**Aim** :- Caching application data with ElastiCache.

**Lab overview and objectives**

In this lab, you will deploy an Amazon ElastiCache cluster. You will also test synchronizing the cache with the Amazon Aurora Serverless database by using Python scripts. Finally, you will update the coffee suppliers application with new Node.js code that will use data caching.

After completing this lab, you should be able to:

* Create a new ElastiCache for Memcached cluster
* Query the ElastiCache for Memcached cluster by using Python and the pymemcache client

**AWS service restrictions**

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

**Scenario**

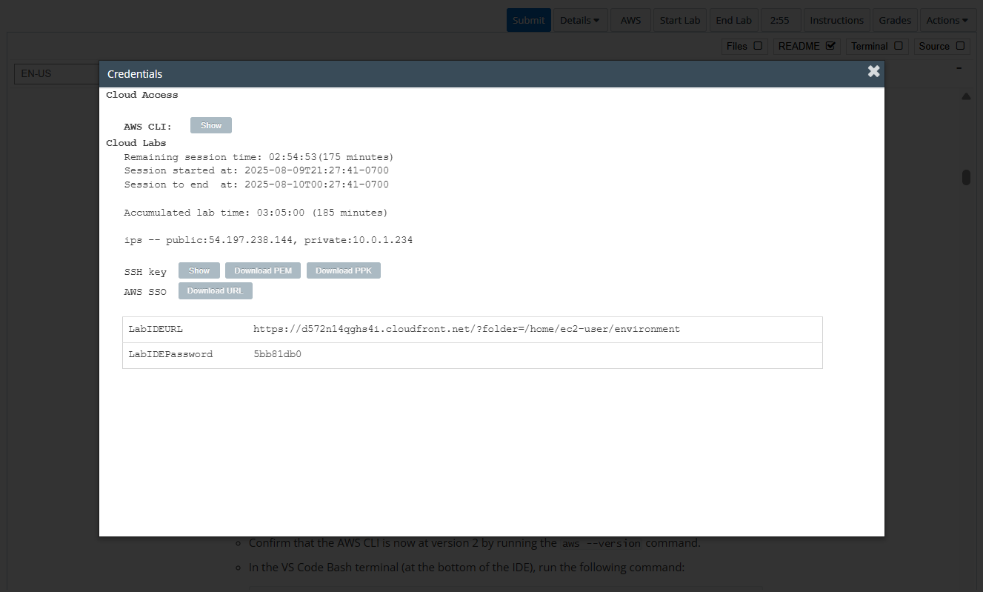
Frank and Martha are excited that the bean inventory from the coffee suppliers application is integrated into the café website. Unfortunately, they are getting occasional negative feedback from customers who say that the bean inventory information takes too long to display on the page. Frank and Martha have asked Sofía if she can have the site display the information more quickly.

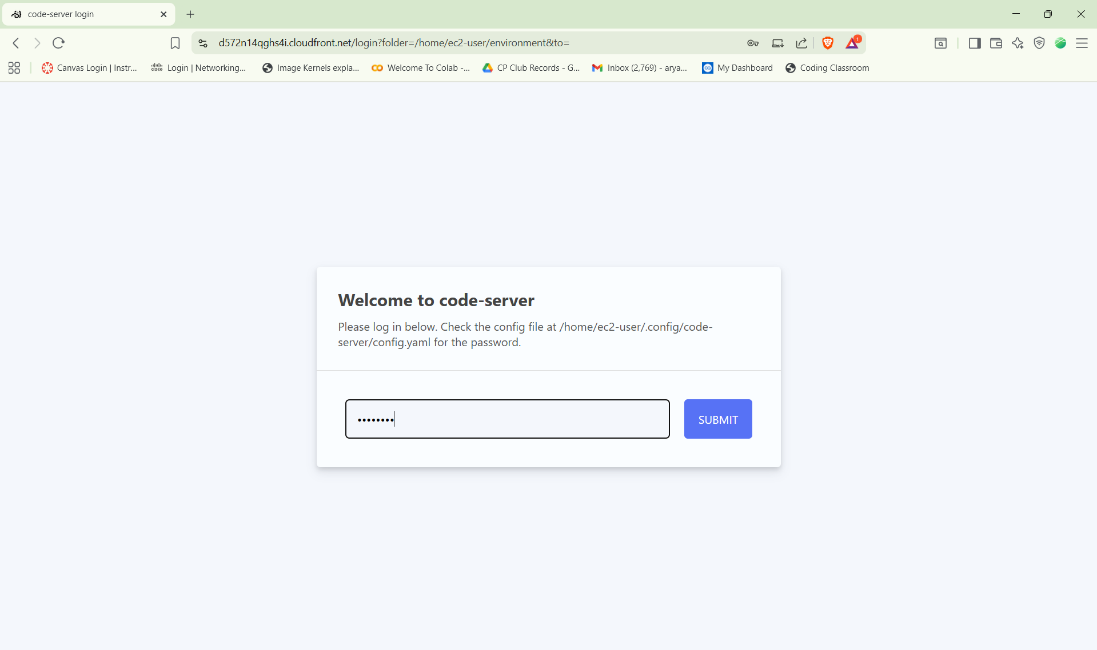
Luckily, one of the café's customers, Olivia, is an AWS consultant and has a lot of database experience. She has offered to help with the website. Olivia proposed adding database caching to improve the speed of the database query. This way, frequently used data would be stored in memory instead of having to be retrieved from the database, which could also require information to be read from database storage.

Olivia explained that you could cache data locally on the Amazon Elastic Compute Cloud (Amazon EC2) instance that is hosting the application containers. However, that architecture would not provide fault tolerance. This strategy would also require more administration because you would need to install and maintain the caching software yourself. Olivia recommended using the Amazon ElastiCache managed service. ElastiCache is similar to Aurora Serverless in that it simplifies the process to deploy and maintain a cache cluster.

**Task 1: Preparing the lab**

Connect to the VS Code IDE.

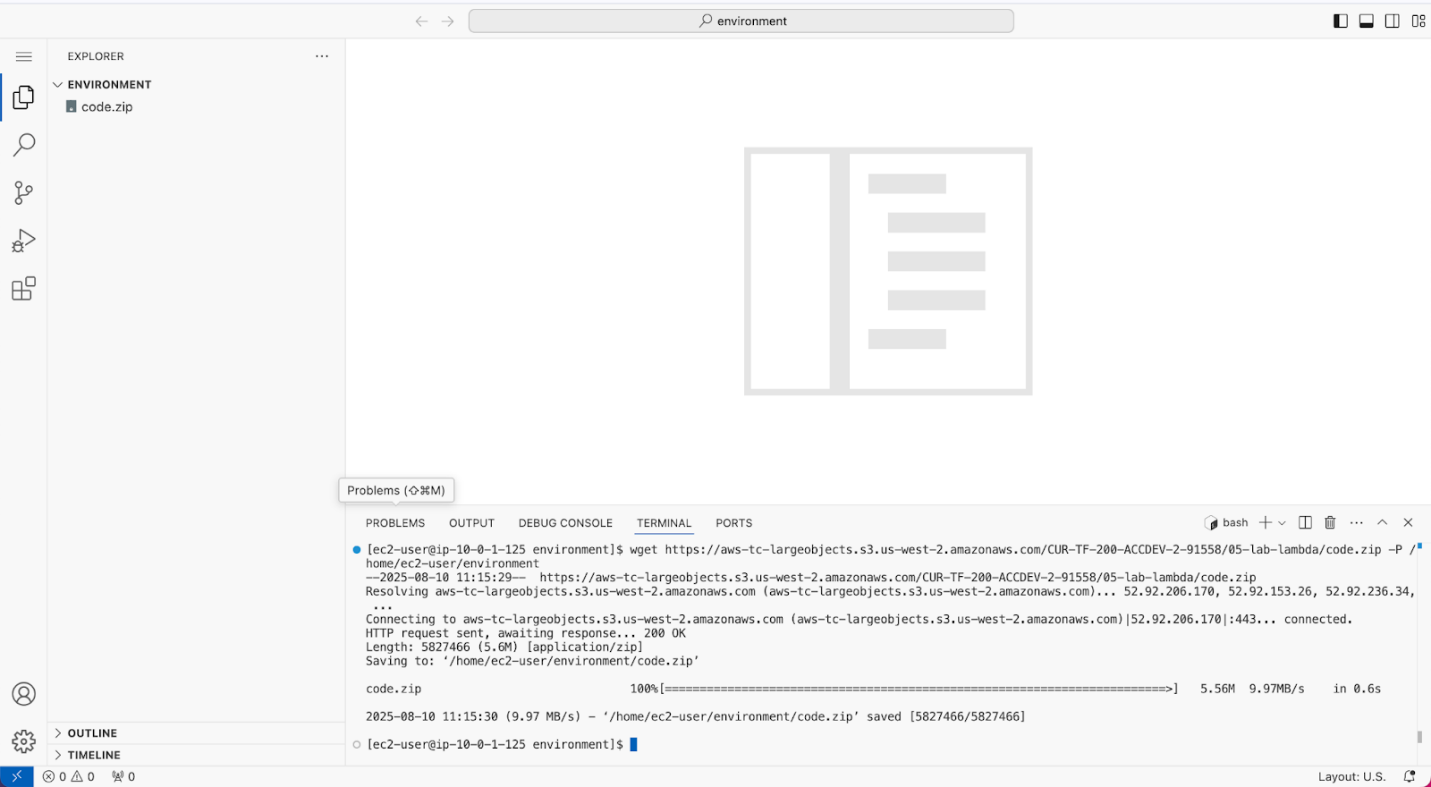
1. At the top of these instructions, choose Details followed by **AWS: Show**
2. Copy values from the table **similar** to the following and paste it into an editor of your choice for use later.
   1. **LabIDEURL**
   2. **LabIDEPassword**
3. In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
4. On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.

****

1. Download and extract the files that you need for this lab.

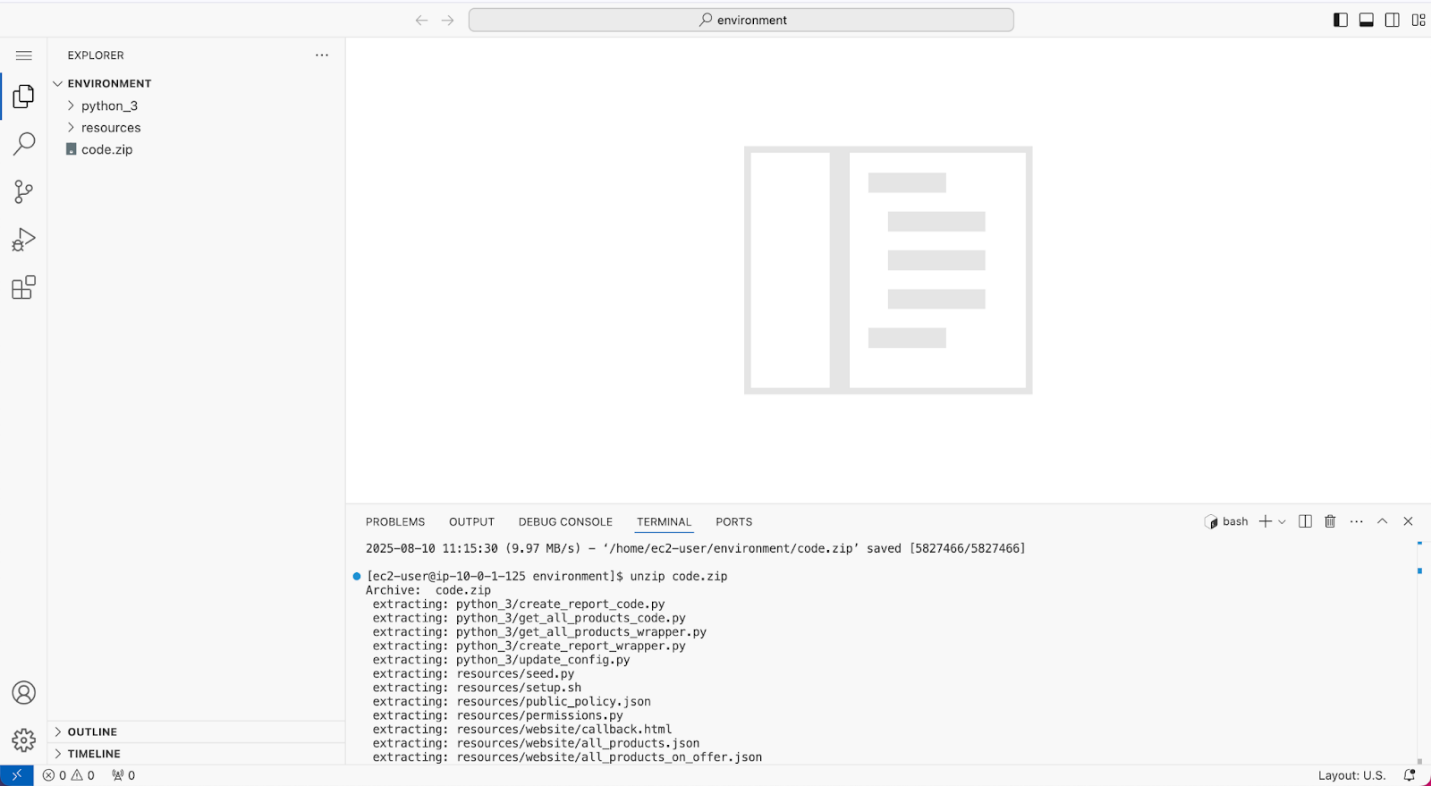
* In the VS Code bash terminal (located at the bottom of the IDE), run the following commands:

**wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/08-lab-db-caching/code.zip -P /home/ec2-user/environment**



1. You should see that the **code.zip** file was downloaded to the VS Code IDE and is now in the left navigation pane.

* Extract the file by running the following command:

unzip code.zip

1. Run a script that upgrades the version of the AWS CLI installed on the VS Code IDE.

* To set permissions on the script and then run it, run the following commands in the Bash terminal:

chmod +x ./resources/setup.sh && ./resources/setup.sh

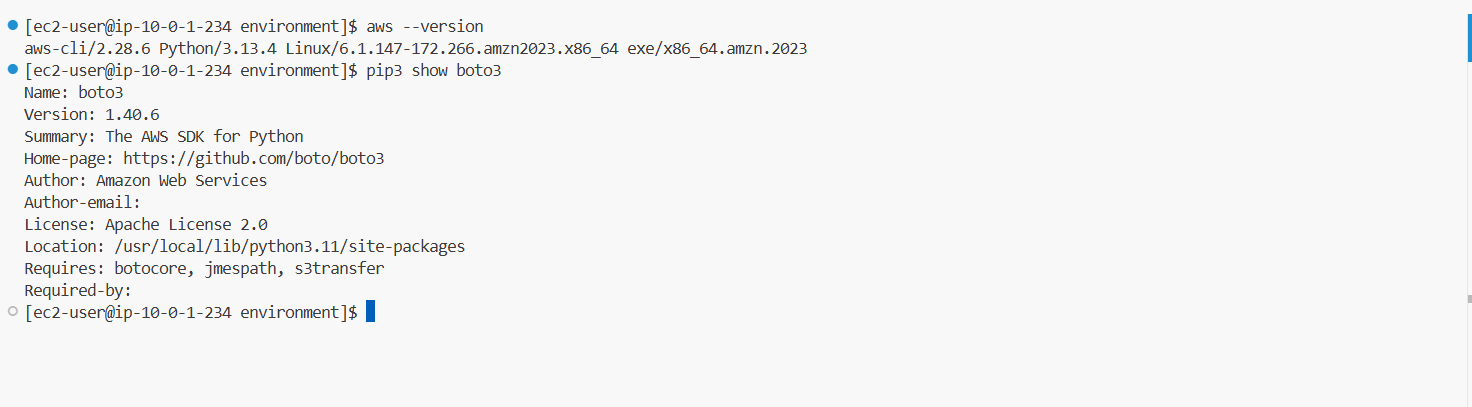
The script will prompt you for the **IP address** by which your computer is known to the internet.

Use [www.whatismyip.com](http://www.whatismyip.com/) to discover this address and then paste the IPv4 address into the command prompt and finish running the script.



1. Verify the AWS CLI version and also verify that the SDK for Python is installed.

* Confirm that the AWS CLI is now at version 2 by running the **aws --version** command.
* In the VS Code Bash terminal (at the bottom of the IDE), run the following command:

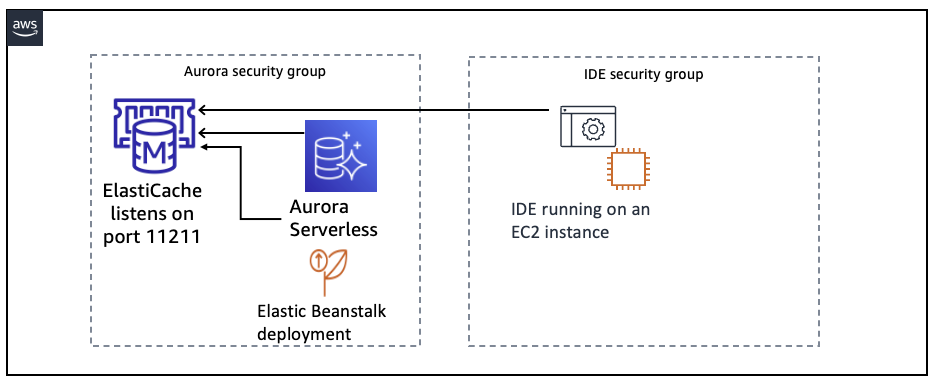
**pip3 show boto3**

**Task 2: Configuring the subnets for ElastiCache to use**

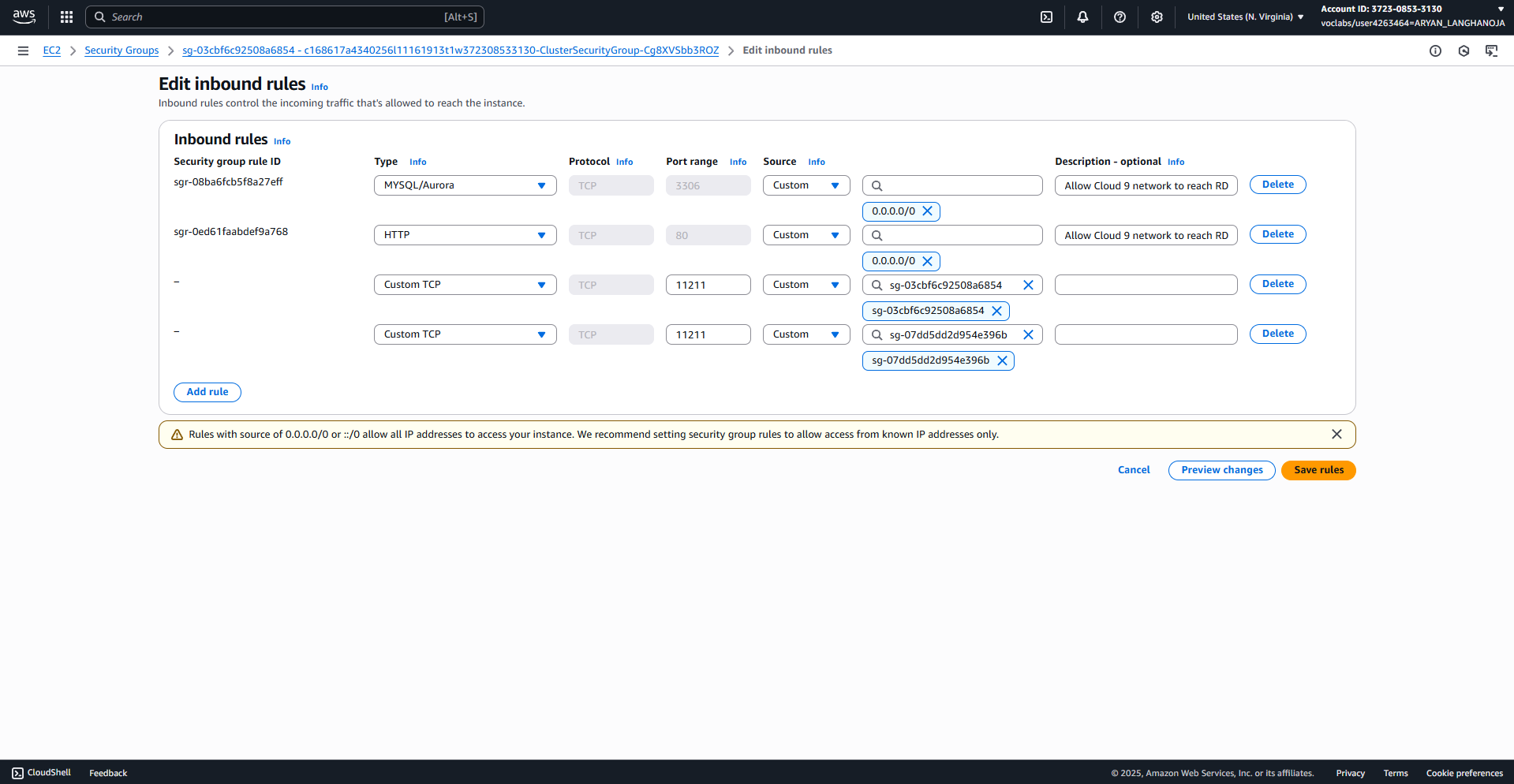
The Memcached database must be able to communicate with the Aurora Serverless database. In addition, the application needs to be able to access both the Memcached database and the Aurora Serverless database.

In this task, you will act as Olivia to configure a security group to enable the necessary communication between the application, database, and cache. You will use the security group that the Aurora Serverless cluster is already using rather than creating an additional security group.

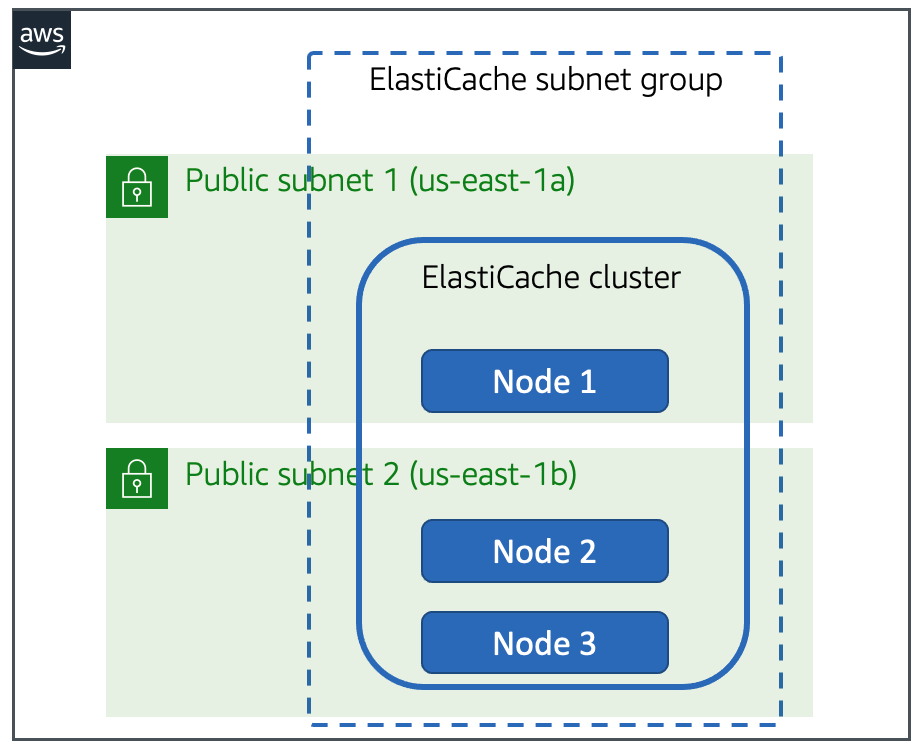
1. Configure the security group that the Aurora Serverless instance will use, as shown in the following diagram.



* Return to the AWS Management Console browser tab.
* From the **Services** menu, choose **EC2**.
* In the left navigation pane, choose **Security Groups**.
* Select the checkbox for the security group that has *Aurora* in the name.
* In the lower pane, choose the **Inbound rules** tab.
* Choose **Edit inbound rules**.
  + Add an entry to enable access to the Memcached port from all sources that are assigned to the *Aurora* security group:
    - Choose **Add rule**
    - **Type:** Choose **Custom TCP**
    - **Port range:** Enter 11211
    - **Source:** Choose the security group with *Aurora* in the name. This is the same group that you are currently editing.
* Add an entry to enable access to the Memcached port from all sources that are assigned to the *Lab IDE SG* security group:
  + Choose **Add rule**
  + **Type:** Choose **Custom TCP**
  + **Port range:** Enter 11211
  + **Source:** Choose the security group with *Lab IDE SG* in the name
* Choose **Save rules**.

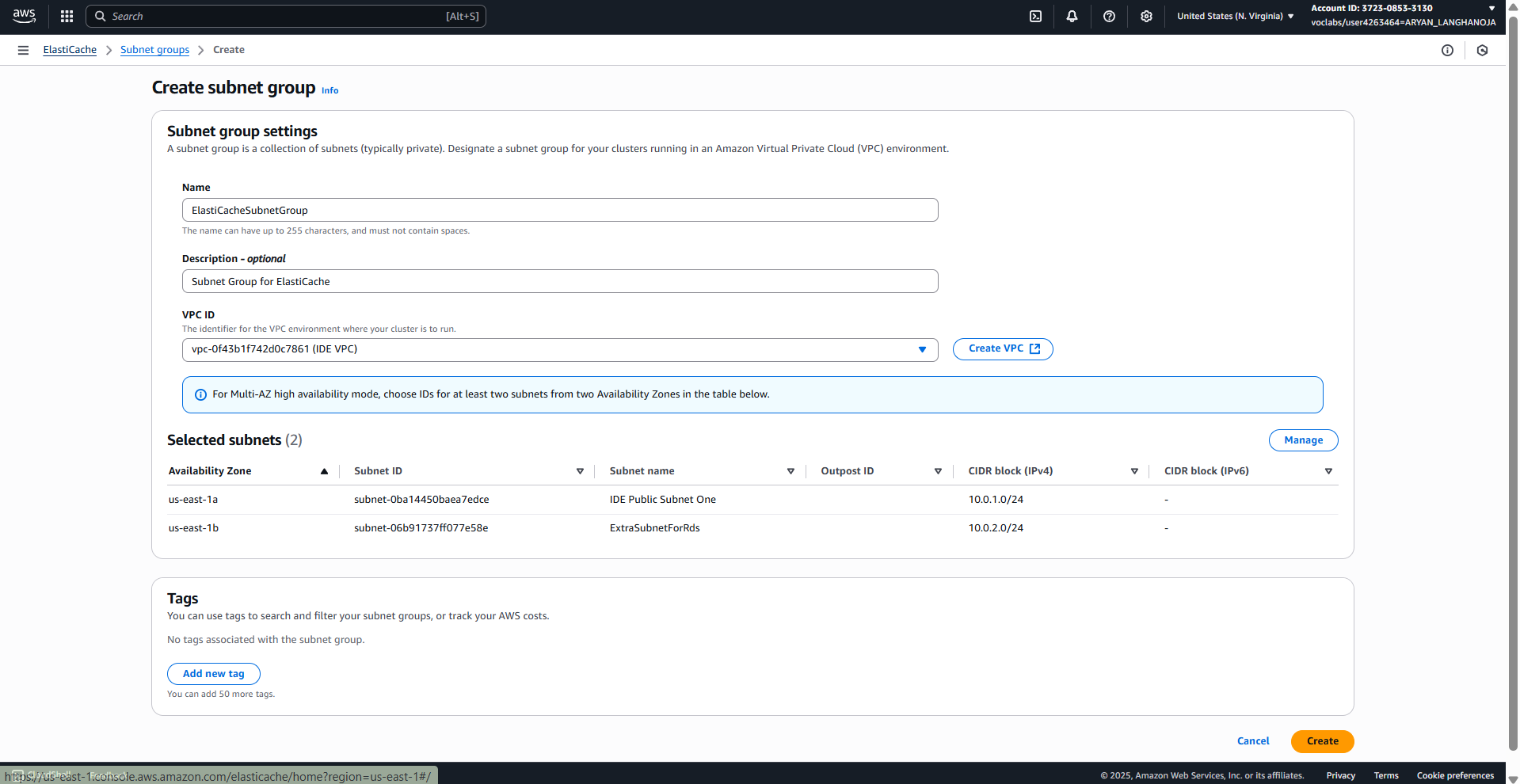


1. Configure the database subnet group that the Aurora Serverless instance will use, as shown in the following diagram.



* + In the search box at the top of the AWS management console, type and choose ElastiCache.
  + From the left navigation pane, choose **Subnet groups**.
  + Choose **Create subnet group**.
    - Configure the following settings:
      * **Name:** Enter ElastiCacheSubnetGroup
      * **Description:** Enter Subnet Group for ElastiCache
      * **VPC ID:** From the dropdown, choose **IDE VPC**
      * Notice that two subnets are already chosen for the cluster **Selected subnets (2)**
        + If you do not see this, choose **Manage** button on the right and select two subnets.
      * Choose **Create**.
        + Message is displayed stating that *The subnet group was created successfully.*

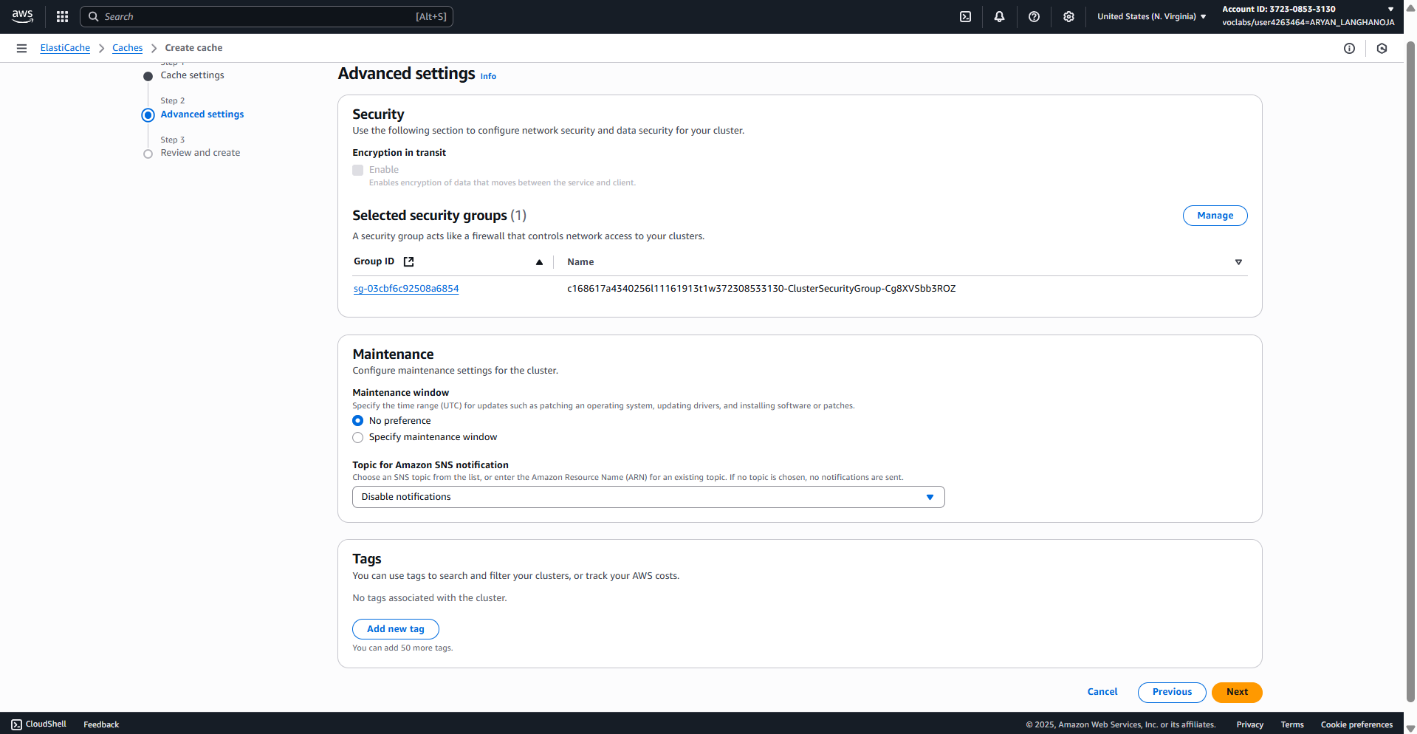
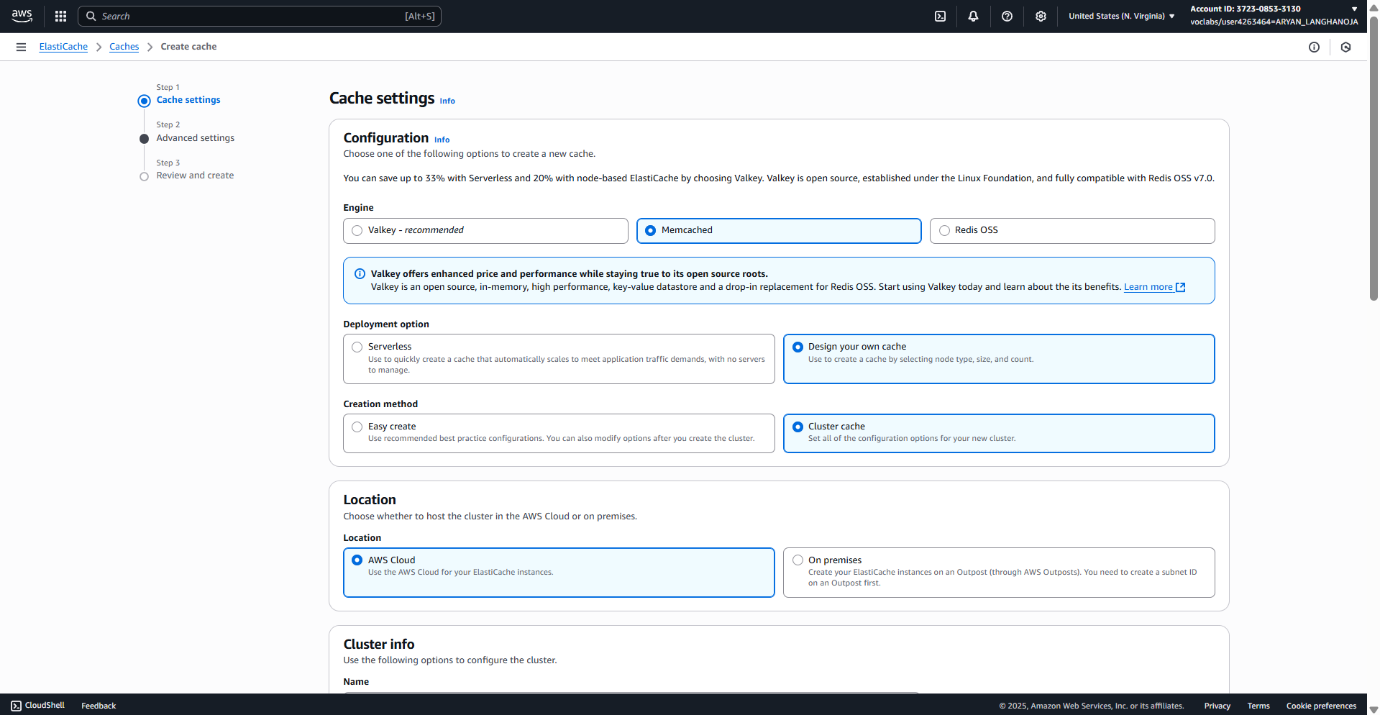
Next, you will deploy an ElastiCache cluster.

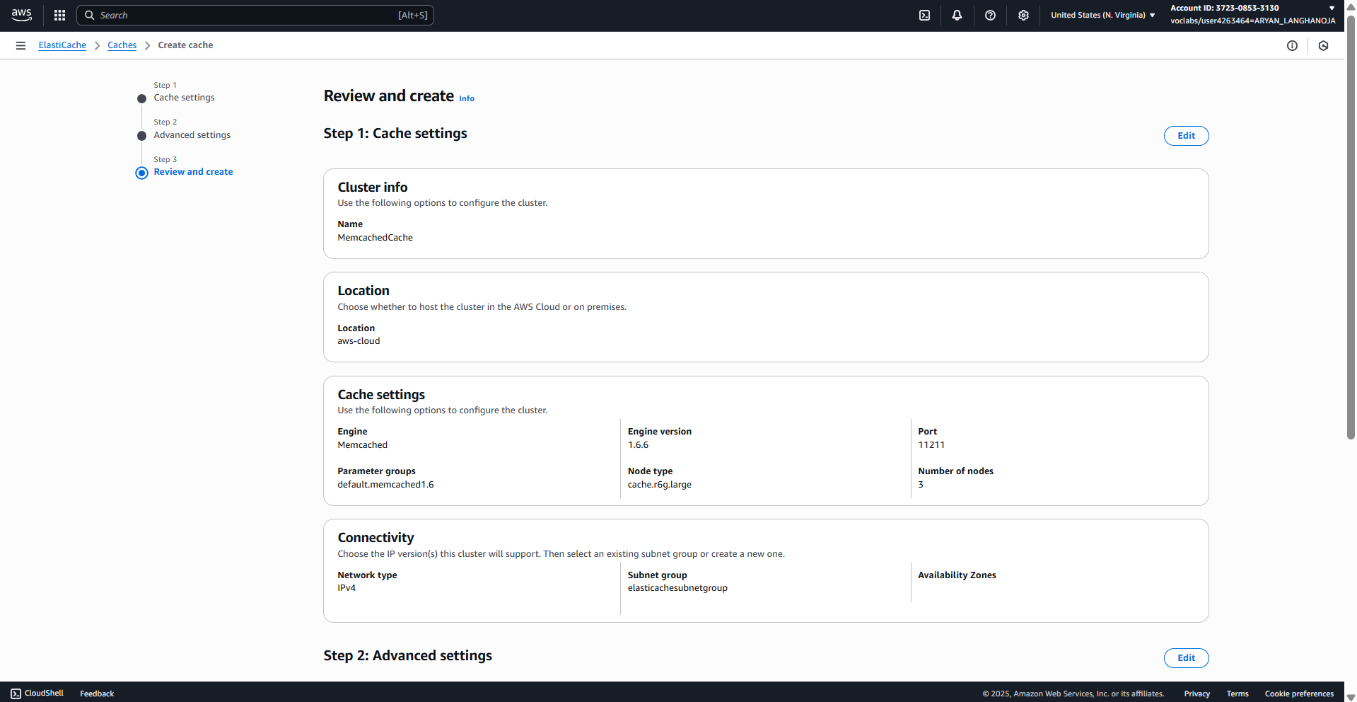


**Task 3: Creating the ElastiCache cluster**

As with the database deployment in a previous lab, the café has chosen to deploy a managed service instead of manually installing software in a container or on an EC2 instance. ElastiCache offers two engine options, Memcached and Redis. Olivia decided to use Memcached for the coffee suppliers application because the application does not require advanced features that Redis offers. Memcached will be a simple way to get started with caching, and the café could move to Redis later if needed.

In this task, you will again act as Olivia to create an ElastiCache for Memcached cluster.

1. Create an ElastiCache for Memcached instance.
   * Remain in the ElastiCache console.
   * In the left navigation pane, choose **Memcached caches**.
   * Choose **Create Memcached cache**.
   * Configure the ElastiCache cluster with the following settings:
     + In the **Choose a cluster creation method** section:
       - Choose: **Design your own cache**.
       - Choose: **Standard create**.
       - In the **Location** section, choose **AWS Cloud**.
     + In the **Cluster info** section:
       - **Name:** Enter MemcachedCache
     + In the **Cluster settings** section:
       - **Engine version:** Choose **1.6.6**
       - **Port:** Enter 11211 (If not already mentioned).
       - **Parameter group:** Choose **default.memcached.1.6**
       - **Node type:** Choose **cache.r6g.large (13.07GiB)**
       - **Number of nodes:** Enter 3
     + Leave the **Connectivity** section at default values.
     + Choose **Next**.
     + On the **Advanced settings** page
       - For **Selected security groups (0)** choose **Manage**.
       - Select the Security group having *ClusterSecurityGroup* in the name.
     + Choose **Next** .
     + ****Choose **Create**.

****

**Task 4: Loading records into the cache from the database**

1. To change to the directory that holds the proof-of-concept scripts, return to the VS Code IDE Bash terminal, and run the following command:

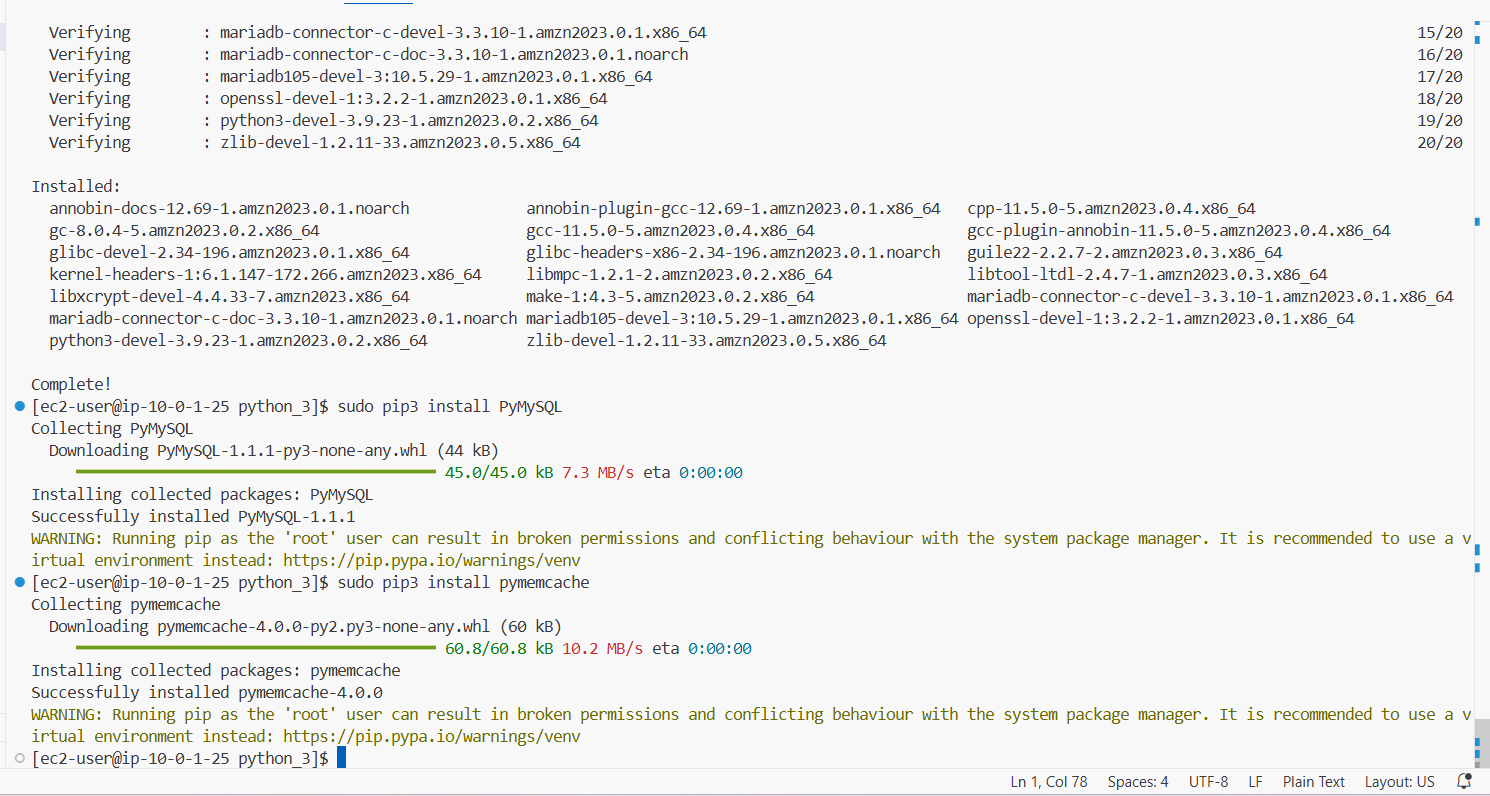
**cd ~/environment/python\_3**

1. To install the libraries that Python needs to interact with the database and the cache, run the following commands:

**sudo dnf install -y mariadb105-devel gcc python3-devel**

**sudo pip3 install PyMySQL**

**sudo pip3 install pymemcache**

In the scripts, you will need to define the connection to both the Aurora database and the Memcached cache. In the following steps, you find the endpoints for these resources.

1. Find the ElastiCache for Memcached endpoint.
   * In the VS Code IDE Bash terminal, run the following command:

**aws elasticache describe-cache-clusters**

1. Find the Aurora Serverless cluster endpoint.

* In the VS Code IDE Bash terminal, run the following command:

**aws rds describe-db-cluster-endpoints**

1. Update the script to define the database and cache connections.
   * In the *find\_all.py* file, replace *<FMI\_1>* with the ElastiCache endpoint address.

The updated code should be similar to the following:

**memcached\_client = base.Client(('memcachedcache.xxxxxxx.cfg.use1.cache.amazonaws.com', 11211))**

* + Replace *<FMI\_2>* with the Aurora endpoint.

The updated code should be similar to the following:

mydb = pymysql.connect(

"supplierdb.cluster-xxxxxxxxxxx.us-east-1.rds.amazonaws.com",

"nodeapp",

"coffee",

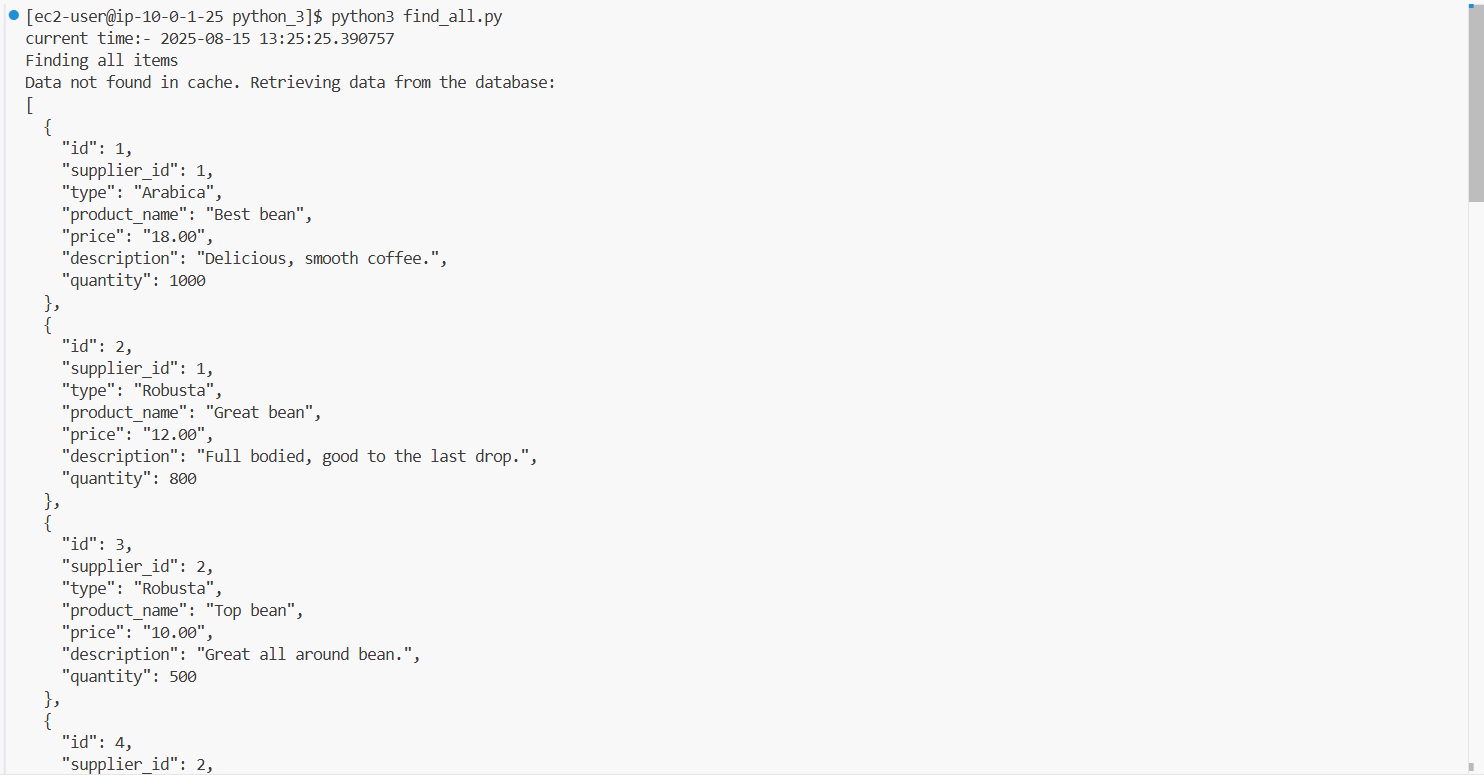
"COFFEE"

)

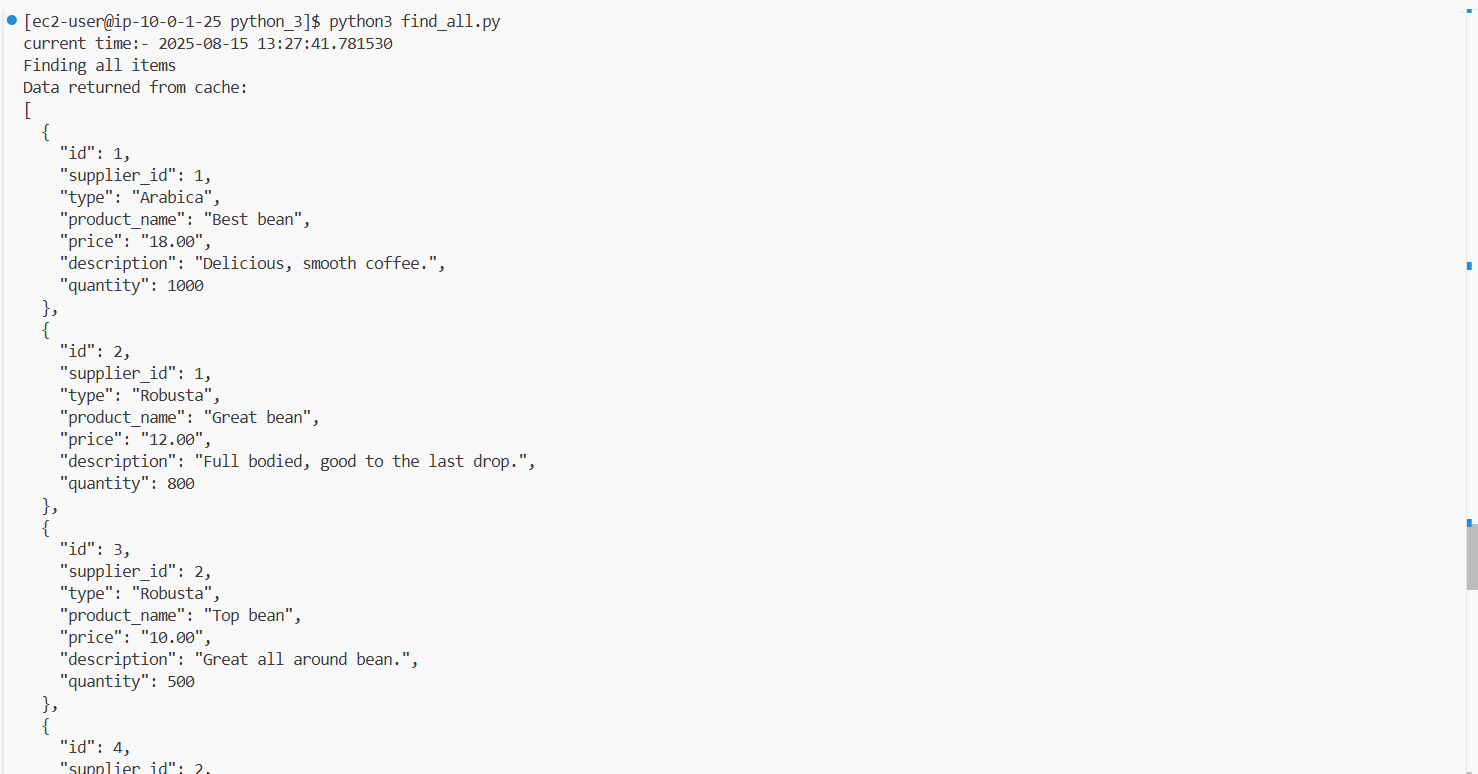
* + From the navigation pane, choose menu, then choose **File > Save**.

1. Test the *find\_all.py* script.

* First, ensure that the status of the Memcached cluster is *available* before you continue.
* In the VS Code IDE Bash terminal, run the following command:

**python3 find\_all.py**

* In the VS Code IDE Bash terminal, run the command again:

**python3 find\_all.py**

**Task 5: Updating data**

1. Test the *update\_item.py* script.

* Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
* Run the *update.py* script:

**python3 update\_item.py**

1. Now, to test the cache refresh.

* Run the *find\_all.py* script:

**python3 find\_all.py**



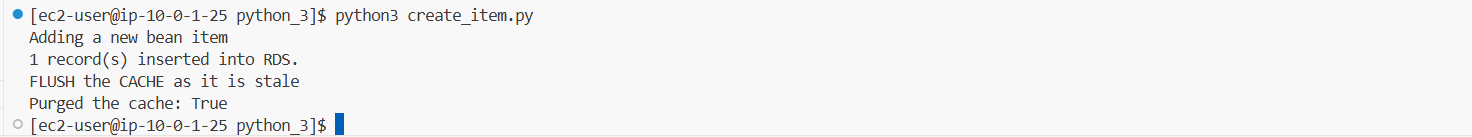
**Task 6: Creating a new record**

In this task, you will create a new item in the database. You will use the *write-through* caching strategy to prevent the cache from becoming stale.

1. Test the *create\_item.py* script.

* Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
* Run the *update.py* script:

**python3 create\_item.py**



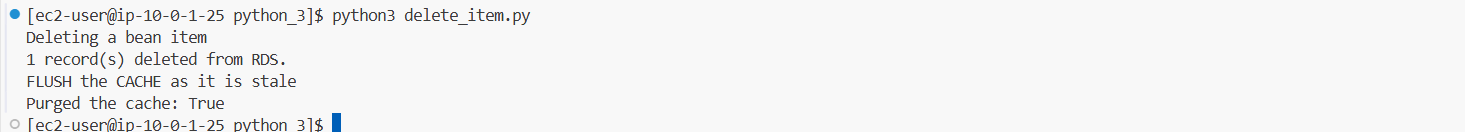
**Task 7: Deleting a record**

The final proof-of-concept script tests deleting an item from the database. You will again use the *write-through* strategy to ensure the cache stays in sync with the database.

Test the *delete\_item.py* script.

* Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
* Run the *delete\_item.py* script:

**python3 delete\_item.py**



**Task 8: Updating the application container**

Nikhil quickly updated the coffee suppliers application. His code expanded on the examples from the proof of concept. He restructured the code, and now rather than having a script for each type of operation (SELECT, INSERT, UPDATE, DELETE), one script handles all database calls, and another interacts with the cache. He also modified the code to use an environment variable to identify the ElastiCache for Memcached endpoint.

You will perform the following steps as Nikhil to review the application code that interacts with the cache. Then, you will update the application.

1. To update the application code, run the following command:

**mv ~/environment/resources/bean.controller\_2.js \**

**~/environment/resources/codebase\_partner/app/controller/bean.controller.js**



1. To rebuild and update the Docker image, run the following commands:

**cd ~/environment/resources/codebase\_partner/**

**npm install**

**docker build --tag node\_app .**



1. Authorize your Docker client to connect to the Amazon Elastic Container Registry (Amazon ECR) service.

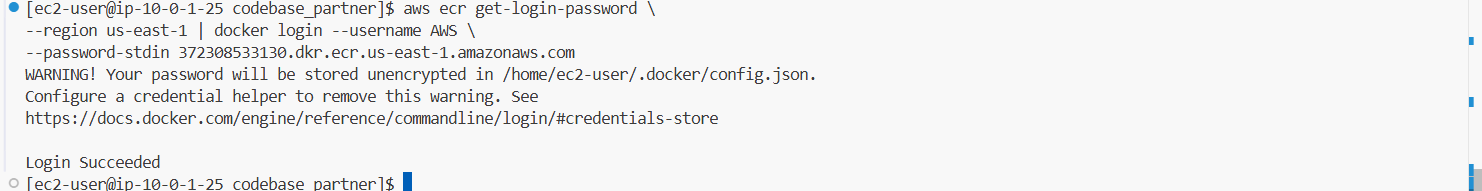
* Discover your AWS account ID:
* In the AWS Management Console, in the upper-right corner, choose your user name, which begins with **voclabs/user**.
* Copy the **My Account** value from the menu. This is your AWS account ID.
* Return to the VS Code IDE Bash terminal.
* To authorize your VS Code IDE Docker client, run the following command. Replace with the actual account ID that you just found in the console:

**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 indicates that the login succeeded.



1. Push the updated container image to Amazon ECR.

* To find the URI for the repository, run the following command:

**aws ecr describe-repositories --query repositories[][repositoryUri] --output text**



* Next, to tag the Amazon ECR image, run the following command. Replace *<FMI\_1>* with the repository URI:

**docker tag node\_app:latest xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app:latest**



* Now, to push the image to Amazon ECR, run the following command. Again, replace *<FMI\_1>* with the repository URI:

**docker push 372308533130.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app:latest**

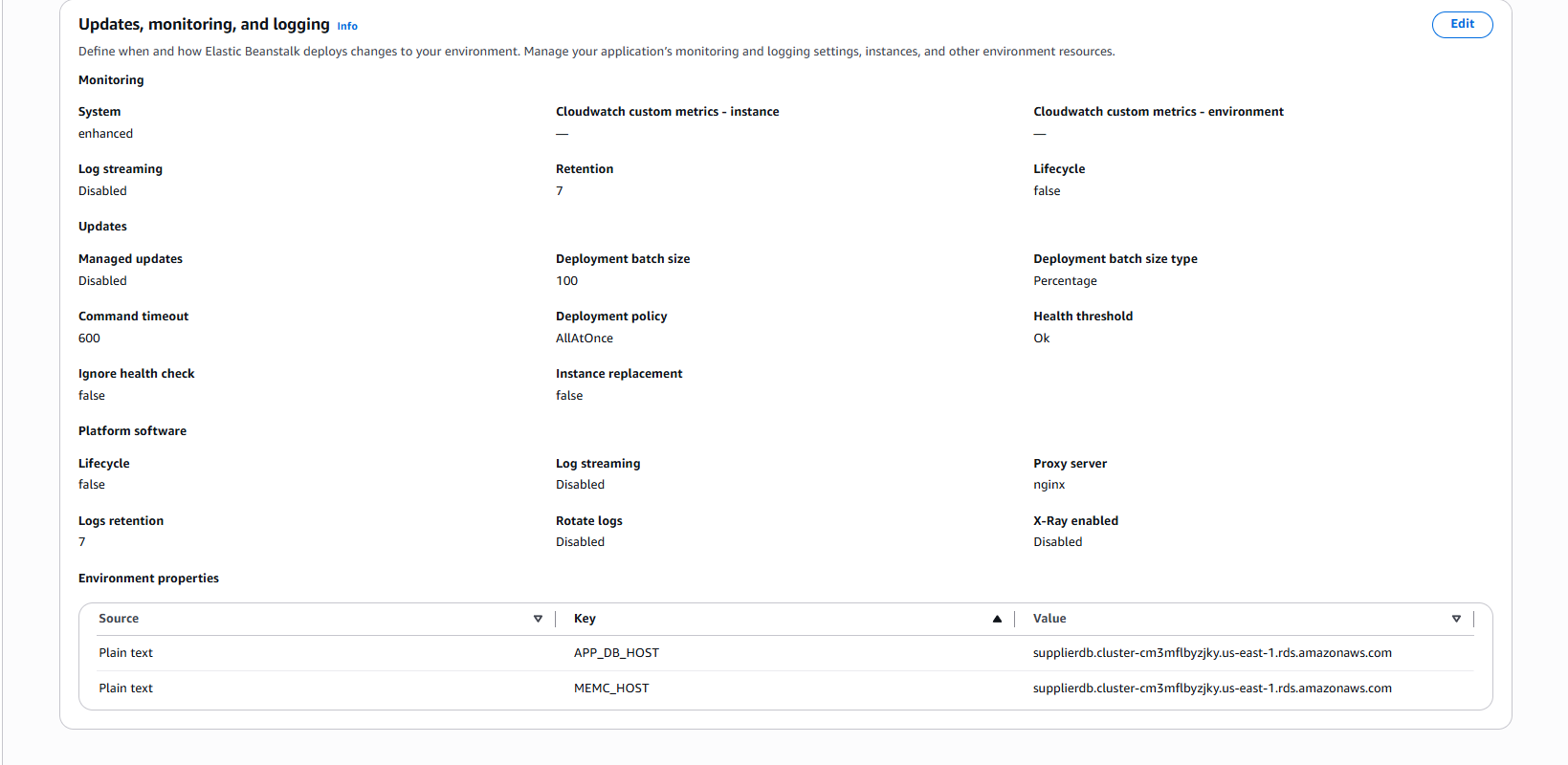
1. Configure the new Memcached environment variable, *MEMC\_HOST*.
   * Return to the browser tab with the **MyEnv** page.
   * In the left navigation pane, choose **Configuration**.
   * In the **Updates, monitoring, and logging** category, choose **Edit**.
   * Scroll down to the **Environment properties** section, and add **Add environment property** as follows:
     + **Name:** Enter MEMC\_HOST
     + **Value:** Enter the same Memcached endpoint that you used when testing the Python scripts
   * Choose **Apply**.

A message indicates that *Elastic Beanstalk is updating your environment.*

It takes a few minutes for the environment to be updated. Once the build has finished, you can continue.

* + Revisit the browser tab with the coffee suppliers application, and refresh the page.

A note now appears under **All beans** to indicate when the results were returned from the database and when they were returned from the cache.



**Conclusion:-**

In this lab,

* I deployed the coffee suppliers app on AWS using Aurora Serverless for the database and Elastic Beanstalk for the web app.
* I downloaded the code,
* upgraded AWS CLI,
* set up VPC subnets,
* created the Aurora DB, and
* connected it to the Docker container.
* I created database tables, loaded sample data, and reviewed the ECR image.
* Then, I deployed the app to Elastic Beanstalk using Dockerrun.aws.json and linked it with API Gateway.
* The app now runs successfully with managed services, scalable, and accessible via an API endpoint.

**Result :-**

