### **Accessing the AWS Management Console**

1. **Start Lab**

### **Task 1: Preparing the Development Environment**

1. **Connect to AWS Cloud9 IDE**
   * From the Services menu, search for and select *Cloud9*.
   * Notice the existing IDE, *Cloud9 Instance*. Click *Open IDE*.
2. **Download and Extract Files**

Run the following command to download the files needed for this lab:  
wgethttps://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/06-lab-containers/code.zip -P /home/ec2-user/environment

Extract the downloaded file:  
unzip code.zip

1. **Run Setup Script**

Set permissions and run the setup script:  
chmod +x ./resources/setup.sh && ./resources/setup.sh

1. **Verify AWS CLI Version**

Check the version of AWS CLI:  
aws --version

1. **Verify Python SDK**

Verify that the SDK for Python is installed:  
pip show boto3

### **Task 2: Analyzing the Existing Application Infrastructure**

1. **Open Coffee Supplier Application**
   * Navigate to the EC2 console in the browser and select *Instances*.
2. **Access the Application**
   * Copy the *Public IPv4 address* of the *AppServerNode* instance and paste it in a new browser tab to view the coffee suppliers website.
3. **Test Web Application Functionality**
   * Add a new supplier by filling in the required fields and submitting.
   * Edit the record and submit changes to verify updates.
4. **Analyze Web Application Code**
   * In AWS Cloud9, expand the *resources* directory and navigate to the *codebase\_partner* directory to review the application code.

### **Task 3: Migrating the Application to a Docker Container**

1. **Create a Working Directory for Node Application**

In AWS Cloud9, run the following commands:  
mkdir containers

cd containers

mkdir node\_app

cd node\_app

mv ~/environment/resources/codebase\_partner ~/environment/containers/node\_app

1. **Create a Dockerfile**

Inside the *node\_app/codebase\_partner* directory, create a Dockerfile:  
cd ~/environment/containers/node\_app/codebase\_partner

touch Dockerfile

Add the following Dockerfile code:  
FROM node:11-alpine

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["npm", "run", "start"]

1. **Build Docker Image**

Build the image with the command:  
docker build --tag node\_app .

1. **Verify Docker Image Creation**

Check the list of available Docker images:  
docker images

1. **Run Docker Container**

Run the container with:  
docker run -d --name node\_app\_1 -p 3000:3000 node\_app

1. **Verify Application in Container**

Verify the application is running with:  
curl http://localhost:3000

1. **Adjust Security Group for Cloud9 Instance**
   * In the EC2 console, modify the inbound rules of the security group to allow traffic on port 3000 from your IP address.
2. **Access Web Interface**
   * Copy the Public IPv4 address of the *aws-cloud9-Cloud9-Instance* EC2 instance and open a browser tab with http://<cloud9-public-ip>:3000 to access the application.

### **Analyze Database Connection Issue**

1. **Database Connection Problem**
   * The container has trouble connecting to the MySQL database on *MysqlServerNode*. The hardcoded IP address in the code is incorrect.
2. **Fix Database Connection**

Run the following command to stop and remove the existing container:  
docker stop node\_app\_1 && docker rm node\_app\_1

1. **Launch New Container with Environment Variable**

Run the container again, passing the correct database IP as an environment variable:  
docker run -d --name node\_app\_1 -p 3000:3000 -e APP\_DB\_HOST="<ip-address>" node\_app

### **Task 4: Migrating MySQL Database to a Docker Container**

1. **Create a mysqldump File:**

Command:  
mysqldump -P 3306 -h <mysql-host-ip-address> -u nodeapp -p --databases COFFEE > ../../my\_sql.sql

* + Replace <mysql-host-ip-address> with the **Public IPv4 address** of MysqlServerNode (found in the EC2 console).

1. **Edit the mysqldump File:**
   * Open my\_sql.sql in the AWS Cloud9 editor.
   * Change a value in the INSERT INTO statement (e.g., update the street name).
2. **Create Dockerfile:**

Directory setup:  
cd /home/ec2-user/environment/containers

mkdir mysql

cd mysql

mv ../my\_sql.sql .

Dockerfile contents:  
FROM mysql:8.0.23

COPY ./my\_sql.sql /

EXPOSE 3306

**Free Up Disk Space:**docker rmi -f $(docker image ls -a -q)

sudo docker image prune -f && sudo docker container prune -f

**Build the Docker Image:**docker build --tag mysql\_server .

docker images

**Run MySQL Container:**docker run --name mysql\_1 -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=rootpw -d mysql\_server

**Import Data and Add User:**docker exec -i mysql\_1 mysql -u root -prootpw < my\_sql.sql

docker exec -i mysql\_1 mysql -u root -prootpw -e \

"CREATE USER 'nodeapp' IDENTIFIED WITH mysql\_native\_password BY 'coffee'; GRANT all privileges on \*.\* to 'nodeapp'@'%';"

### **Task 5: Testing the MySQL Container with the Node Application**

**Stop and Remove Old Node Application Container:**docker stop node\_app\_1 && docker rm node\_app\_1

**Get mysql\_1 Container IPv4 Address:**docker inspect network bridge

**Start Node Application with Updated Database Host:**docker run -d --name node\_app\_1 -p 3000:3000 -e APP\_DB\_HOST=<mysql\_1-IP-address> node\_app

**Verify:**

Visit http://<cloud9-public-ip-address>:3000 and check for updated data.

### **Task 6: Adding Docker Images to Amazon ECR**

**Authorize Docker with Amazon ECR:**aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com

**Create Amazon ECR Repository:**aws ecr create-repository --repository-name node-app

**Tag Docker Image:**docker tag node\_app:latest <registry-id>.dkr.ecr.us-east-1.amazonaws.com/node-app:latest

**Push Image to Amazon ECR:**docker push <registry-id>.dkr.ecr.us-east-1.amazonaws.com/node-app:latest

**Verify Uploaded Image:**aws ecr list-images --repository-name node-app

### **Completion**

* Test application to confirm functionality.
* Submit and end the lab.