073b
--> db845b6f095c
Step 6/7 : EXPOSE 8081
--> Running in 676b001c5d98
Removing intermediate container 676b001c5d98
--> d66f5ab04f61
Step 7/7 : CMD [ "npm", "run", "start" ]
--> Running in 18fe26dc061
Removing intermediate container 18fe26dc061
--> efd1a975aae
Successfully built efd1a975aae
Successfully tagged employee:latest
vclabs:-/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e DB_ENDPOINT=<database_endpoint> employee
bash: database endpoint: No such file or directory
vclabs:-/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
122948539ee0c6e002a2af34aba9aa81f5cd8083272b61ff00acd986944f5
vclabs:-/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer
122948539ee0c6e002a2af34aba9aa81f5cd8083272b61ff00acd986944f5
vclabs:-/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
vclabs:-/environment/microservices/employee (dev) $ echo $dbEndpoint
vclabs:-/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
docker: Error response from daemon: Conflict. The container name "employee_1" is already in use by container "122948539ee0c6e002a2af34aba9aa81f5cd8083272b61ff00acd986944f5".
You have to remove (or rename) that container to be able to reuse that name.
See 'docker run -h'.
vclabs:-/environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
122948539ee0 employee "docker-entrypoint.s..." 4 minutes ago Up 4 minutes 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp employee_1
9ea3df13a4a7 customer "docker-entrypoint.s..." 34 minutes ago Up 34 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
vclabs:-/environment/microservices/employee (dev) $
```

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

When you tested the employee microservice on the AWS Cloud9 instance, you ran it on port 8081. However, when you deploy it to Amazon ECS, you will want it to run on port 8080. To adjust this, you need to modify two files.

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

```

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

```

```

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
595c2ed5cc31 employee "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:8081->8081/tcp, ::8081->8081/tcp employee_1
f38d87396f8a customer "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:8080->8080/tcp, ::8080->8080/tcp customer_1
voclabs:-/environment/microservices/employees (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
595c2ed5cc31 employee "docker-entrypoint.s..." 12 minutes ago Up 12 minutes 0.0.0.0:8081->8081/tcp, ::8081->8081/tcp employee_1
f38d87396f8a customer "docker-entrypoint.s..." 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, ::8080->8080/tcp customer_1
voclabs:-/environment/microservices/employees (dev) $ |

```

```

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

```

```

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
595c2ed5cc31 employee "docker-entrypoint.s..." About a minute ago Up About a minute 0.0.0.0:8081->8081/tcp, ::8081->8081/tcp employee_1
f38d87396f8a customer "docker-entrypoint.s..." About an hour ago Up About an hour 0.0.0.0:8080->8080/tcp, ::8080->8080/tcp customer_1
voclabs:-/environment/microservices/employees (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
595c2ed5cc31 employee "docker-entrypoint.s..." 12 minutes ago Up 12 minutes 0.0.0.0:8081->8081/tcp, ::8081->8081/tcp employee_1
f38d87396f8a customer "docker-entrypoint.s..." 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, ::8080->8080/tcp customer_1
voclabs:-/environment/microservices/employees (dev) $ |

```

2. Rebuild the Docker image for the employee microservice.

- To stop and delete the existing container (assumes that the container name is `employee_1`), run the following command:

`docker rm -f employee_1`

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree displays a project structure with files like Dockerfile, index.js, package-lock.json, package.json, labsuser.pem, and README.md. The Dockerfile content is:

```

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

```

The terminal window shows the command `docker ps` output:

```

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
595c2ed5cc31 employee_1 "docker-entrypoint.s..." 12 minutes ago Up 12 minutes 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp employee_1
f38d87396f8a customer_1 "docker-entrypoint.s..." 2 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1

```

- Ensure that your terminal is in the employee directory.
- Use the docker build command to build a new image from the latest source files that you edited. Use the employee tag.

Use the following command:

`docker build --tag customer .`

The screenshot shows the AWS Cloud9 IDE interface. The terminal window shows the command `docker build --tag employee .` being run, followed by the build process output:

```

Step 1/7 : FROM node:11-alpine
--> f18da2f58cd3
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> 39f01094b424
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> 40204abdc28d
Step 4/7 : COPY .
--> 25667edcc54
Step 5/7 : RUN npm install
--> Running in d66279bbdc5d
npm WARN coffee_apm@0.0 No repository field.

audited 78 packages in 0.832s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run "npm audit fix" to fix them, or "npm audit" for details
Removing intermediate container d66279bbdc5d
--> 306559bd49
Step 6/7 : EXPOSE 8080
--> Running in 1153abde4ac
Removing intermediate container 1153abde4ac
--> 8501e59ab888
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in b941d1ae9b3
Removing intermediate container b941d1ae9b3
--> bc4d739555d
Successfully built bc4d739555d
Successfully tagged employee:latest

```

Tip: If you build an image with the name of an existing image, the existing image will be overwritten.

Note: You don't need to run a new test container, so you don't need to run docker run.

Task 4.7: Check code into CodeCommit

In this task, you will commit and push the changes that you made to the employee microservice to CodeCommit.

1. Review the updates that you made to the source code. To accomplish this:

- Choose the source control icon in the AWS Cloud9 IDE.

```

bash -lp 10-16-10-131 o x Immediate
Successfully tagged employee:latest
voclab:~/environment/microservices/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.014MB
Step 1/7 : FROM node:11-alpine
--> f18da2f2f58c3d
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> 39f0f094bd424
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> d920bd4c28d
Step 4/7 : COPY . .
--> Using cache
--> 256676eccc54
Step 5/7 : RUN npm install
--> Using cache
--> 30a405980d48
Step 6/7 : EXPOSE 8080
--> Using cache
--> 8501fe9b988
Step 7/7 : CMD ["npm", "run", "start"]
--> Using cache
--> bc4c8739555d
Successfully built bc4c8739555d
Successfully tagged employee:latest
voclab:~/environment/microservices/employee (dev) $ git status
On branch dev
Your branch is up to date with 'origin/dev'.

```

Notice the changes list, which indicates which files were changed since you last checked files in to the remote Git repository (CodeCommit).

File	Type
Dockerfile	U
index.js	M
supplier.controller.js	M
header.html	M
home.html	M
nav.html	M
supplier-add.html	M
supplier-list-all.html	M
supplier-update.html	M

- Choose one of the files that was modified, such as `index.js`, to compare the version from the last Git commit to the latest version. Changes are highlighted.

This demonstrates a benefit of using a source control system and a Git-compatible IDE such as AWS Cloud9. You can review your code changes prior to committing.

```

1 const express = require("express");
2 const bodyParser = require("body-parser");
3 const cors = require("cors")
4 const supplierController = require("./app/controller/supplier.controller");
5 const mustacheExpress = require("mustache-express")
6 const favicon = require('serve-favicon');
7
8 // parse requests of content-type: application/json
9 app.use(bodyParser.json());
10 // parse requests of content-type: application/x-www-form-urlencoded
11 app.use(bodyParser.urlencoded({extended: true}));
12 app.use(cors());
13 app.engine("html", mustacheExpress());
14 app.set("view engine", "html")
15 app.set("views", __dirname + "/views")
16 app.use(express.static('public'));
17 app.use(favicon(__dirname + '/public/img/favicon.ico'));
18
19 // list all the suppliers
20 app.get("/", (req, res) => {
21   res.render('home', {});
22 });
23
24 app.get('/suppliers/', supplier.findAll);
25 // show the add supplier form
26 app.get('/supplier-add', (req, res) => {
27   res.render('supplier-add', {});
28 });
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
40 app.use(function (req, res, next) {
41   res.status(404).render('404', {});
42 })

```

bash - ip-10-16-10-153 e x Immediate (Javascript (br x)

2. Check your changes into CodeCommit.

Tip: You performed this same type of action in task 4.3. You can accomplish this step by using the Git source control panel, or you can use the git commit and git push commands in the terminal.

```

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
        modified:   app/controller/supplier.controller.js
        modified:   Dockerfile
        modified:   views/header.html
        modified:   views/home.html
        modified:   views/nav.html
        modified:   views/supplier-add.html
        modified:   views/supplier-list-all.html
        modified:   views/supplier-update.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Dockerfile

no changes added to commit (use "git add" and/or "git commit -a")

```

bash - ip-10-16-10-153 e x Immediate (Javascript (br x)

vectabs:/environment/microservices/employee (dev) \$ git status

On branch dev

Your branch is up to date with 'origin/dev'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: app/controller/supplier.controller.js

modified: Dockerfile

modified: views/header.html

modified: views/home.html

modified: views/nav.html

modified: views/supplier-add.html

modified: views/supplier-list-all.html

modified: views/supplier-update.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

Dockerfile

no changes added to commit (use "git add" and/or "git commit -a")

vectabs:/environment/microservices/employee (dev) \$

File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)

MicroservicesIDE - /home/k

c9

microservices

customer

employee

app

node_modules

public

css

img

js

views

Dockerfile

JS index.js

{} package-lock.json

{} package.json

labsuser.pem

README.md

bash - ip-10-16-10-153.e x Immediate (Javascript br x)

```
28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // receive the update POST
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // handle 404
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })

```

Your branch is up to date with "origin/dev".

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: app/controller/supplier-controller.js

modified: index.js

modified: views/header.html

modified: views/home.html

modified: views/nav.html

modified: views/supplier-add.html

modified: views/supplier-list-all.html

modified: views/supplier-update.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

Dockerfile

no changes added to commit (use "git add" and/or "git commit -a")

veclabs:-/environment/microservices/employee (dev) \$ git add .

veclabs:-/environment/microservices/employee (dev) \$

veclabs:-/environment/microservices/employee (dev) \$

45 46 JavaScript Spaces: 4

File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)

MicroservicesIDE - /home/k

c9

microservices

customer

employee

app

node_modules

public

css

img

js

views

Dockerfile

JS index.js

{} package-lock.json

{} package.json

labsuser.pem

README.md

bash - ip-10-16-10-153.e x Immediate (Javascript br x)

```
28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // receive the update POST
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // handle 404
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })

```

(use "git restore <file>..." to discard changes in working directory)

modified: app/controller/supplier-controller.js

modified: index.js

modified: views/header.html

modified: views/home.html

modified: views/nav.html

modified: views/supplier-add.html

modified: views/supplier-list-all.html

modified: views/supplier-update.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

Dockerfile

no changes added to commit (use "git add" and/or "git commit -a")

veclabs:-/environment/microservices/employee (dev) \$ git add .

veclabs:-/environment/microservices/employee (dev) \$

veclabs:-/environment/microservices/employee (dev) \$ git commit -m "Updated employee microservice port to 8080"

[dev 6f3df6] Updated employee microservice port to 8080

9 files changed, 28 insertions(+), 20 deletions(-)

create mode 100644 employee/Dockerfile

veclabs:-/environment/microservices/employee (dev) \$

45 46 JavaScript Spaces: 4

File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)

MicroservicesIDE - /home/k

c9

microservices

customer

employee

app

node_modules

public

css

img

js

views

Dockerfile

JS index.js

{} package-lock.json

{} package.json

labsuser.pem

README.md

git - ip-10-16-10-153.ec2 x Immediate (Javascript br x)

```
28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // receive the update POST
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // handle 404
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })

```

Untracked files:

(use "git add <file>..." to include in what will be committed)

Dockerfile

no changes added to commit (use "git add" and/or "git commit -a")

veclabs:-/environment/microservices/employee (dev) \$ git add .

veclabs:-/environment/microservices/employee (dev) \$

veclabs:-/environment/microservices/employee (dev) \$ git commit -m "Updated employee microservice port to 8080"

[dev 6f3df6] Updated employee microservice port to 8080

9 files changed, 28 insertions(+), 20 deletions(-)

create mode 100644 employee/Dockerfile

veclabs:-/environment/microservices/employee (dev) \$ git push

Enumerating objects: 100K (20/28), done.

Counting objects: 100K (20/28), done.

Delta compression using up to 2 threads

Compressing objects: 100% (15/15), done.

Writing objects: 100% (15/15), 1.63 KiB | 555.00 KiB/s, done.

Total 15 (delta 8), reused 0 (delta 0), pack-reused 0

remote: Validating objects: 100%

To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices

 cb85e60..6f3df6 dev -> dev

veclabs:-/environment/microservices/employee (dev) \$

45 46 JavaScript Spaces: 4

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories**
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

Go to resource Feedback

Developer Tools > CodeCommit > Repositories

Repositories info

Name	Description	Last modified	Clone URL	AWS KMS Key
microservices	-	Just now	HTTPS SSH HTTPS (GR)	arn:aws:kms:us-east-1:082451908674:key/cb78b9cf-6753-41aa-9e4c-eac54ea3156a

Create repository

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

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories**
- Code**
- Pull requests
- Commits
- Branches
- Git tags
- Settings
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

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

Developer Tools > CodeCommit > Repositories > microservices

microservices

Reference dev Create pull request Clone URL

microservices Info

Name
customer
employee

Add file

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

AWS Services Search [Alt+S] N. Virginia v vocabs/user\$223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories**
- Code**
- Pull requests
- Commits
- Branches
- Git tags
- Settings
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

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

Developer Tools > CodeCommit > Repositories > microservices

microservices

Reference dev Create pull request Clone URL

microservices / employee Info

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

Add file

AWS Services Search [Alt+S] N. Virginia v vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools CodeCommit

Source • CodeCommit

Getting started

Repositories

Code

Pull requests

Commits

Branches

Git tags

Settings

Approval rule templates

Artifacts • CodeArtifact

Build • CodeBuild

Deploy • CodeDeploy

Pipeline • CodePipeline

Settings

CloudShell Feedback

Developer Tools > CodeCommit > Repositories > microservices > Commits

microservices

Commits Commit visualizer Compare commits

Commits Info Reference dev

Commit ID Commit message Commit date Authored date Author Committer Actions

6f3dfe65	Updated employee microservice port to 8080	1 minute ago	1 minute ago	Nancy	Nancy	Copy ID	Browse
cb85e606	I am pushing source code into code commit as said	1 hour ago	1 hour ago	Nancy	Nancy	Copy ID	Browse
f3143dc2	two unmodified copies of the application code	7 hours ago	7 hours ago	Nancy	Nancy	Copy ID	Browse

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

AWS Services Search [Alt+S] N. Virginia v vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools CodeCommit

Source • CodeCommit

Getting started

Repositories

Code

Pull requests

Commits

Branches

Git tags

Settings

Approval rule templates

Artifacts • CodeArtifact

Build • CodeBuild

Deploy • CodeDeploy

Pipeline • CodePipeline

Settings

CloudShell Feedback

Commit 6f3dfe65cbd0b76607a63ead366417fe67bb52

Copy commit ID [Copy](#) Browse

▼ Details

Author	Authored date	Committer
Nancy ngupta19@stevens.edu	1 minute ago	Nancy ngupta19@stevens.edu
Commit date	Parent commit	
1 minute ago	cb85e60678c0ef44ac3afc14e94a7b342a7b1c14	

Commit message
Updated employee microservice port to 8080

< Page 1 of 1 > Go to file Hide comments Hide whitespace changes Unified Split

▼ employee/Dockerfile Added

Browse file contents Comment on file

```
1 + FROM node:11-alpine
2 + RUN mkdir -p /usr/src/app
3 + WORKDIR /usr/src/app
```

▼ employee/app/controller/supplier.controller.js

Browse file contents Comment on file

```
1 + FROM node:11-alpine
2 + RUN mkdir -p /usr/src/app
3 + WORKDIR /usr/src/app
4 + COPY . .
5 + RUN npm install
6 + EXPOSE 8080
7 + CMD ["npm", "run", "start"]
```

```
EOF
```

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

AWS Services Search [Alt+S] N. Virginia v vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674 ▾

Developer Tools CodeCommit

Source • CodeCommit

Getting started

Repositories

Code

Pull requests

Commits

Branches

Git tags

Settings

Approval rule templates

Artifacts • CodeArtifact

Build • CodeBuild

Deploy • CodeDeploy

Pipeline • CodePipeline

Settings

CloudShell Feedback

▼ employee/Dockerfile Added

Browse file contents Comment on file

```
1 + FROM node:11-alpine
2 + RUN mkdir -p /usr/src/app
3 + WORKDIR /usr/src/app
4 + COPY . .
5 + RUN npm install
6 + EXPOSE 8080
7 + CMD ["npm", "run", "start"]
```

EOF

▼ employee/app/controller/supplier.controller.js

Browse file contents Comment on file

```
*** *** @@ -22,7 +22,7 @@
22 22     Supplier.create(supplier, (err, data) => {
23 23         if (err)
24 24             res.render("500", {message: 'Error occurred while creating the Supplier.'});
25 25         else res.redirect("/suppliers");
26 26     });
27 27 }
28 28 }
*** *** @@ -83,7 +83,7 @@
83 83     } else {
84 84         res.render("500", {message: 'Error updating Supplier with id ${req.body.id}'});
85 85 }
```

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

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories
- Code
- Pull requests
- Commits**
- Branches
- Git tags
- Settings
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

employee/index.js

```

85   85      )
86   - } else res.redirect("/suppliers");
86   + } else res.redirect("/admin/suppliers");
87   87      }
88   88      );
89   89      }
*** 100    @@ -100,7 +100,7 @@
100  100      } else {
101  101          res.render("500", {message: `Could not delete supplier with id ${req.body.id}`});
102  102      }
103  - } else res.redirect("/suppliers");
103  + } else res.redirect("/admin/suppliers");
104  104      );
105  105      );
106  106      );
*** 107    ***

```

Browse file contents Comment on file © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories
- Code
- Pull requests
- Commits**
- Branches
- Git tags
- Settings
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

```

25   - app.get("/suppliers/", supplier.findAll);
25   + app.get("/admin/suppliers/", supplier.findAll);
26   26    // show the add supplier form
27   - app.get("/supplier-add", (req, res) => {
27   + app.get("/admin/supplier-add", (req, res) => {
28   28        res.render("supplier-add", {});
29   29    });
30   30    // receive the add supplier POST
31   - app.post("/supplier-add", supplier.create);
31   + app.post("/admin/supplier-add", supplier.create);
32   32    // show the update form
33   - app.get("/supplier-update/:id", supplier.findOne);
33   + app.get("/admin/supplier-update/:id", supplier.findOne);
34   34    // receive the update POST
35   - app.post("/supplier-update", supplier.update);
35   + app.post("/admin/supplier-update", supplier.update);
36   36    // receive the POST to delete a supplier
37   - app.post("/supplier-remove/:id", supplier.remove);
37   + app.post("/admin/supplier-remove/:id", supplier.remove);
38   38    // handle 404
39   39    app.use(function (req, res, next) {
40   40        res.status(404).render("404", {});
*** 42    @@ -42,7 +42,7 @@
42   42
43   43    // set port, listen for requests
44   44    // set port, listen for requests
45   - const app_port = process.env.APP_PORT || 80
45   + const app_port = process.env.APP_PORT || 8080
46   46    app.listen(app_port, () => {

```

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

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta_Nancy @ 0824-5190-8674

Developer Tools **CodeCommit**

- Source • CodeCommit
- Getting started
- Repositories
- Code
- Pull requests
- Commits**
- Branches
- Git tags
- Settings
- Approval rule templates
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
- Settings

employee/views/header.html

```

*** 4    @@ -4,7 +4,7 @@
4     <meta charset="UTF-8">
5     <link rel="stylesheet" href="/css/bootstrap.min.css">
6     <link rel="stylesheet" href="/css/base.css">
7   - <title>Coffee suppliers</title>
7   + <title>Manage coffee suppliers</title>
8     </head>
9     <body>
10    10

```

employee/views/home.html

```

*** 4    @@ -4,7 +4,7 @@
4     <div class="container">
5       <h1>Welcome</h1>
6       <p>Use this app to keep track of your coffee suppliers</p>
7   - <p><a href="/suppliers">List of suppliers</a></p>
7   + <p><a href="/admin/suppliers">List of suppliers</a></p>
8     </div>
9     </div>
10    10  ({>footer})

```

Browse file contents Comment on file © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

employee/views/nav.html

```

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

```

employee/views/supplier-add.html

```

*** *** @@ -8,7 +8,7 @@
8  8 |   </div>
9  9 |   {{/errors}}
10 10 | </div>
11 11 | <form action="/supplier-add" method="POST">
12 12 |   <form action="/admin/supplier-add" method="POST">

```

employee/views/supplier-list-all.html

```

13 13 |   <button type="submit" class="btn btn-primary">Submit</button>
14 14 | </form>
*** ***

```

```

*** *** @@ -24,12 +24,12 @@
24 24 |   <td>{{email}}</td>
25 25 |   <td>{{phone}}</td>
26 26 |   <td><h4><span class="badge badge-info"><a class="text-light"
27 27 |     href="/supplier-update/{{id}}"/>Edit</a></span></h4></td>
28 28 |   </tr>
29 29 |   {{/suppliers}}
30 30 | </tbody>
31 31 | </table>
32 32 | <h4><a class="badge badge-success" href="/supplier-add">Add a new supplier</a></h4>
33 33 | <h4><a class="badge badge-success" href="/admin/supplier-add">Add a new supplier</a></h4>
34 34 | </div>
35 35 | {{>footer}}

```

employee/views/supplier-update.html

```

*** *** @@ -9,7 +9,7 @@
0 0 | {{/errors}}

```

employee/views/supplier-update.html

```

*** *** @@ -9,7 +9,7 @@
9  9 |   {{/errors}}
10 10 | </div>
11 11 | {{#supplier}}
12 12 | <form action="/supplier-update" method="POST">
13 13 |   <form action="/admin/supplier-update" method="POST">
14 14 |     <input type="hidden" id="id" name="id" value="{{id}}">
15 15 |     <button type="submit" class="btn btn-primary">Submit</button>
*** *** @@ -35,7 +35,7 @@
35 35 |   </div>
36 36 |   <div class="modal-footer">
37 37 |     <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
38 38 |     <form action="/supplier-remove/{{id}}/" method="POST">
39 39 |       <form action="/admin/supplier-remove/{{id}}/" method="POST">
40 40 |         <button type="submit" class="float-right btn btn-danger">Delete this supplier</button>
41 41 |       </form>
*** ***

```

Comments on changes

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

At this point, you have successfully implemented numerous solution requirements. You split the monolithic application into two microservices that can run as Docker containers. You have also verified that the containers support the needed application actions, such as adding, editing, and deleting entries from the database. The microservices architecture still uses Amazon RDS to store the coffee supplier entries.

However, your work isn't finished. There are more solution requirements to implement. The containers are able to run on the AWS Cloud9 instance, but that isn't a scalable deployment architecture. You need the ability to scale the number of containers that run on each microservice up and down depending on need. Also, you need to have a load balancer to route traffic to the appropriate microservice. Finally, you need to be able to easily update each application microservice's codebase independently and roll those changes into production. In the remaining phases of the project, you will work to accomplish these solution requirements.

Task 5.1: Create ECR repositories and upload the Docker images

In this phase, you will upload the latest Docker images of the two microservices to separate Amazon ECR repositories.

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)
```

The screenshot shows the AWS Cloud9 IDE interface with several tabs open: 'labuser.pem', 'nav.html', 'Dockerfile', 'index.js', and a terminal window. The terminal window displays the following command and its output:

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

Below this, the terminal shows the execution of a 'git' command:

```
no changes added to commit (use "git add" and/or "git commit -a")
vec1abs:/environment/microservices/employee (dev) $ git add .
vec1abs:/environment/microservices/employee (dev) $
vec1abs:/environment/microservices/employee (dev) $ git commit -m "Updated employee microservice port to 8080"
[dev 6f3df6] Updated employee microservice port to 8080
 9 files changed, 28 insertions(+), 20 deletions(-)
create mode 100644 employee/Dockerfile
vec1abs:/environment/microservices/employee (dev) $ git push
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 2 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 1.63 KiB | 555.00 KiB/s, done.
Total 15 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices
 cb85e60..6f3df6 dev -> dev
vec1abs:/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
vec1abs:/environment/microservices/employee (dev) $
```

```
echo $account_id
```

The screenshot shows the AWS Lambda CodeWhisperer IDE interface. On the left, the Lambda function configuration pane displays the function name 'MicroservicesIDE - /home/ek' and various environment variables like 'c9', 'microservices', 'customer', 'employee', 'app', 'node_modules', and 'public'. The 'public' section contains files for CSS, images, and a Dockerfile. The code editor on the right shows the 'index.js' file with the following content:

```

28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // show the update form
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // receive the POST to delete a supplier
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })

```

The terminal at the bottom shows the command being run: `bash - ip-10-16-10-153.eks.amazonaws.com`. The output of the command shows the git push process, including committing changes to the employee microservice port to 8080, pushing the code to the repository, and logging the 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
```

A message in the command output indicates that the login succeeded.

The screenshot shows the AWS Lambda CodeWhisperer IDE interface. The Lambda function configuration pane is identical to the previous screenshot. The code editor on the right shows the same 'index.js' file content. The terminal at the bottom shows the command being run: `To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservices`. The output of the command shows the password being stored unencrypted in the Dockerfile's config.json file, followed by a warning about the security risk and a link to the documentation.

2. Create a separate private ECR repository for each microservice.

- Name the first repository customer

The screenshot shows the AWS Lambda IDE interface. In the left sidebar, there's a tree view of a project structure under 'MicroservicesIDE - /home/ek'. The 'public' folder is expanded, showing 'css', 'img', 'js', and 'views'. Inside 'views', there's a 'Dockerfile', 'JS index.js', 'package-lock.json', 'package.json', 'labsuser.pem', and 'README.md'. The main workspace has four tabs: 'labsuser.pem', 'nav.html', 'Dockerfile', and 'JS index.js'. The 'JS index.js' tab contains code for a Node.js application. A terminal window at the bottom shows the command: 'aws ecr create-repository --repository-name customer'. The output of the command is displayed, showing the creation of a new repository with details like ARN, registry ID, repository name, and creation date.

```

28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // show the update form
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // receive the POST to delete a supplier
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })
42 }

bash - ip-10-16-10-153.ec2.us-east-1.amazonaws.com
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
veclabs:/environment/microservices/employee (dev) $ aws ecr create-repository --repository-name customer
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:082451908674:repository/customer",
    "registryId": "082451908674",
    "repositoryName": "customer",
    "repositoryUri": "082451908674.dkr.ecr.us-east-1.amazonaws.com/customer",
    "createdAt": "2024-04-28T03:27:57.046000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
veclabs:/environment/microservices/employee (dev) $

```

- Name the second repository employee

This screenshot is similar to the previous one but shows the creation of a new ECR repository named 'employee'. The terminal output shows the command 'aws ecr create-repository --repository-name employee' being run, and the resulting repository details are displayed.

```

28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // show the update form
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // receive the POST to delete a supplier
37   app.post("/admin/supplier-remove/:id", supplier.remove);
38   // handle 404
39   app.use(function (req, res, next) {
40     res.status(404).render("404", {});
41   })
42 }

aws - ip-10-16-10-153.ec2.us-east-1.amazonaws.com
        "encryptionType": "AES256"
      }
    }
  }
veclabs:/environment/microservices/employee (dev) $ aws ecr create-repository --repository-name employee
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:082451908674:repository/employee",
    "registryId": "082451908674",
    "repositoryName": "employee",
    "repositoryUri": "082451908674.dkr.ecr.us-east-1.amazonaws.com/employee",
    "createdAt": "2024-04-28T03:29:09.315000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
veclabs:/environment/microservices/employee (dev) $

```

3. Set permissions on the *customer* ECR repository.

- For information about editing the existing JSON policy, see [Setting a Private Repository Statement](#) in the Amazon ECR User Guide.
- Replace the existing lines in the policy with the following:

```
{
  "Version": "2008-10-17",
```

```
  "Statement": [
```

```
    {
```

```
      "Effect": "Allow",
```

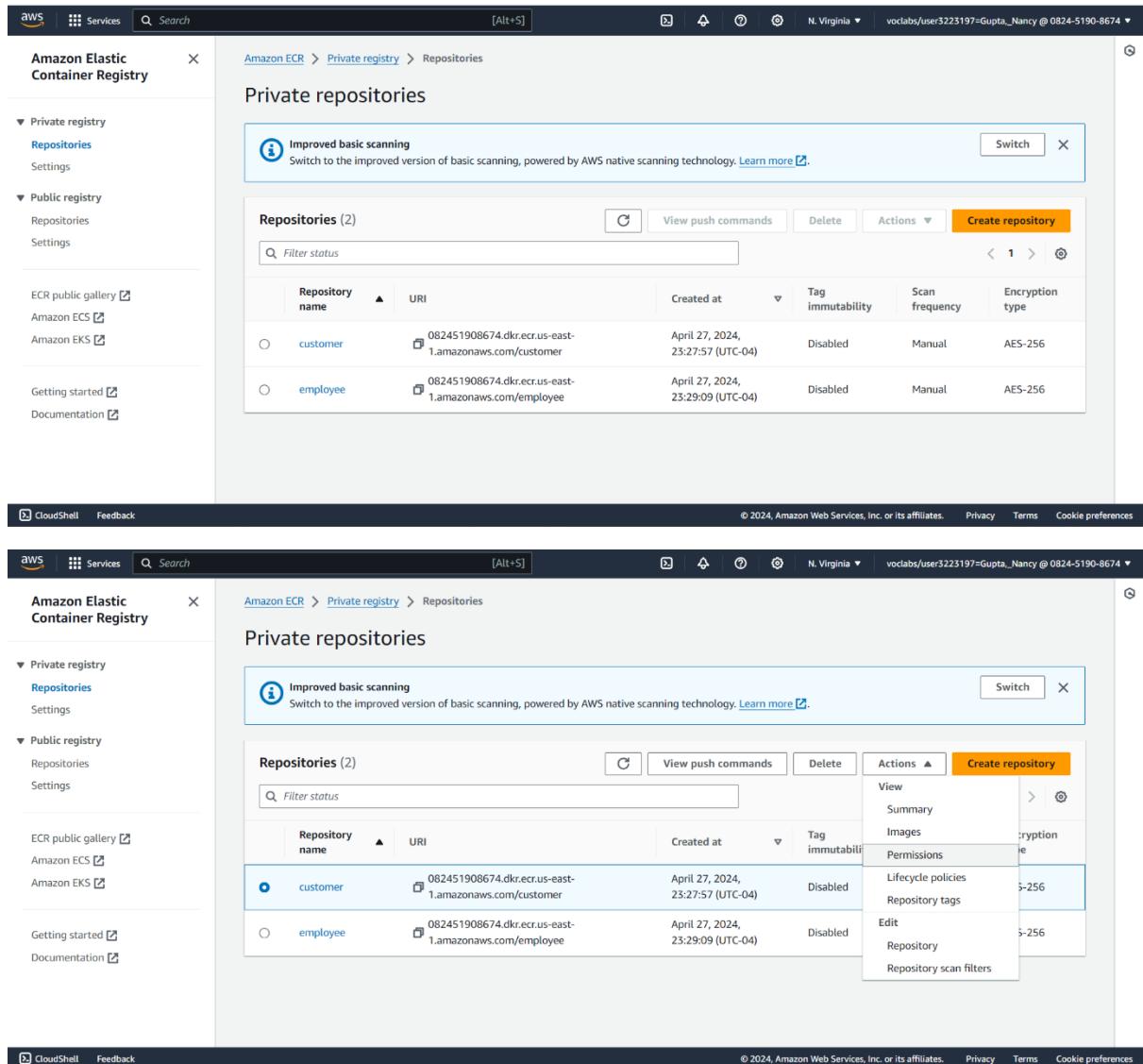
```
      "Principal": "*",
    
```

```
      "Action": "ecr:*
```

}

]

}



The screenshot shows the AWS Amazon Elastic Container Registry (ECR) interface. The left sidebar is titled "Amazon Elastic Container Registry" and includes sections for "Private registry" (Repositories, Settings), "Public registry" (Repositories, Settings), and links to ECR public gallery, Amazon ECS, and Amazon EKS. The main content area is titled "Private repositories" and shows a table of repositories. The table has columns: Repository name, URI, Created at, Tag immutability, Scan frequency, and Encryption type. There are two entries: "customer" (URI: 082451908674.dkr.ecr.us-east-1.amazonaws.com/customer, Created at: April 27, 2024, 23:27:57 (UTC-04), Tag immutability: Disabled, Scan frequency: Manual, Encryption type: AES-256) and "employee" (URI: 082451908674.dkr.ecr.us-east-1.amazonaws.com/employee, Created at: April 27, 2024, 23:29:09 (UTC-04), Tag immutability: Disabled, Scan frequency: Manual, Encryption type: AES-256). A modal window titled "Improved basic scanning" suggests switching to native scanning technology.

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	082451908674.dkr.ecr.us-east-1.amazonaws.com/customer	April 27, 2024, 23:27:57 (UTC-04)	Disabled	Manual	AES-256
employee	082451908674.dkr.ecr.us-east-1.amazonaws.com/employee	April 27, 2024, 23:29:09 (UTC-04)	Disabled	Manual	AES-256

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

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Amazon Elastic Container Registry

Permissions

Statements

No statements
You don't have any permission statements for this repository.

Edit

Private registry

- Repositories
- Summary
- Images
- Permissions**
- Lifecycle Policy
- Repository tags
- Settings

Public registry

- Repositories
- Settings

ECR public gallery

- Amazon ECS
- Amazon EKS

Getting started CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Edit permissions

Statements

You don't have any permission statements for this repository.

Reset Save

Edit JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": []  
}
```

AWS Services Search [Alt+S] N. Virginia vocabs/user3223197=Gupta,_Nancy @ 0824-5190-8674

Amazon Elastic Container Registry

Permissions

Statements

Effect All Principal ecr:*

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

Reset Save

Edit policy JSON Edit

Private registry

- Repositories
- Summary
- Images
- Permissions
- Lifecycle Policy
- Repository tags
- Settings

Public registry

- Repositories
- Settings

ECR public gallery

- Amazon ECS
- Amazon EKS

Getting started CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS ECR Permissions page for the 'customer' repository. The left sidebar includes options for Private registry (Repositories, Summary, Images, Permissions, Lifecycle Policy, Repository tags, Settings) and Public registry (Repositories, Settings). The main content area displays a policy statement:

```
Statement 1
Effect: Allow
Principal: *
Actions: ecr:*
Service principals: -
AWS Account IDs: -
```

4. Use the same approach to set the same permissions on the *employee* ECR repository.

The screenshot shows the AWS ECR Repositories page. The left sidebar includes options for Private registry (Repositories, Settings) and Public registry (Repositories, Settings). The main content area lists two repositories:

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	082451908674.dkr.ecr.us-east-1.amazonaws.com/customer	April 27, 2024, 23:27:57 (UTC-04)	Disabled	Manual	AES-256
employee	082451908674.dkr.ecr.us-east-1.amazonaws.com/employee	April 27, 2024, 23:29:09 (UTC-04)	Disabled	Manual	AES-256

The screenshot shows the AWS ECR Repositories page with a context menu open over the 'employee' repository. The menu items are:

- View
- Summary
- Images
- Permissions (selected)
- Lifecycle policies
- Repository tags
- Edit
- Repository
- Repository scan filters

The screenshot shows the AWS ECR Permissions page for a repository named 'employee'. The left sidebar includes sections for Private registry (Repositories, Summary, Images, Permissions, Lifecycle Policy, Repository tags, Settings) and Public registry (Repositories, Settings). Under ECR public gallery, links to Amazon ECS and Amazon EKS are shown. At the bottom, there are CloudShell and Feedback links.

Statements

No statements
You don't have any permission statements for this repository.
Edit

Edit policy JSON Edit

The screenshot shows the 'Edit JSON' dialog box. It contains a JSON policy template:

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

Reset Close Save

The screenshot shows the AWS ECR Permissions page after saving the JSON policy. The policy now contains one statement:

Statement 1

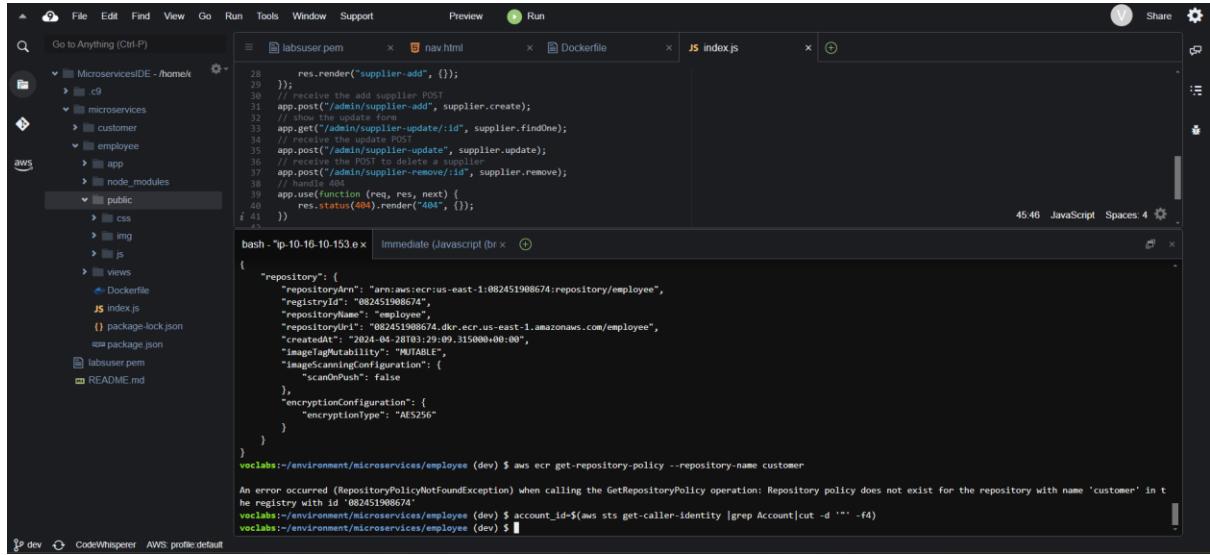
Effect	Service principals
Allow	-
Principal	AWS Account IDs
*	-
Actions	ecr:*

Edit policy JSON Edit

5. Tag the Docker images with your unique *registryId* (account ID) value to make it easier to manage and keep track of these images.

- o In the AWS Cloud9 IDE, run the following commands:

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



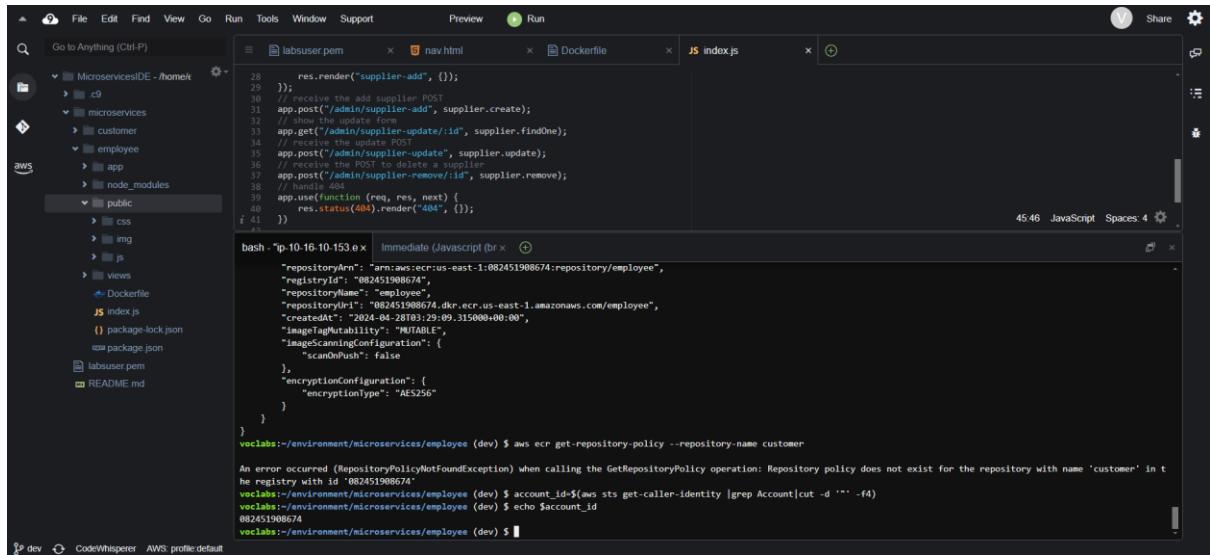
```
28     res.render("supplier-add", {});
29   // receive the add supplier POST
30   app.post("/admin/supplier-add", supplier.create);
31   // show the update form
32   app.get("/admin/supplier-update/:id", supplier.findOne);
33   // receive the update POST
34   app.post("/admin/supplier-update", supplier.update);
35   // receive the POST to delete a supplier
36   app.post("/admin/supplier-remove/:id", supplier.remove);
37   // handle 404
38   app.use(function(req, res, next) {
39     res.status(404).render("404", {});
40   })
41 }

{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:082451908674:repository/employee",
    "registryId": "082451908674",
    "repositoryName": "employee",
    "repositoryLabel": "082451908674.dkr.ecr.us-east-1.amazonaws.com/employee",
    "createdAt": "2024-04-28T03:29:09.315000+00:00",
    "imageTagMutability": "IMMUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

veclabs:/environment/microservices/employee (dev) $ aws ecr get-repository-policy --repository-name customer
An error occurred (RepositoryPolicyNotFoundException) when calling the GetRepositoryPolicy operation: Repository policy does not exist for the repository with name 'customer' in t
he registry with id '082451908674'
veclabs:/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
veclabs:/environment/microservices/employee (dev) $ 
```

```
# Verify that the account_id value is assigned to the $account_id variable
```

```
echo $account_id
```



```
28     res.render("supplier-add", {});
29   // receive the add supplier POST
30   app.post("/admin/supplier-add", supplier.create);
31   // show the update form
32   app.get("/admin/supplier-update/:id", supplier.findOne);
33   // receive the update POST
34   app.post("/admin/supplier-update", supplier.update);
35   // receive the POST to delete a supplier
36   app.post("/admin/supplier-remove/:id", supplier.remove);
37   // handle 404
38   app.use(function(req, res, next) {
39     res.status(404).render("404", {});
40   })
41 }

{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:082451908674:repository/employee",
    "registryId": "082451908674",
    "repositoryName": "employee",
    "repositoryLabel": "082451908674.dkr.ecr.us-east-1.amazonaws.com/employee",
    "createdAt": "2024-04-28T03:29:09.315000+00:00",
    "imageTagMutability": "IMMUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

veclabs:/environment/microservices/employee (dev) $ aws ecr get-repository-policy --repository-name customer
An error occurred (RepositoryPolicyNotFoundException) when calling the GetRepositoryPolicy operation: Repository policy does not exist for the repository with name 'customer' in t
he registry with id '082451908674'
veclabs:/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
veclabs:/environment/microservices/employee (dev) $ echo $account_id
082451908674
veclabs:/environment/microservices/employee (dev) $ 
```

```
# Tag the customer image
```

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

The screenshot shows the AWS Lambda Code Editor interface. The left sidebar displays the project structure:

```

└─ MicroservicesIDE - /home/ek
    └─ aws
        └─ public
            └─ employee
                └─ app
                    └─ index.js

```

The main editor area shows the contents of `index.js`:

```

28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // receive the update POST
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // receive the POST to delete a supplier
37   app.post("/admin/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 module.exports = app;

```

Below the code, the terminal window shows the command being run:

```

bash - ip-10-16-10-153.e x | Immediate (Javascript (br x)
"repositoryId": "08245198674",
"repositoryName": "employee",
"repositoryUrl": "08245198674.dkr.ecr.us-east-1.amazonaws.com/employee",
"createdAt": "2024-04-28T03:29:09.315000+00:00",
"imageTagMutability": "MUTABLE",
"imageScanningConfiguration": {
  "scanOnPush": false
},
"encryptionConfiguration": {
  "encryptionType": "AES256"
}
}

vclabs:-/environment/microservices/employee (dev) $ aws ecr get-repository-policy --repository-name customer
An error occurred (RepositoryNotFoundException) when calling the GetRepositoryPolicy operation: Repository policy does not exist for the repository with name 'customer' in t
he registry with id '08245198674'
vclabs:-/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '-' -f4)
vclabs:-/environment/microservices/employee (dev) $ echo $account_id
08245198674
vclabs:-/environment/microservices/employee (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vclabs:-/environment/microservices/employee (dev) $

```

Tag the employee image

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

Note: The commands don't return output.

The screenshot shows the AWS Lambda Code Editor interface. The left sidebar displays the project structure:

```

└─ MicroservicesIDE - /home/ek
    └─ aws
        └─ public
            └─ employee
                └─ app
                    └─ index.js

```

The main editor area shows the contents of `index.js`:

```

28     res.render("supplier-add", {});
29   });
30   // receive the add supplier POST
31   app.post("/admin/supplier-add", supplier.create);
32   // receive the update POST
33   app.get("/admin/supplier-update/:id", supplier.findOne);
34   // receive the update POST
35   app.post("/admin/supplier-update", supplier.update);
36   // receive the POST to delete a supplier
37   app.post("/admin/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 module.exports = app;

```

Below the code, the terminal window shows the command being run:

```

bash - ip-10-16-10-153.e x | Immediate (Javascript (br x)
"repositoryName": "employee",
"repositoryUrl": "08245198674.dkr.ecr.us-east-1.amazonaws.com/employee",
"createdAt": "2024-04-28T03:29:09.315000+00:00",
"imageTagMutability": "MUTABLE",
"imageScanningConfiguration": {
  "scanOnPush": false
},
"encryptionConfiguration": {
  "encryptionType": "AES256"
}
}

vclabs:-/environment/microservices/employee (dev) $ aws ecr get-repository-policy --repository-name customer
An error occurred (RepositoryNotFoundException) when calling the GetRepositoryPolicy operation: Repository policy does not exist for the repository with name 'customer' in t
he registry with id '08245198674'
vclabs:-/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '-' -f4)
vclabs:-/environment/microservices/employee (dev) $ echo $account_id
08245198674
vclabs:-/environment/microservices/employee (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vclabs:-/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vclabs:-/environment/microservices/employee (dev) $

```

- Run the appropriate docker command to verify that the images exist and the tags were applied.

Tip: To find the command that you need to run, see [Use the Docker Command Line](#) in the Docker documentation.

Tip: The output of the command should be similar to the following image. Notice that the *latest* tag was applied and that the image names now include the remote repository name where you intend to store it:

Use the following command:

docker images

```

repository -> "REDACTED"
"repositoryUri": "REDACTED.dkr.ecr.us-east-1.amazonaws.com/customer",
"createdAt": "2024-04-29T05:26:34.921000+00:00",
"imageTagMutability": "MUTABLE",
"imageScanningConfiguration": {
    "scanOnPush": false
},
"encryptionConfiguration": {
    "encryptionType": "AES256"
}
}
vclabs:-/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
vclabs:-/environment/microservices/employee (dev) $ echo $account_id
REDACTED
vclabs:-/environment/microservices/employee (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vclabs:-/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vclabs:-/environment/microservices/employee (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
REDACTED.dkr.ecr.us-east-1.amazonaws.com/customer latest b4c4b73955d 27 minutes ago 82.7MB
employee latest bc4c4b73955d 27 minutes ago 82.7MB
<none> efed149753ae 42 minutes ago 82.7MB
REDACTED.dkr.ecr.us-east-1.amazonaws.com/customer latest 2a3653127983 About an hour ago 82.7MB
customer latest 2a3653127983 About an hour ago 82.7MB
node 11-alpine f18da2f58c3d 4 years ago 75.5MB
vclabs:-/environment/microservices/employee (dev) $ 

```

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

Tip: To find the command that you need to run, see [Use the Docker Command Line](#) in the Docker documentation.

Tip: Before running the Docker commands, run the following command to set `account_id` as a variable in the terminal. Then, when you run the Docker commands, you can reference the account ID as `$account_id`.

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

```

// show the update route
32 app.get("/admin/supplier-update/:id", supplier.findOne);
33 // receive the update POST
34 app.post("/admin/supplier-update/", supplier.update);
35 // handle /supplier/:id to delete a supplier
36 app.delete("/admin/supplier/:id", supplier.remove);
37 // handle 404
38 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(`Listening on port ${app_port}`);
48 })
}
}
vclabs:-/environment/microservices/employee (dev) $ aws ecr get-repository-policy --repository-name customer
An error occurred (RepositoryNotFoundException) when calling the GetRepositoryPolicy operation: Repository policy does not exist for the repository with name 'customer' in the registry with id 'REDACTED'
vclabs:-/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
vclabs:-/environment/microservices/employee (dev) $ echo $account_id
REDACTED
vclabs:-/environment/microservices/employee (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vclabs:-/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vclabs:-/environment/microservices/employee (dev) $ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
REDACTED.dkr.ecr.us-east-1.amazonaws.com/customer latest 8880859f60cd 49 minutes ago 82.7MB
customer latest 00f94368886e About an hour ago 82.7MB
employee latest 00f94368886e About an hour ago 82.7MB
<none> 2a3653127983 10 hours ago 82.7MB
node 11-alpine f18da2f58c3d 4 years ago 75.5MB
vclabs:-/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity --output text --query 'Account')
vclabs:-/environment/microservices/employee (dev) $ 

```

Additional tip: The commands that you run should look like the following commands but with `REPLACE_ME` replaced with the correct command:

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

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

The output for *each* Docker command that you run to push each image to Amazon ECR should look similar to the following:

The push refers to repository [642015801240.dkr.ecr.us-east-1.amazonaws.com/node-app]

006e0ec54dba: Pushed

59762f95cb06: Pushed

22736f780b31: Pushed

d81d715330b7: Pushed

1dc7f3bb09a4: Pushed

dcaceb729824: Pushed

f1b5933fe4b5: Pushed

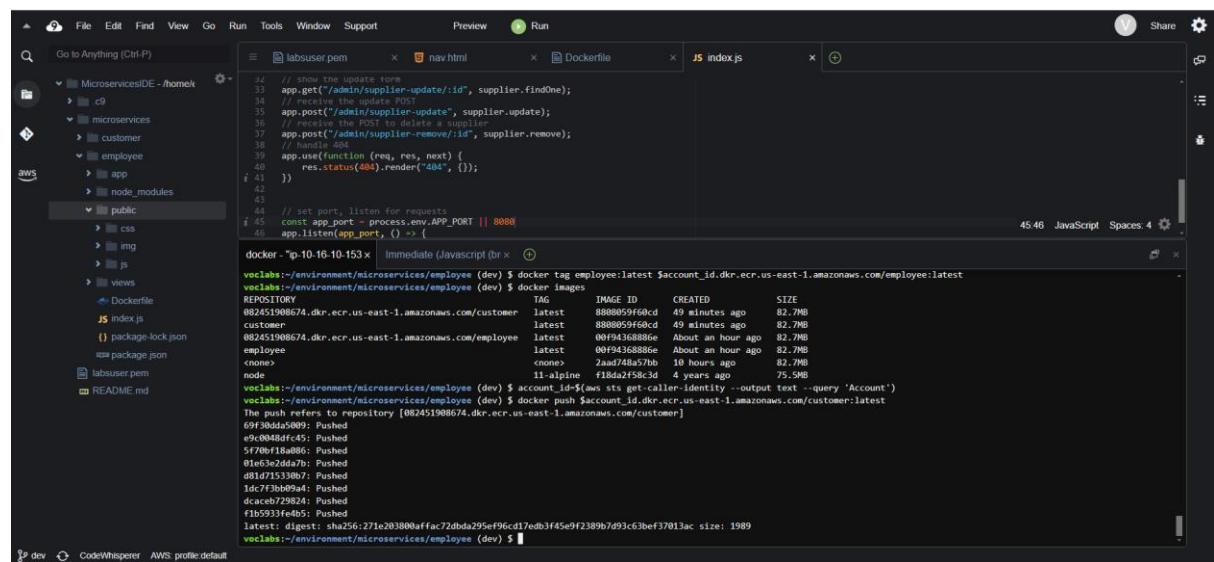
latest: digest: sha256:f75b60addd8d6343b9dff690533a1cd1fbb34ccce6f861e84c857ba7a27b77d

size: 1783

Use the following commands:

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

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



A screenshot of a terminal window showing a file tree on the left and several tabs at the top. The tabs include 'labuser.pem', 'nav.html', 'Dockerfile', and 'JS index.js'. The terminal content shows a sequence of Docker commands and AWS Lambda logs.

```
36 // show the update type
37 app.get("/admin/supplier-update/:id", supplier.findOne);
38 // receive the update POST
39 app.post("/admin/supplier-update", supplier.update);
40 // receive the POST to delete a supplier
41 app.post("/admin/supplier-remove/:id", supplier.remove);
42 // render the update
43 app.use(function (req, res, next) {
44   res.status(404).render("404", {});
45 });
46 // set port, listen for requests
47 const app_port = process.env.APP_PORT || 8080
48 app.listen(app_port, () => {
  docker -w p-10-16-10-153 x  Immediate (Javascript br x)
  vercel:~/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
  vercel:~/environment/microservices/employee (dev) $ docker images
  REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
  082451908674.dkr.ecr.us-east-1.amazonaws.com/customer    latest    8888059f60cd  49 minutes ago   82.7MB
  customer           latest    00f94368868e  About an hour ago  82.7MB
  082451908674.dkr.ecr.us-east-1.amazonaws.com/employee    latest    00f94368868e  About an hour ago  82.7MB
  employee           latest    <none>       2aud748a57b0  10 hours ago   82.7MB
  node               11-alpine  f18da2f58c3d  4 years ago    75.5MB
  vercel:~/environment/microservices/employee (dev) $ account_id.amazonaws.com:latest
  vercel:~/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
  The push refers to repository [082451908674.dkr.ecr.us-east-1.amazonaws.com/customer]
  69f3ddaa5089: Pushed
  e9c0048df45: Pushed
  5f70bf18a86: Pushed
  01e63e2ddaa7b: Pushed
  d81d715330b7: Pushed
  1dc7f3bb09a4: Pushed
  dcaceb729824: Pushed
  f1b5933fe4b5: Pushed
  latest: digest: sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac size: 1989
  vercel:~/environment/microservices/employee (dev) $
```

The screenshot shows a code editor interface with several tabs open. On the left, a file tree displays a directory structure for a project named 'MicroservicesIDE - /home/ec2-user'. The 'public' folder contains files like Dockerfile, index.js, package.json, and README.md. The 'index.js' tab shows a snippet of Node.js code for handling supplier-related requests. The 'Dockerfile' tab shows a standard Dockerfile for building the application. The bottom part of the screen shows the terminal output of a 'docker push' command, which has successfully uploaded two images to an Amazon ECR repository.

```

// show the update form
32 app.get("/admin/supplier-update/:id", supplier.findOne);
33 // receive the update POST
34 app.post("/admin/supplier-update", supplier.update);
35 // handle the POST to delete a supplier
36 app.delete("/admin/supplier-remove/:id", supplier.remove);
37 // handle 404
38 app.use(function (req, res, next) {
39   res.status(404).render("404", {});
40 });
41 }
42
43 // set port, listen for requests
44 const app_port = process.env.APP_PORT || 8080;
45 app.listen(app_port, () => {
46   docker ->p-10-16-10-153x | Immediate Javascript (br x)
The push refers to repository [082451908674.dkr.ecr.us-east-1.amazonaws.com/customer]
69f30ddaa509: Pushed
e9c9048dfc45: Pushed
57f0fe18a086: Pushed
04ec3e2ddaa: Pushed
d81d71533467: Pushed
1dc7f3bb9fa4: Pushed
dcacab298242: Pushed
fb5933feab5: Pushed
latest: digest: sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac size: 1989
vclabs:-/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [082451908674.dkr.ecr.us-east-1.amazonaws.com/employee]
61b026c3875c: Pushed
ae12684c4x0: Pushed
57f0fe18a086: Pushed
04ec3e2ddaa: Pushed
d81d71533467: Pushed
1dc7f3bb9fa4: Pushed
dcacab298242: Pushed
fb5933feab5: Pushed
latest: digest: sha256:6dc2aa5f38d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37 size: 1989
vclabs:-/environment/microservices/employee (dev) $ 

```

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

This screenshot shows the 'Private registry' section of the Amazon ECR console. It lists two repositories: 'customer' and 'employee'. Both repositories were created on April 27, 2024, at different times. They are both disabled and have a manual scan frequency. The encryption type for both is AES-256.

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	082451908674.dkr.ecr.us-east-1.amazonaws.com/customer	April 27, 2024, 23:27:57 (UTC-04)	Disabled	Manual	AES-256
employee	082451908674.dkr.ecr.us-east-1.amazonaws.com/employee	April 27, 2024, 23:29:09 (UTC-04)	Disabled	Manual	AES-256

This screenshot shows the details of the 'customer' repository within the Amazon ECR console. It displays one image, 'latest', which was pushed on April 27, 2024, at 23:56:51 (UTC-04). The image size is 27.70 MB and its digest is sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac.

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	April 27, 2024, 23:56:51 (UTC-04)	27.70	Copy URI	sha256:271e203800affac...

Amazon ECR > Private registry > Repositories > customer > sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac

Image

Details

Image tags
latest

URI
082451908674.dkr.ecr.us-east-1.amazonaws.com/customer/latest

Digest
sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac

General information

Artifact type Image	Repository customer	Pushed at April 27, 2024, 23:56:51 (UTC-04)
Size (MB) 27.70		

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

Amazon ECR > Private registry > Repositories > employee

employee

Images (1)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	April 27, 2024, 23:58:08 (UTC-04)	27.70	Copy URI	sha256:6dc2aaaf538d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37

View push commands Edit

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

Amazon ECR > Private registry > Repositories > employee > sha256:6dc2aaaf538d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37

Image

Details

Image tags
latest

URI
082451908674.dkr.ecr.us-east-1.amazonaws.com/employee/latest

Digest
sha256:6dc2aaaf538d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37

General information

Artifact type Image	Repository employee	Pushed at April 27, 2024, 23:58:08 (UTC-04)
Size (MB) 27.70		

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

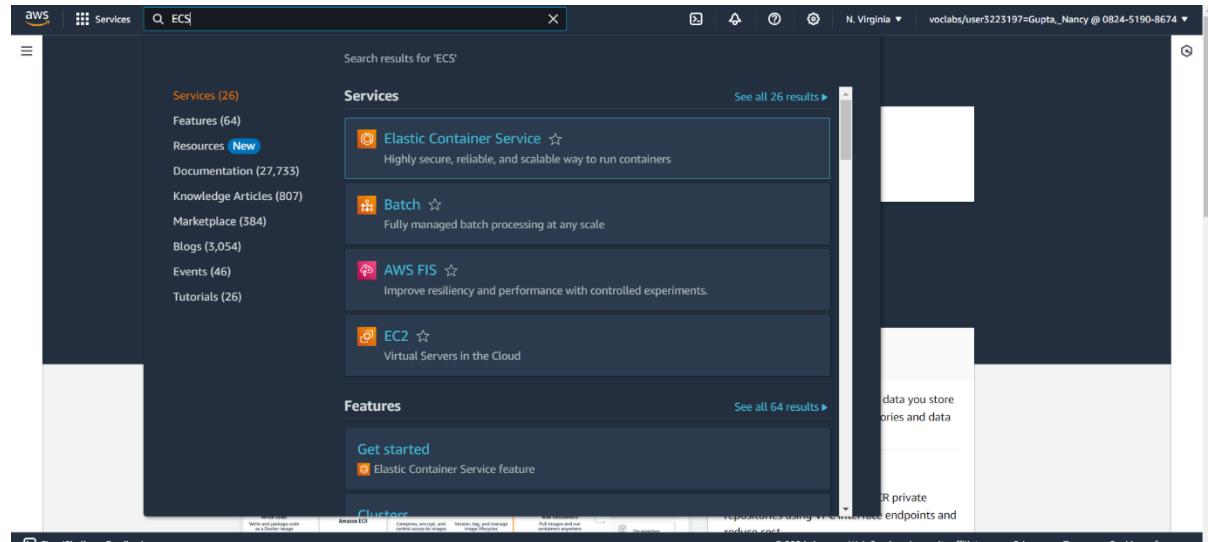
Task 5.2: Create an ECS cluster

In this task, you will create an Amazon ECS cluster.

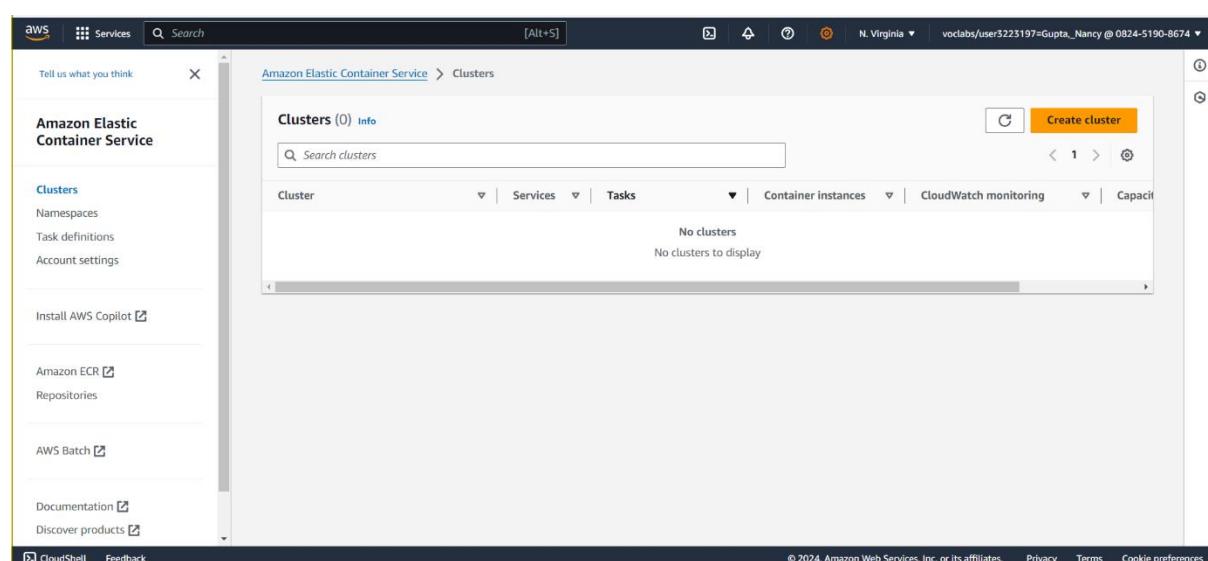
1. Create a serverless AWS Fargate cluster that is named microservices-serverlesscluster

Ensure that it's configured to use *LabVPC*, *PublicSubnet1*, and *PublicSubnet2* (remove any other subnets). *DON'T* select Amazon EC2 instances or ECS Anywhere.

Important: After choosing the button to create the cluster, in the banner that appears across the top of the page, choose **View in CloudFormation**. Wait until the stack that creates the cluster attains the status *CREATE_COMPLETE* before you proceed to the next task. *If the stack fails to create for any reason and therefore rolls back*, repeat these steps to try again. It should succeed the second time.



The screenshot shows the AWS Services search results for 'ECS'. The search bar at the top has 'ECS' typed into it. On the left, there is a sidebar with categories like Services (26), Features (64), Resources (New), Documentation (27,733), Knowledge Articles (807), Marketplace (384), Blogs (3,054), Events (46), and Tutorials (26). The main area displays results under 'Services' and 'Features'. Under 'Services', there are cards for Elastic Container Service (Highly secure, reliable, and scalable way to run containers), Batch (Fully managed batch processing at any scale), AWS FIS (Improve resiliency and performance with controlled experiments), and EC2 (Virtual Servers in the Cloud). Under 'Features', there is a 'Get started' card for the Elastic Container Service feature. A large, semi-transparent modal window is overlaid on the page, partially obscuring the content. The modal has a dark background with white text and contains sections for 'data you store', 'data you store', and 'data you store'.



The screenshot shows the 'Clusters' page for the Amazon Elastic Container Service. The top navigation bar includes 'Tell us what you think', 'Amazon Elastic Container Service', 'Clusters', and 'Create cluster'. The main content area is titled 'Clusters (0) Info' and features a search bar labeled 'Search clusters'. Below the search bar is a table with columns: Cluster, Services, Tasks, Container instances, CloudWatch monitoring, and Capacity. A message 'No clusters' is displayed. The left sidebar contains links for 'Clusters', 'Namespaces', 'Task definitions', 'Account settings', 'Install AWS Copilot', 'Amazon ECR', 'Repositories', 'AWS Batch', 'Documentation', and 'Discover products'. The bottom of the page includes standard AWS footer links: 'CloudShell', 'Feedback', '© 2024, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

Screenshot of the AWS Elastic Container Service (ECS) Create cluster page.

Cluster configuration

Cluster name: microservices-serverlesscluster

Default namespace - optional: Q microservices-serverlesscluster

Infrastructure (Serverless)

AWS Fargate (serverless): Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

Amazon EC2 instances: Manual configurations. Use for large workloads with consistent resource demands.

External instances using ECS Anywhere: Manual configurations. Use to add data center compute.

Monitoring - optional

Tags - optional

Cancel **Create**

Screenshot of the AWS Elastic Container Service (ECS) Create cluster page.

Infrastructure (Serverless)

AWS Fargate (serverless): Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

Amazon EC2 instances: Manual configurations. Use for large workloads with consistent resource demands.

External instances using ECS Anywhere: Manual configurations. Use to add data center compute.

Monitoring - optional

Tags - optional

Cancel **Create**

Screenshot of the AWS Elastic Container Service (ECS) Clusters page.

Clusters (1) Info

Cluster	Services	Tasks	Container instances	CloudWatch monitoring	Capacity
microservices-serverlesscluster	0	No tasks running	0 EC2	Default	No def.

Create cluster

Stack name	Status	Created time	Description
Infra-ECS-Cluster-microservices-serverlesscluster-c6f1a25a	CREATE_COMPLETE	2024-04-28 00:31:56 UTC-0400	The template used to create an ECS Cluster from the ECS Console.
aws-cloud9-MicroservicesIDE-84d2e6feefcda408214113159ea5acf	CREATE_COMPLETE	2024-04-26 23:58:35 UTC-0400	-
c110323a2605598b6548437t1w082451908674	CREATE_COMPLETE	2024-04-26 14:45:58 UTC-0400	Academy Capstone 2 Cfn

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:us-east-1:082451908674:cluster/microservices-serverlesscluster	Active	Default	-

Services	Tasks
Draining	Active
-	Pending
-	Running

Task 5.3: Create a CodeCommit repository to store deployment files

In this task, you will create another CodeCommit repository. This repository will store the task configuration specification files that Amazon ECS will use for each microservice. The repository will also store AppSpec specification files that CodeDeploy will use for each microservice.

1. **Create a new CodeCommit repository that is named deployment to store deployment configuration files.**

The screenshot shows the AWS CloudSearch interface. The search bar at the top contains the query 'codeCommit'. The left sidebar has sections for 'Amazon Elastic Container Service', 'Clusters', 'Amazon ECR', 'AWS Batch', and 'Documentation'. The main content area displays search results for 'Services' and 'Documentation'. The 'Services' section highlights 'CodeCommit' with the description 'Store Code in Private Git Repositories'. The 'Documentation' section highlights 'CodeCommit User Guide'. A modal window titled 'Introducing resource search' is open, explaining how it enables cross-region resource visibility.

The screenshot shows the AWS CodeCommit interface. The left sidebar includes 'Developer Tools' and 'CodeCommit' sections with links like 'Source', 'Getting started', 'Repositories', 'Approval rule templates', 'Artifacts', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The main content area is titled 'Developer Tools > CodeCommit > Repositories'. It shows a table of repositories, with one entry for 'microservices' which was created 1 hour ago. The table includes columns for Name, Description, Last modified, Clone URL, and AWS KMS Key.

The screenshot shows the 'Create repository' wizard in the AWS CodeCommit interface. The title is 'Create repository'. It instructs the user to create a secure repository to store and share code by typing a repository name and description. The 'Repository settings' section contains fields for 'Repository name' (set to 'deployment'), 'Description - optional' (empty), 'Tags' (empty), and 'Additional configuration' (empty). There is also an optional checkbox for 'Enable Amazon CodeGuru Reviewer for Java and Python - optional'. The bottom of the screen shows standard AWS navigation and footer links.

Repository name
deployment
100 characters maximum. Other limits apply.

Description - optional

Tags
Add tag

Additional configuration
AWS KMS key

Enable Amazon CodeGuru Reviewer for Java and Python - optional
Get recommendations to improve the quality of the Java and Python code for all pull requests in this repository.
A service-linked role will be created in IAM on your behalf if it does not exist.

Cancel Create

Name	Description	Last modified	Clone URL	AWS KMS Key
deployment	-	Just now	HTTPS SSH HTTPS (GRC)	arn:aws:kms:us-east-1:08245190674:key/cb78b9cf-6753-41aa-9e4c-eac54ea3156a
microservices	-	1 hour ago	HTTPS SSH HTTPS (GRC)	arn:aws:kms:us-east-1:08245190674:key/cb78b9cf-6753-41aa-9e4c-eac54ea3156a

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

Use the following commands:

```
# Create the deployment directory
```

```
mkdir deployment
```

The screenshot shows the AWS Lambda Functions IDE interface. The left sidebar displays the project structure under 'MicroservicesIDE - /home/ecr'. The main area shows a terminal window with the command 'cd deployment' entered. The output of the command is displayed, showing the creation of a 'deployment' directory and its contents.

```

bash - "ip-10-16-10-153.x" Immediate (Javascript br x
cd deployment
daceb729824: Pushed
latest: digest: sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac size: 1989
veclabs:-/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [082451908674.dkr.ecr.us-east-1.amazonaws.com/employee]
61b026c3875c: Pushed
ae1268c4ca50: Pushed
5f700f18a86: Pushed
01e63e2dd47b: Pushed
d81d71513867: Pushed
1d7f3bb9b4: Pushed
daceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:6dc2aaef538d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37 size: 1989
veclabs:-/environment/microservices/employee (dev) $ mkdir deployment
veclabs:-/environment/microservices/employee (dev) $ rm deployment
rm: cannot remove 'deployment': Is a directory
veclabs:-/environment/microservices/employee (dev) $ rm -rf deployment
veclabs:-/environment/microservices/employee (dev) $ cd ../..
veclabs:-/environment $ mkdir deployment
veclabs:-/environment $ 

```

Navigate into the deployment directory

cd deployment

The screenshot shows the AWS Lambda Functions IDE interface. The left sidebar displays the project structure under 'MicroservicesIDE - /home/ecr'. The main area shows a terminal window with the command 'git init' entered. The output of the command is displayed, showing the initialization of a new Git repository.

```

bash - "ip-10-16-10-153.x" Immediate (Javascript br x
git init
daceb729824: Pushed
latest: digest: sha256:271e203800affac72dbda295ef96cd17edb3f45e9f2389b7d93c63bef37013ac size: 1989
veclabs:-/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [082451908674.dkr.ecr.us-east-1.amazonaws.com/employee]
61b026c3875c: Pushed
ae1268c4ca50: Pushed
5f700f18a86: Pushed
01e63e2dd47b: Pushed
d81d71513867: Pushed
1d7f3bb9b4: Pushed
daceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:6dc2aaef538d7588a7396a74915c30bce26d8ecfc02ff621011d2be85be0a0d37 size: 1989
veclabs:-/environment/microservices/employee (dev) $ mkdir deployment
veclabs:-/environment/microservices/employee (dev) $ rm deployment
rm: cannot remove 'deployment': Is a directory
veclabs:-/environment/microservices/employee (dev) $ rm -rf deployment
veclabs:-/environment/microservices/employee (dev) $ cd ../..
veclabs:-/environment $ mkdir deployment
veclabs:-/environment $ 

```

Initialize the directory as a Git repository

git init

The screenshot shows the AWS Lambda Code Editor interface. On the left, a file tree displays a project structure under 'MicroservicesIDE - /home/ec2-user'. The current file is 'index.js' in the 'public' directory. The code in index.js includes logic for handling supplier updates and removals. A terminal window at the bottom shows the command 'git checkout -b dev' being run, followed by a message indicating the switch to a new branch.

```

// show the update form
app.get("/admin/supplier-update/:id", supplier.findOne);
// receive the POST
app.post("/admin/supplier-update", supplier.update);
// receive the POST to delete a supplier
app.post("/admin/supplier-remove/:id", supplier.remove);
app.use(function (req, res, next) {
  res.status(404).render("404", {});
})
// set port, listen for requests
const app_port = process.env.APP_PORT || 8080
app.listen(app_port, () => {
  console.log(`Pushed: ${sha}`);
  latest: digest: sha256:5dc2aaef538d7588a7f0674015c30bce268ecfc82ff621011d2be85be0a0d37 size: 1989
  vocabs:-/environment/microservices/employee (dev) $ mkdir deployment
  vocabs:-/environment/microservices/employee (dev) $ rm -rf deployment
  rm: cannot remove 'deployment': Is a directory
  vocabs:-/environment/microservices/employee (dev) $ rm -rf deployment
  vocabs:-/environment/microservices/employee (dev) $ cd ../..
  vocabs:-/environment $ mkdir deployment
  vocabs:-/environment $ cd deployment
  vocabs:-/environment/deployment $ git init
  hint: Using 'master' as the name for the initial branch. This default branch name
  hint: is subject to change. To configure the initial branch name to use in all
  hint: of your new repositories, which will suppress this warning, call:
  hint:
  hint: git config --global init.defaultBranch <name>
  hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
  hint: 'development'. The just-created branch can be renamed via this command:
  hint:
  hint: git branch -m <name>
  Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
  vocabs:-/environment/deployment (master) $ git checkout -b dev
  vocabs:-/environment/deployment (dev) $ 

```

Create and switch to a new branch named "dev"

git checkout -b dev

This screenshot is identical to the one above, showing the AWS Lambda Code Editor interface with the same file tree and terminal output. It demonstrates the command 'git checkout -b dev' being run again, confirming the switch to the 'dev' branch.

```

// show the update form
app.get("/admin/supplier-update/:id", supplier.findOne);
// receive the POST
app.post("/admin/supplier-update", supplier.update);
// receive the POST to delete a supplier
app.post("/admin/supplier-remove/:id", supplier.remove);
app.use(function (req, res, next) {
  res.status(404).render("404", {});
})
// set port, listen for requests
const app_port = process.env.APP_PORT || 8080
app.listen(app_port, () => {
  console.log(`Pushed: ${sha}`);
  latest: digest: sha256:5dc2aaef538d7588a7f0674015c30bce268ecfc82ff621011d2be85be0a0d37 size: 1989
  vocabs:-/environment/microservices/employee (dev) $ mkdir deployment
  vocabs:-/environment/microservices/employee (dev) $ rm -rf deployment
  rm: cannot remove 'deployment': Is a directory
  vocabs:-/environment/microservices/employee (dev) $ rm -rf deployment
  vocabs:-/environment/microservices/employee (dev) $ cd ../..
  vocabs:-/environment $ mkdir deployment
  vocabs:-/environment $ cd deployment
  vocabs:-/environment/deployment $ git init
  hint: Using 'master' as the name for the initial branch. This default branch name
  hint: is subject to change. To configure the initial branch name to use in all
  hint: of your new repositories, which will suppress this warning, call:
  hint:
  hint: git config --global init.defaultBranch <name>
  hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
  hint: 'development'. The just-created branch can be renamed via this command:
  hint:
  hint: git branch -m <name>
  Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
  vocabs:-/environment/deployment (master) $ git checkout -b dev
  vocabs:-/environment/deployment (dev) $ 

```

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

In this task, you will create a task definition file for each microservice and then register the task definitions with Amazon ECS.

1. In the new deployment directory, create an empty file named **taskdef-customer.json**

```

// show the update form
app.get("/admin/supplier-update/:id", supplier.findOne);
// receive the update POST
app.post("/admin/supplier-update", supplier.update);
// receive the POST to delete a supplier
app.post("/admin/supplier-remove/:id", supplier.remove);
// handle 404
app.use(function (req, res, next) {
  res.status(404).render("404", {});
})
// set port, listen for requests
const app_port = process.env.APP_PORT || 8080
app.listen(app_port, () => {
  console.log(`App listening on port ${app_port}!`);
  console.log(`Press Ctrl+C to stop.`);
});

// This function is not intended to be JSON.parse()'d, instead
// it's serialized and sent to the container
// to make it easier to define environment variables
// that only need to be defined once across many tasks.
// See https://docs.aws.amazon.com/lambda/latest/dg/task-definition-environment.html
// for more information.
module.exports = {
  taskType: "Node.js"
};

```

```

{
  "containerDefinitions": [
    {
      "name": "customer",
      "image": "customer",
      "environment": [
        {
          "name": "APP_DB_HOST",
          "value": "<RDS-ENDPOINT>"
        }
      ]
    }
  ]
}

```

2. Edit the taskdef-customer.json file.

- Paste the following JSON code into the file:

```
{

```

```
"containerDefinitions": [
```

```
{
```

```
  "name": "customer",
```

```
  "image": "customer",
```

```
  "environment": [
```

```
{
```

```
    "name": "APP_DB_HOST",
```

```
    "value": "<RDS-ENDPOINT>"
```

```
        },
      ],
      "essential": true,
      "portMappings": [
        {
          "hostPort": 8080,
          "protocol": "tcp",
          "containerPort": 8080
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "awslogs-capstone",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "awslogs-capstone"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024",
  "executionRoleArn": "arn:aws:iam::<ACCOUNT-ID>:role/PipelineRole",
  "family": "customer-microservice"
}
```

```

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

```

bash - ip-10-16-10-153.e.x | Immediate (Javascript (br x) | Switched to a new branch 'dev'

- Replace a couple values in the file:

- On line 37, replace <ACCOUNT-ID> with the actual account ID.

Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN
MicroservicesIDE	Open	EC2 instance	Secure Shell (SSH)	Owner	arn:aws:sts:08245190:role/voclabs/user3223197

```

1  {
2     "name": "customer",
3     "image": "customer",
4     "environment": [
5         {
6             "name": "APP_DB_HOST",
7             "value": "<RDS-ENDPOINT>"
8         }
9     ],
10    "essential": true,
11    "portMappings": [
12        {
13            "hostPort": 8080,
14            "protocol": "tcp",
15            "containerPort": 8080
16        }
17    ],
18    "logConfiguration": {
19        "logDriver": "awslogs",
20        "options": {
21            "awslogs-create-group": "true",
22            "awslogs-group": "awslogs-capstone",
23            "awslogs-region": "us-east-1",
24            "awslogs-stream-prefix": "awslogs-capstone"
25        }
26    }
27}

```

bash - ip-10-16-10-153.e x Immediate (Javascript) (br x)

hint: 'development'. The just-created branch can be renamed via this command:
hint: hint: git branch -m <name>
Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
vocabs:/environment/deployment (master) \$ git checkout -b dev
Switched to a new branch 'dev'
vocabs:/environment/deployment (dev) \$ cd ~/environment/deployment
vocabs:/environment/deployment (dev) \$ touch taskdef-customer.json
vocabs:/environment/deployment (dev) \$ aws sts get-caller-identity --query 'Account' --output text
802451908674
vocabs:/environment/deployment (dev) \$ aws rds describe-db-instances --db-instance-identifier your-db-instance-name --query 'DBInstances[0].Endpoint.Address' --output text
An error occurred (DBInstanceNotFound) when calling the DescribeDBInstances operation: DBInstance your-db-instance-name not found.
vocabs:/environment/deployment (dev) \$

```

1  {
2     "containerDefinitions": [
3         {
4             "name": "customer",
5             "image": "customer",
6             "environment": [
7                 {
8                     "name": "APP_DB_HOST",
9                     "value": "supplierdb.cjoadg7ycvt.us-east-1.rds.amazonaws.com"
10                }
11            ],
12            "essential": true,
13            "portMappings": [
14                {
15                    "hostPort": 8080,
16                    "protocol": "tcp",
17                    "containerPort": 8080
18                }
19            ],
20            "logConfiguration": {
21                "logDriver": "awslogs",
22                "options": {
23                    "awslogs-create-group": "true",
24                    "awslogs-group": "awslogs-capstone",
25                    "awslogs-region": "us-east-1",
26                    "awslogs-stream-prefix": "awslogs-capstone"
27                }
28            }
29        },
30        {
31            "requiresCompatibilities": [
32                "FARGATE"
33            ],
34            "networkMode": "awsvpc",
35            "cpu": "512",
36            "memory": "1024",
37            "executionRoleArn": "arn:aws:iam::082451908674:role/PipelineRole",
38            "family": "customer-microservice"
39        }
40    }
41}

```

bash - ip-10-16-10-153.e x Immediate (Javascript) (br x)

An error occurred (DBInstanceNotFound) when calling the DescribeDBInstances operation: DBInstance your-db-instance-name not found.

■ On line 9, replace <RDS-ENDPOINT> with the actual RDS endpoint.

The screenshot shows the Amazon RDS console with the 'Databases' section selected. A modal window provides information about Blue/Green Deployments. The main table lists one database instance:

DB identifier	Status	Role	Engine	Region & AZ	Size	Recommendations	CPU
supplierdb	Available	Instance	MySQL Community	us-east-1b	db.t3.micro	5 Informational	

Screenshot of the AWS RDS console showing the database `supplierdb`. The left sidebar shows various RDS management options like Performance insights, Snapshots, and Automated backups. The main summary table provides details such as DB identifier (`supplierdb`), Status (`Available`), Role (`Instance`), Engine (`MySQL Community`), and Recommendations (`5 Informational`). Below the summary, tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Zero-ETL integrations, Maintenance & backups, and Tags are visible.

Screenshot of a terminal window titled "dev" showing the AWS CodeWhisperer interface. The terminal displays the contents of the file `taskdef-customer.json`, which contains a JSON configuration for a Lambda function. The configuration includes container definitions, port mappings, log configurations, and network mode. The terminal also shows the command `bash -ip-10-16-10-153.e x` and the message "An error occurred (DBInstanceNotFound) when calling the DescribeDBInstances operation: DBInstance your-db-instance-name not found." The AWS CloudWatch logs tab is open at the bottom.

- Save the changes.

Screenshot of the AWS CodeWhisperer interface after saving changes. The terminal window now shows the updated configuration for the Lambda function, reflecting the changes made earlier. The AWS CloudWatch logs tab is still visible at the bottom.

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
```

```
aws      ecs      register-task-definition --cli-input-json file:///home/ec2-user/environment/deployment/taskdef-customer.json
An error occurred (DBInstanceNotFound) when calling the DescribeDBInstances operation: DBInstance your-db-instance-name not found.
```

```
aws      ecs      register-task-definition --cli-input-json file:///home/ec2-user/environment/deployment/taskdef-customer.json
taskDefinitionArn: "arn:aws:ecs:us-east-1:882451908674:task-definition/customer-microservice:1"
```

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

Tip: Consult the [ECS documentation](#) if it is helpful.

Search results for 'ECS'

Services (26) See all 26 results ▾

Elastic Container Service Highly secure, reliable, and scalable way to run containers

Batch Fully managed batch processing at any scale

AWS FIS Improve resiliency and performance with controlled experiments.

EC2 Virtual Servers in the Cloud

Features (64) See all 64 results ▾

Get started

Tell us what you think X

Amazon Elastic Container Service > Clusters

Clusters (1) Info

Cluster	Services	Tasks	Container instances	CloudWatch monitoring	Capacity
microservices-serverlesscluster	0	No tasks running	0 EC2	Default	No d

Tell us what you think X

Amazon Elastic Container Service > Task definitions

Task definitions (1) Info

Task definition	Status of last revision
customer-microservice	ACTIVE

Notice that the revision number displays after the task definition name

The screenshot shows the AWS Elastic Container Service (ECS) Task Definitions page. In the left sidebar, under 'Task definitions', the 'customer-microservice' task is selected. The main content area displays the 'customer-microservice (1/1) Info' section. At the top right, there are buttons for 'Deploy', 'Actions', and 'Create new revision'. Below these are filters for 'Filter task definition revisions by value' (set to 'Active') and 'Status' (set to 'ACTIVE'). A table lists one task definition revision: 'customer-microservice:1'.

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

The screenshot shows a terminal window in a developer environment. The current directory is 'MicroservicesIDE - /home/e'. The user has run the command 'touch taskdef-employee.json' to create the file. The terminal output shows:

```

bash - "ip-10-10-10-153 ~" 
voclabs:~/environment $ cd deployment
bash: cd: deployment: No such file or directory
voclabs:~/environment $ cd deployment
voclabs:~/environment/deployment (dev) $ touch taskdef-employee.json
voclabs:~/environment/deployment (dev) $ 

```

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

```

{
    "containerDefinitions": [
        {
            "name": "customer",
            "image": "customer",
            "environment": [
                {
                    "name": "APP_DB_HOST",
                    "value": "supplierdb.cjoadg7hyct.us-east-1.rds.amazonaws.com"
                }
            ],
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 8080,
                    "protocol": "tcp",
                    "containerPort": 8080
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-capstone",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "awslogs-capstone"
                }
            }
        },
        {
            "name": "employee",
            "image": "employee",
            "environment": [
                {
                    "name": "APP_DB_HOST",
                    "value": "supplierdb.cjoadg7hyct.us-east-1.rds.amazonaws.com"
                }
            ],
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 8080,
                    "protocol": "tcp",
                    "containerPort": 8080
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-capstone",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "awslogs-capstone"
                }
            }
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "512",
    "memory": "1024",
    "executionRoleArn": "arn:aws:iam::082451908674:role/PipelineRole",
    "family": "customer-microservice"
}

```

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

```

{
    "containerDefinitions": [
        {
            "name": "employee",
            "image": "employee",
            "environment": [
                {
                    "name": "APP_DB_HOST",
                    "value": "supplierdb.cjoadg7hyct.us-east-1.rds.amazonaws.com"
                }
            ],
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 8080,
                    "protocol": "tcp",
                    "containerPort": 8080
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-capstone",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "awslogs-capstone"
                }
            }
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "512",
    "memory": "1024",
    "executionRoleArn": "arn:aws:iam::082451908674:role/PipelineRole",
    "family": "employee-microservice"
}

```

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

Use the following command:

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

```

taskdef-customer.json taskdef-employee.json
4      name : employee ,
5      "image": "employee",
6      "environment": [
7      {
8      "taskDefinition": {
9          "taskDefinitionArn": "arn:aws:ecs:us-east-1:082451908674:task-definition/employee-microservice:1",
10         "taskDefinition": {
11             "taskDefinitionArn": "arn:aws:ecs:us-east-1:082451908674:task-definition/employee-microservice:1",
12             "taskDefinition": {
13                 "taskDefinitionArn": "arn:aws:ecs:us-east-1:082451908674:task-definition/employee-microservice:1",
14                 "containerDefinitions": [
15                     {
16                         "name": "employee",
17                         "image": "employee",
18                         "cpu": 0,
19                         "portMappings": [
20                             {
21                                 "containerPort": 8080,
22                                 "hostPort": 8080,
23                                 "protocol": "tcp"
24                             }
25                         ],
26                         "essential": true,
27                         "environment": [
28                             {
29                                 "name": "APP_DB_HOST",
30                                 "value": "supplierdb.joadg7hycvt.us-east-1.rds.amazonaws.com"
31                             }
32                         ],
33                         "mountPoints": [],
34                         "volumesFrom": []
35                     }
36                 ]
37             }
38         }
39     }
40   }
41
bash -ip-10-16-10-153 ~ $ aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdef-employee.json"

```

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

The screenshot shows the AWS Cloud9 Services page. On the left, there's a sidebar with links like 'My environments', 'Shared with me', 'All account environments', 'Documentation', 'Blogs', 'Events', and 'Tutorials'. The main area has a search bar for 'ECS'. Below it, there are sections for 'Services' and 'Features'. Under 'Services', 'Elastic Container Service' is highlighted with a description: 'Highly secure, reliable, and scalable way to run containers'. Other services listed include 'Batch', 'AWS FIS', and 'EC2'. Under 'Features', there's a 'Get started' link for the 'Elastic Container Service feature'.

The screenshot shows the Amazon Elastic Container Service (ECS) console. The left sidebar includes links for 'Clusters', 'Namespaces', 'Task definitions', 'Account settings', 'Install AWS Copilot', 'Amazon ECR', 'Repositories', 'AWS Batch', 'Documentation', and 'Discover products'. The main content area is titled 'Clusters (1) Info' and shows a table with one entry: 'microservices-serverlesscluster'. The table columns are 'Cluster', 'Services', 'Tasks', 'Container instances', 'CloudWatch monitoring', and 'Capacity'. The 'Cluster' row contains 'microservices-serverlesscluster', '0', 'No tasks running', '0 EC2', 'Default', and 'No d'. There are also 'Create cluster' and search/filter buttons at the top of the table.

The screenshots show the AWS Cloud Console interface for the Amazon Elastic Container Service (ECS). The left sidebar navigation includes 'Clusters', 'Namespaces', **Task definitions**, 'Account settings', 'Install AWS Copilot', 'Amazon ECR', 'Repositories', 'AWS Batch', 'Documentation', and 'Discover products'. The top navigation bar shows 'Amazon Elastic Container Service > Task definitions'. The main content area displays the 'Task definitions (2)' list with columns for 'Task definition' and 'Status of last revision'. Two entries are listed: 'customer-microservice' and 'employee-microservice', both marked as 'ACTIVE'. The second screenshot provides a detailed view of the 'employee-microservice' task definition, showing 'Task definition: revision' and 'employee-microservice:1' with 'Status' set to 'ACTIVE'.

Task 5.5: Create AppSpec files for CodeDeploy for each microservice

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

1. Create an AppSpec file for the *customer* microservice.

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

```

{
  "name": "employee",
  "image": "employee",
  "environment": [
    {
      "name": "APP_DB_HOST",
      "value": "suppliedb.cjoadg7hyct.us-east-1.rds.amazonaws.com"
    }
  ],
  "essential": true,
  "portMappings": [
    {
      "hostPort": 8080,
      "protocol": "tcp",
      "containerPort": 8080
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "lambda"
    }
  }
}

```

- Paste the following YAML code into the file:

Important: DON'T modify <TASK_DEFINITION>. This setting will be updated automatically when the pipeline runs.

version: 0.0

Resources:

- TargetService:

Type: AWS::ECS::Service

Properties:

TaskDefinition: <TASK_DEFINITION>

LoadBalancerInfo:

ContainerName: "customer"

ContainerPort: 8080

Note: This file is in YAML format. In YAML, indentation is important. Verify that the code in your file maintains the indentation levels as shown in the previous code block.

- Save the changes.

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file structure of the 'MicroservicesIDE - home' directory is visible, including files like taskdef-customer.json, taskdef-employee.json, appspec-customer.yaml, Dockerfile, index.js, package-lock.json, package.json, labuser.pem, and README.md. In the center, three tabs are open: 'taskdef-customer.json', 'taskdef-employee.json', and 'appspec-customer.yaml'. The 'appspec-customer.yaml' tab contains the following YAML code:

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

Below the tabs, a terminal window titled 'bash - *p-10-16-10-153.e*x' shows the command: 'touch appspec-employee.yaml'. The bottom status bar indicates the session is on 'dev' and shows AWS profile information.

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

- Name the file *appspec-employee.yaml*

The screenshot shows the AWS Cloud9 IDE interface. The file structure remains the same as the previous screenshot. The 'appspec-employee.yaml' file has been added to the directory. The 'appspec-customer.yaml' tab still contains the original YAML code. The terminal window now shows the command: 'touch appspec-employee.yaml'. The bottom status bar indicates the session is on 'dev' and shows AWS profile information.

- The contents of the file should be the same as the *appspec-customer.yaml* file. However, change customer`` on the containerName line to be employee`

```

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

```

Task 5.6: Update files and check them into CodeCommit

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

1. Edit the taskdef-customer.json file.

- **Modify line 5 to match the following line:**

"image": "<IMAGE1_NAME>,"

- **Save the change.**

Analysis: <IMAGE1_NAME> is not a valid image name, which is why you originally set the image name to *customer* before running the AWS CLI command to register the first revision of the file with Amazon ECS. However, at this point in the project, it's important to set the image value to a placeholder text value. Later in this project, when you configure a pipeline, you will identify IMAGE1_NAME as placeholder text that can be dynamically updated. In summary, CodePipeline will set the correct image name dynamically at runtime.

```

"image": "<IMAGE1_NAME>,"
```

2. Edit the taskdef-employee.json file.

- Modify line 5 to match the following line:

"image": "<IMAGE1_NAME>,"

- Save the change.

The screenshot shows the AWS Lambda Code Editor interface. The left sidebar lists files: taskdef-customer.json, taskdef-employee.json, appspec-customer.yaml, appspec-employee.yaml, Dockerfile, index.js, package-lock.json, package.json, labuser.pem, and README.md. The right pane displays the content of taskdef-employee.json. Line 5 has been modified from "image": "awslogs-capstone" to "image": "<IMAGE1_NAME>". The bottom terminal window shows the command 'cd deployment' followed by 'touch appspec-customer.yaml' and 'touch appspec-employee.yaml'. The status bar at the bottom indicates 'dev' and 'AWS profile default'.

```
1  {
2      "containerDefinitions": [
3          {
4              "name": "employee",
5              "image": "<IMAGE1_NAME>",
6              "environment": [
7                  {
8                      "name": "APP_DB_HOST",
9                      "value": "supplierdb.cjoadg7hycvt.us-east-1.rds.amazonaws.com"
10                 }
11             ],
12             "essential": true,
13             "portMappings": [
14                 {
15                     "hostPort": 8088,
16                     "protocol": "tcp",
17                     "containerPort": 8088
18                 }
19             ],
20             "logConfiguration": [
21                 {
22                     "logDriver": "awslogs",
23                     "options": [
24                         "awslogs-create-group", "true",
25                         "awslogs-group": "awslogs-capstone",
26                         "awslogs-region": "us-east-1",
27                         "awslogs-stream-prefix": "awslogs-capstone"
28                     ]
29                 }
30             ],
31             "requiresCompatibilities": [
32                 "FARGATE"
33             ]
34         }
35     ]
36 }
```

3. Push all four files to CodeCommit.

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

The screenshot shows the AWS Lambda Code Editor interface. The left sidebar lists the same files as before. The right pane shows the terminal output after pushing the changes to CodeCommit. It shows the command 'git add .', 'git commit -m "Pushing updated files to CodeCommit"', and 'git push'. The status bar at the bottom indicates 'dev' and 'AWS profile default'.

```
bash - *ip-10-16-10-153.e x | ⏎
veclabs:~/environment $ cd deployment
veclabs:~/environment/deployment (dev) $ touch appspec-customer.yaml
veclabs:~/environment/deployment (dev) $ touch appspec-employee.yaml
veclabs:~/environment/deployment (dev) $ git status
On branch dev
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    appspec-customer.yaml
    appspec-employee.yaml
    taskdef-customer.json
    taskdef-employee.json

nothing added to commit but untracked files present (use "git add" to track)
veclabs:~/environment/deployment (dev) $ ⏎
```

The screenshot shows the AWS CodeWhisperer IDE interface. On the left is a file tree for a project named 'MicroservicesIDE - /home/ec2-user'. The 'deployment' folder contains several files: 'appspec-customer.yaml', 'appspec-employee.yaml', 'taskdef-customer.json', and 'taskdef-employee.json'. The terminal window on the right displays a bash session on a host with IP 10.16.10.153. The user runs 'cd deployment', then 'git add appspec-customer.yaml', 'git add appspec-employee.yaml', 'git add taskdef-customer.json', and 'git add taskdef-employee.json'. They then run 'git status' which shows 'On branch dev' and 'No commits yet'. The user then runs 'git add <file>...' to include untracked files in the commit, adds 'appspec-customer.yaml', 'appspec-employee.yaml', 'taskdef-customer.json', and 'taskdef-employee.json', and finally runs 'git commit -m "Updated task definition files"'.

This screenshot is nearly identical to the one above, showing the same file tree and terminal session. The difference is in the terminal output of the 'git commit' command. Instead of just committing the task definition files, it also includes the untracked files ('appspec-customer.yaml', 'appspec-employee.yaml', 'taskdef-customer.json', and 'taskdef-employee.json') in the commit message: 'git commit -m "Updated task definition files
4 files changed, 96 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'.

The screenshot shows the AWS CodeCommit service in the AWS Management Console. The left sidebar has a 'Developer Tools' section with a 'CodeCommit' tab selected. Under 'Source' there are links for 'Getting started', 'Repositories', 'Approval rule templates', 'Artifacts', 'Build', 'Deploy', 'Pipeline', and 'Settings'. Below these are 'Go to resource' and 'Feedback' links. The main content area shows a 'Repositories' list with two entries: 'deployment' and 'microservices'. Each entry has columns for 'Name', 'Description', 'Last modified', 'Clone URL', and 'AWS KMS Key'. The 'deployment' entry was modified 1 hour ago, while the 'microservices' entry was modified 2 hours ago. Both entries have 'HTTPS' and 'SSH' clone URLs, and 'HTTPS (GCR)' options. The AWS KMS keys listed are arn:aws:kms:us-east-1:082451908674:key/cb78b9cf-6753-41aa-9e4c-eac54ea3156a and arn:aws:kms:us-east-1:082451908674:key/cb78b9cf-6753-41aa-9e4c-eac54ea3156a.

The screenshot shows the AWS Lambda Task Definition Editor interface. The left sidebar lists files: appspec CUSTOMER.yaml, appspec EMPLOYEE.yaml, taskdef CUSTOMER.json, taskdef EMPLOYEE.json, microservices, customer, employee, app, node_modules, public, css, img, js, views, Dockerfile, index.js, package-lock.json, package.json, labuser.pem, and README.md. The right pane displays the JSON configuration for the task definition:

```
1  {
2    "containerDefinitions": [
3      {
4        "name": "employee",
5        "image": "<IMAGE1_NAME>",
6        "environment": [
7          {
8            "name": "APP_DB_HOST",
9            "value": "supplierdb.cjoadg7hycvt.us-east-1.rds.amazonaws.com"
10           }
11         ],
12         "essential": true,
13         "portMappings": [
14           {
15             "hostPort": 8080,
16             "containerPort": 8080
17           }
18         ]
19       }
20     ]
21   }
```

Below the code editor, a terminal window shows the command history for pushing the task definition to Git:

```
bash - *ip-10-16-10-153 ec2 ~
git remote add origin <repository-url>
git commit -m "Updated task definition files"
git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository using
git remote add <name> <url>
and then push using the remote name
git push <name>
git remote add origin <repository-url>
bash: syntax error near unexpected token `newline'
git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
git push
```

The screenshot shows the AWS Lambda Task Definition Editor interface. The left sidebar lists the same files as the first screenshot. The right pane displays the JSON configuration for the task definition, identical to the first screenshot.

Below the code editor, a terminal window shows the command history for pushing the task definition to Git, including the addition of a remote repository and a successful push:

```
git remote add origin <repository-url>
git commit -m "Updated task definition files"
git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.05 KIB | 535.00 KIB/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
git push
```

The screenshot shows the AWS Lambda Task Definition Editor interface. The left sidebar lists the same files as the previous screenshots. The right pane displays the JSON configuration for the task definition, identical to the previous screenshots.

Below the code editor, a terminal window shows the command history for pushing the task definition to Git, including the addition of a remote repository and a successful push:

```
git remote add origin <repository-url>
git commit -m "Updated task definition files"
git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.05 KIB | 535.00 KIB/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
git push
```

The screenshots show the AWS CodeCommit interface. The top screenshot displays the 'Commits' tab, showing a single commit with ID 4970d713, which updated task definition files. The bottom screenshot displays the 'Code' tab, showing four configuration files: appspec-customer.yaml, appspec-employee.yaml, taskdef-customer.json, and taskdef-employee.json.

Phase 6: Creating target groups and an Application Load Balancer

In this phase, you will create an Application Load Balancer, which provides an endpoint URL. This URL will act as the HTTPS entry point for customers and employees to access your application through a web browser. The load balancer will have listeners, which will have routing and access rules that determine which target group of running containers the user request should be directed to.

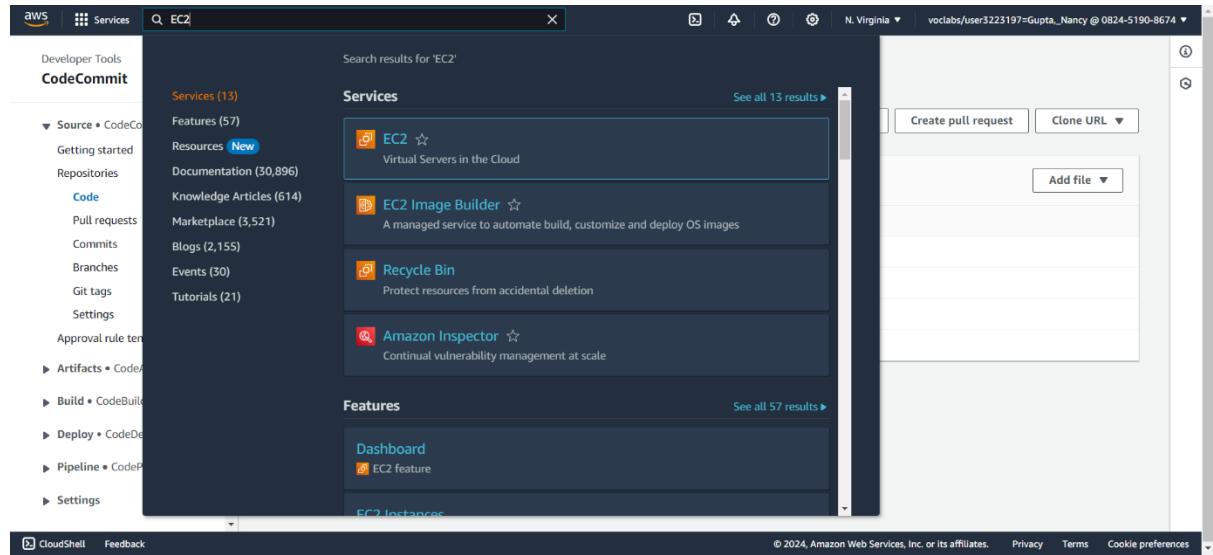
Task 6.1: Create four target groups

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

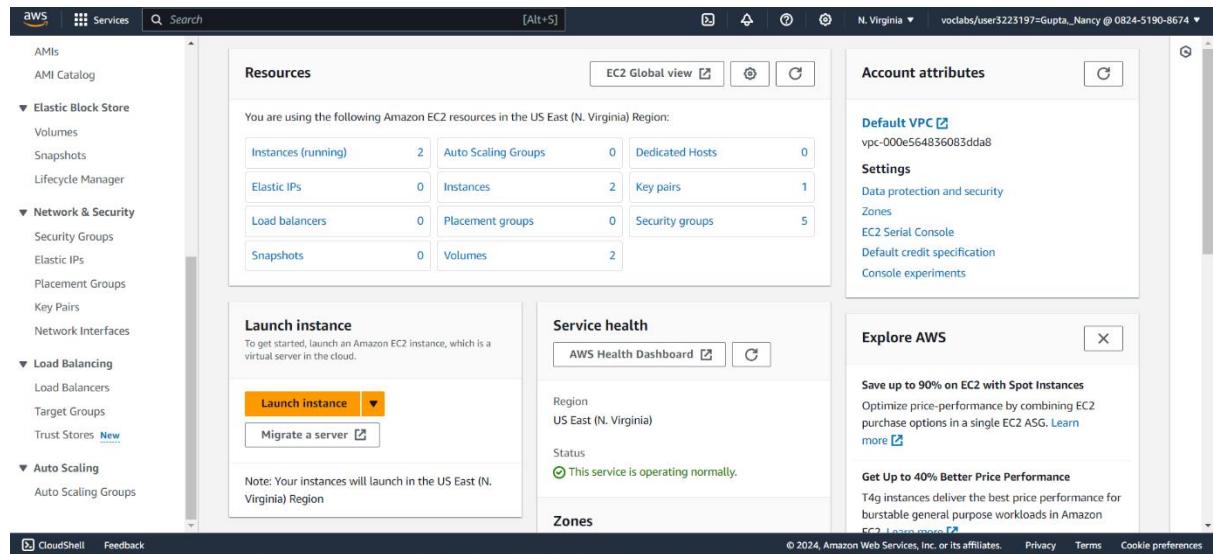
Note: Blue/green is a deployment strategy where you create two separate but identical environments. One environment (blue) runs the current application version, and one environment (green) runs the new application version. For more information, see [Blue/Green Deployments](#) in the *Overview of Deployment Options on AWS* whitepaper.

1. Create the first target group for the *customer* microservice.

- Navigate to the Amazon EC2 console.



- In the navigation pane, choose Target Groups.



- Choose Create target group and configure the following:

The screenshot shows the AWS Management Console with the EC2 service selected. In the left navigation pane, under the Network & Security section, the 'Target Groups' option is highlighted. The main content area displays the 'Target groups' page with a heading 'Target groups Info'. A search bar at the top says 'Filter target groups'. Below it is a table with columns: Name, ARN, Port, Protocol, Target type, and Load balancer. A message 'No target groups' is displayed, followed by 'You don't have any target groups in us-east-1'. At the bottom, there is a button labeled 'Create target group'.

▪ Choose a target type: Choose IP addresses.

The screenshot shows the 'Create target group' wizard at Step 1: 'Specify group details'. The title is 'Specify group details' and a sub-instruction says 'Your load balancer routes requests to the targets in a target group and performs health checks on the targets.' Below this is a 'Basic configuration' section with a note: 'Settings in this section can't be changed after the target group is created.' It contains three options: 'Instances' (radio button), 'IP addresses' (radio button, currently selected), and 'Lambda function' (radio button). The 'IP addresses' section includes a bulleted list: 'Supports load balancing to VPC and on-premises resources.', 'Facilitates routing to multiple IP addresses and network interfaces on the same instance.', 'Offers flexibility with microservice based architectures, simplifying inter-application communication.', and 'Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.'

▪ Target group name: Enter customer-tg-one

The screenshot shows the 'Create target group' wizard at Step 1: 'Specify group details'. The 'Basic configuration' section is visible. Below it is a 'Target group name' field containing the value 'customer-tg-one'. A note below the field says 'A maximum of 52 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.' Further down are sections for 'Protocol : Port' (HTTP, port 80) and 'IP address type' (IPv4 selected). A note at the bottom says 'Select the VPC that hosts the load balancer. Only VPCs that support the IP address type selected above are available in this list. On the Register targets page, you can specify IP addresses from this VPC or from outside IP addresses located outside of the load balancer's VPC.'