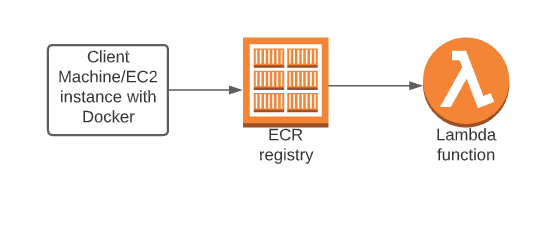
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Declaration** |  | | | | |
| Questions in this exercise are intentionally complex and could be convoluted or confusing. This is by design and to simulate real life situations where customers seldom give crystal clear requirements and ask unambiguous questions. | | | | | |
|
|
| I have read the above statement and agree to these conditions | | | | | |
| I AGREE | Sachin Kumar Jha | | | | |
| <Enter your name above this line to indicate that you are in agreement> | | | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| **Instructions** |  |  |  |  |  |
| Every screenshot requested in this workbook is compulsory and carries 1 point | | | | | |
| Your AWS account ID must be clearly visible in every screenshot using the AWS console; missing id or using someone else's id is not permitted. Such cases will be considered as plagiarism and severe penalty will be imposed. | | | | | |
| All screenshots must be in the order mentioned under "Expected Screenshots" for every step | | | | | |
| DO NOT WAIT UNTIL THE LAST MINUTE. The program office will not extend the project submission deadline under any circumstances. | | | | | |
| The file should be renamed in the format BATCH\_FIRSTNAME\_LASTNAME\_PROJECT1.  For example: PGPCCMAY18\_VIJAY\_DWIVEDI\_PROJECT1.pdf | | | | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| **Resource Clean Up** | |  |  |  |  |
| Cloud is always pay per use model and all resources/services that we consume are chargeable. Cleaning up when you’ve completed your lab or project is always necessary. This is true whether you’re doing a lab or implementing a project at your workplace. | | | | | |
| After completing the lab, make sure to delete each resource created in reverse chronological order. | | | | | |

**Scenario**

The introduction of Lambda support for OCI container images provides customers with more choices when it comes to packaging formats. Developers can now choose to take advantage of the event-driven runtime model and cost-savings advantages of AWS Lambda, while taking advantage of the predictability and control offered by a container-based development and deployment cycle.

**Architecture diagram**

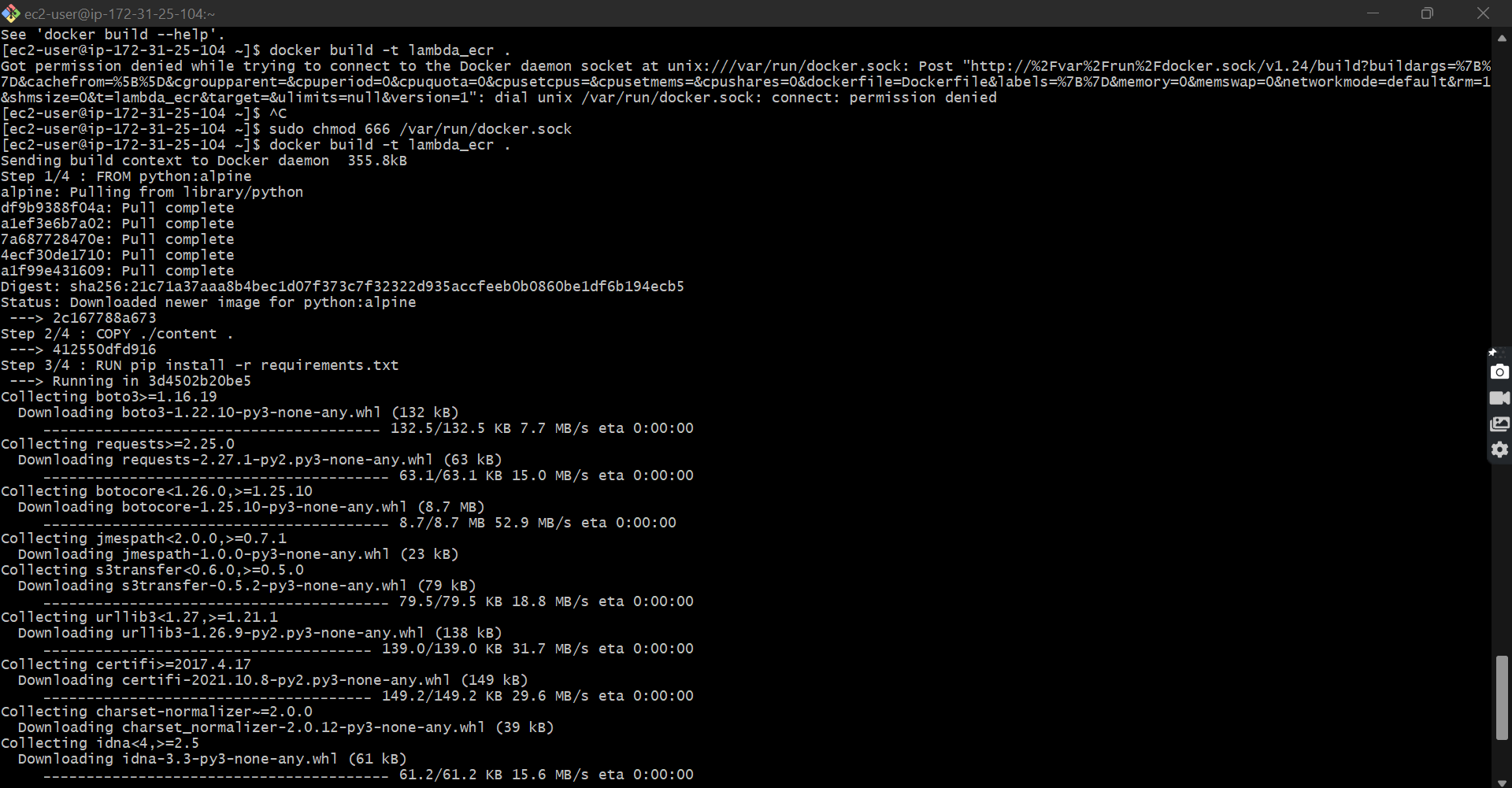
****

|  |  |
| --- | --- |
| **Architecture Implementation** | |
| 1 | Download the Dockerfile and the app code folder provided with this workbook |
| 2 | Package the web application as a Docker image running on Alpine with Python |
| 3 | Create an ECR repository and login to it. |
| 4 | Build the image with the downloaded dockerfile and the support files |
| 5 | Tag the image appropriately and push it into the ECR repository. |
| 6 | Create a Lambda function with the image in ECR. |

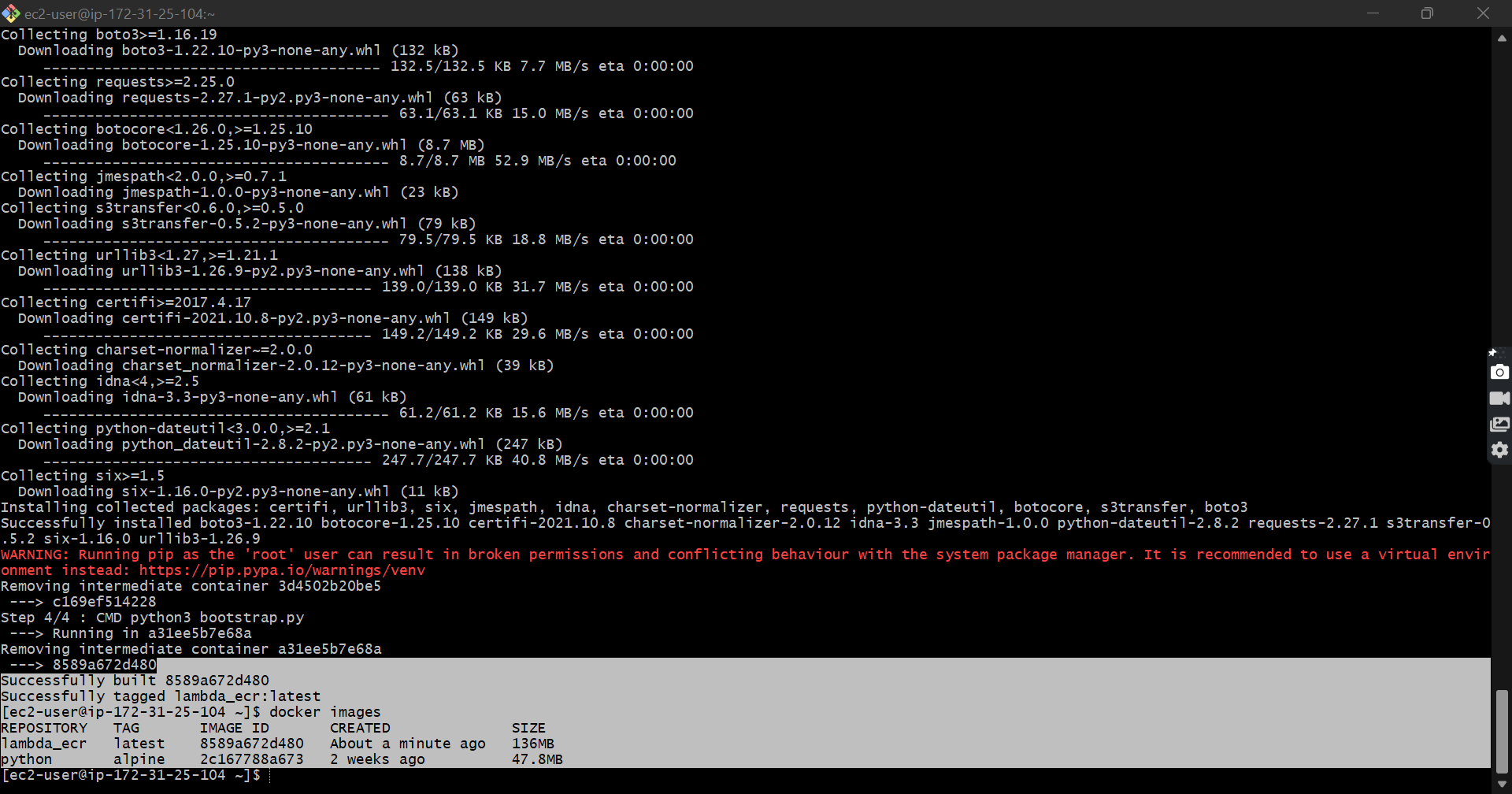
**Step 1 : Docker Image creation**

|  |  |
| --- | --- |
| Step number | a |
| Step name | Creation of Docker image |
| Instructions | 1. Create an EC2 instance using the Amazon Linux 2 AMI in the default VPC. 2. Attach the role “LabInstanceProfile” to the instance created above 3. Download the file OCI.zip provided with this workbook and copy it to the EC2 instance using the scp command scp -i project3.pem ./OCI.zip [ec2-user@ec2-54-218-14-230.us-west-2.compute.amazonaws.com](mailto:ec2-user@ec2-54-218-14-230.us-west-2.compute.amazonaws.com):/home/ec2-user (Ensure that the file OCI.zip and the pem file are in the same folder before running this command) 4. Login to the instance using SSH and run the following commands to set up the environment *sudo yum update sudo yum install unzip sudo unzip OCI.zip sudo amazon-linux-extras install docker sudo service docker start sudo usermod -a -G docker ec2-user* **(At this point, log out of the instance and log in again to ensure that the above command works. Then continue with the rest of the commands)** *sudo yum install awscli -y aws configure*   **Skip the access key and secret access key fields by pressing the Enter key. Enter the region as**  **us-east-1 and format as json**   1. Run the below command to create the Docker image *docker build -t lambda\_ecr .* 2. Run the below command to verify the creation of the image *docker images* |
| Expected screenshots | 1 )Building the Docker image 3) List of the created image |

**<Insert screenshot for a(1) here>**

****

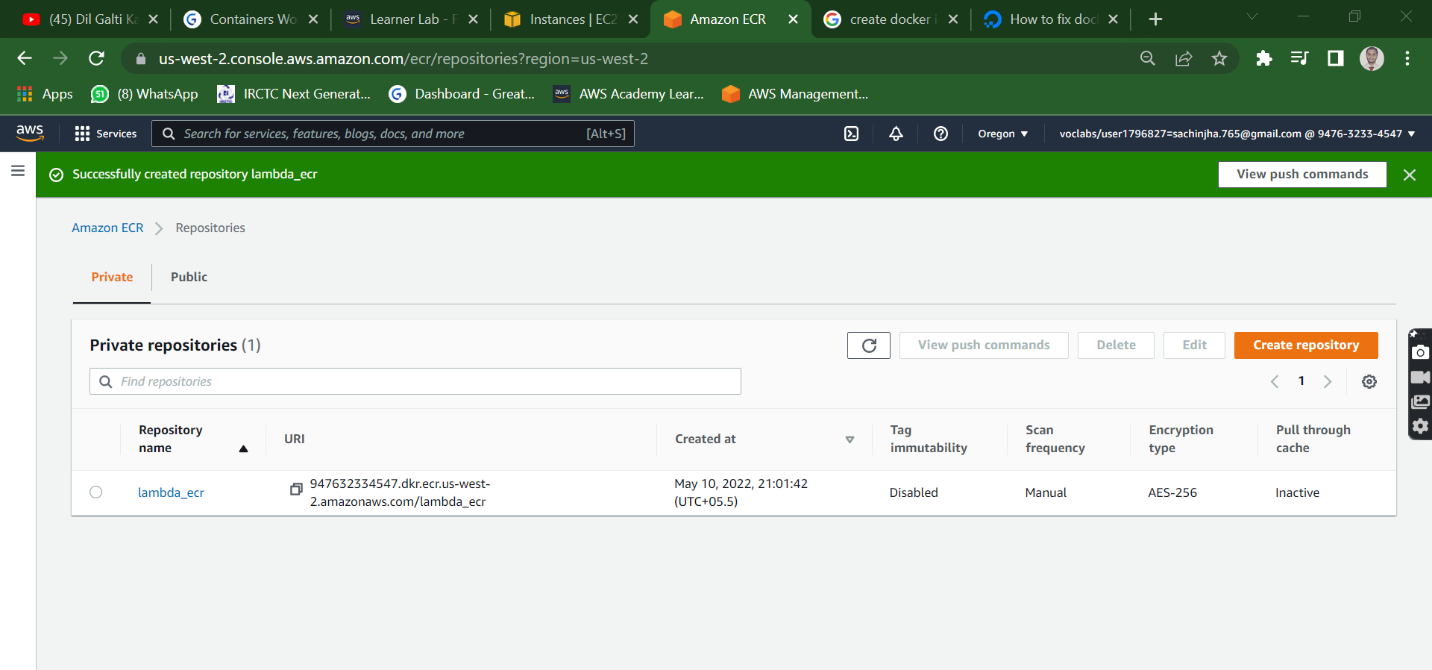
**<Insert screenshot for a(2) here>**

****

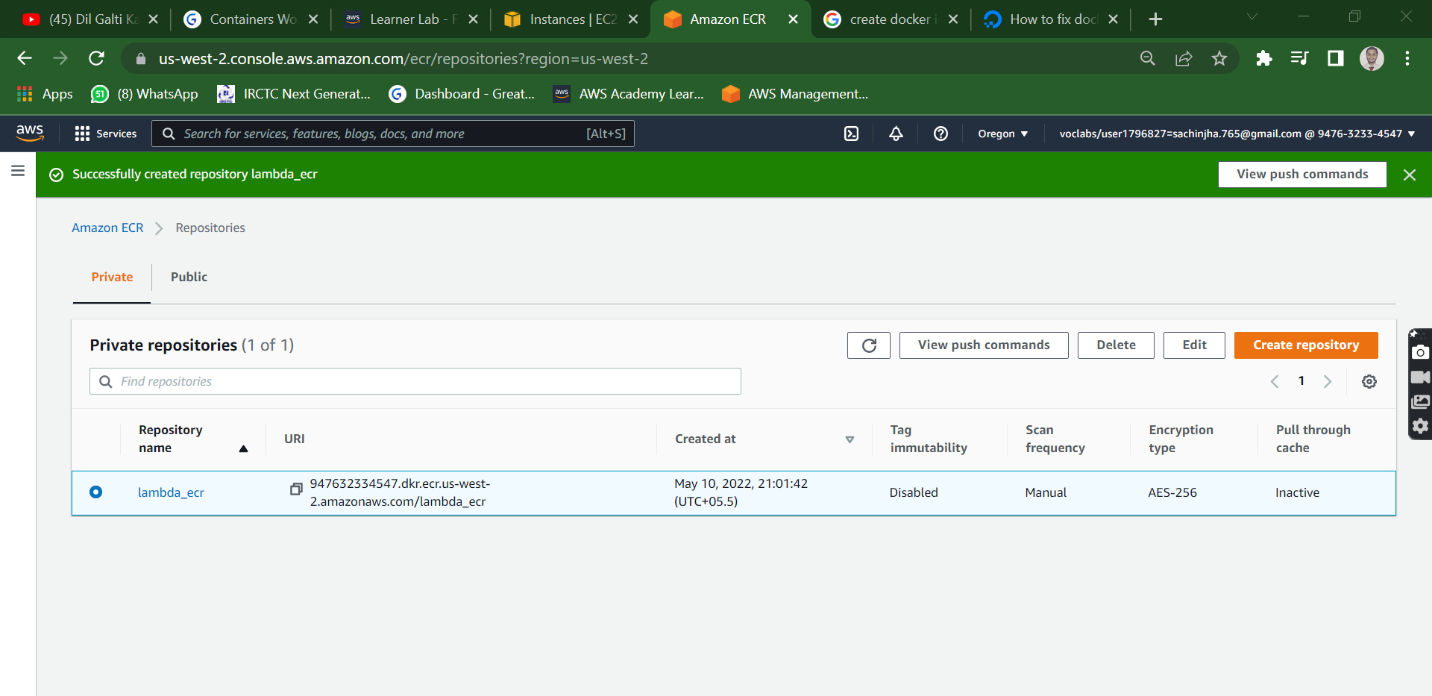
**Step 2: Create ECR repository and upload image to ECR**

|  |  |  |
| --- | --- | --- |
| Step number | a |  |
| Step name | Creating the ECR repository |  |
| Instructions | 1. Go to the ECR service on the AWS console 2. Select the Repositories from the left pane 3. Create a new private repository named **lambda\_ecr** with the default settings |  |
| Step number | b |  |
| Step name | Image upload to ECR |  |
| Instructions | 1. Once the repository is created, select the repository and then click  on “View push commands“ on the top right 2. From the pop up screen which appears, run commands 1, 3 and 4 after logging into the EC2 instance created above. Note that command 2 was already executed in the previous step when the image was created. For reference, the commands will be in the format shown below:   **aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <xxxxxxx.dkr.ecr.us-east-1.amazonaws.com>**  **docker tag lambda\_ecr\_image:latest <xxxxxxx.dkr.ecr.us-east-1.amazonaws.com/lambda\_ecr>:latest**  **docker push <xxxxxxx.dkr.ecr.us-east-1.amazonaws.com/lambda\_ecr>:latest** |  |
| Expected screenshots | 1) Creation of Repository 2) View push commands  3) Login Succeeded 4) Tagging of the image 5) Pushing of image to ECR 6) Image uploaded on the ECR repo |  |

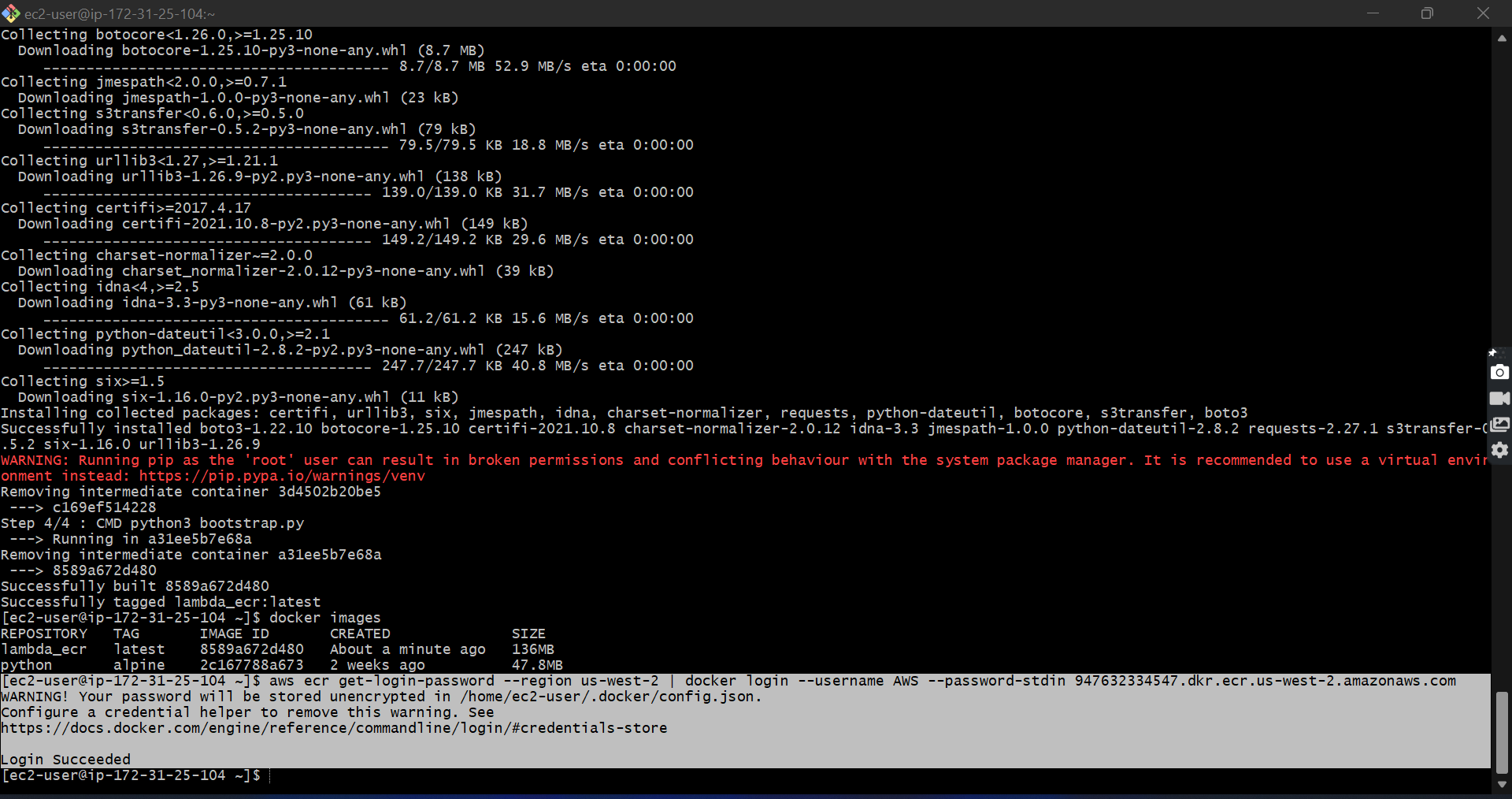
**<Insert screenshot for a(1) here>**

****

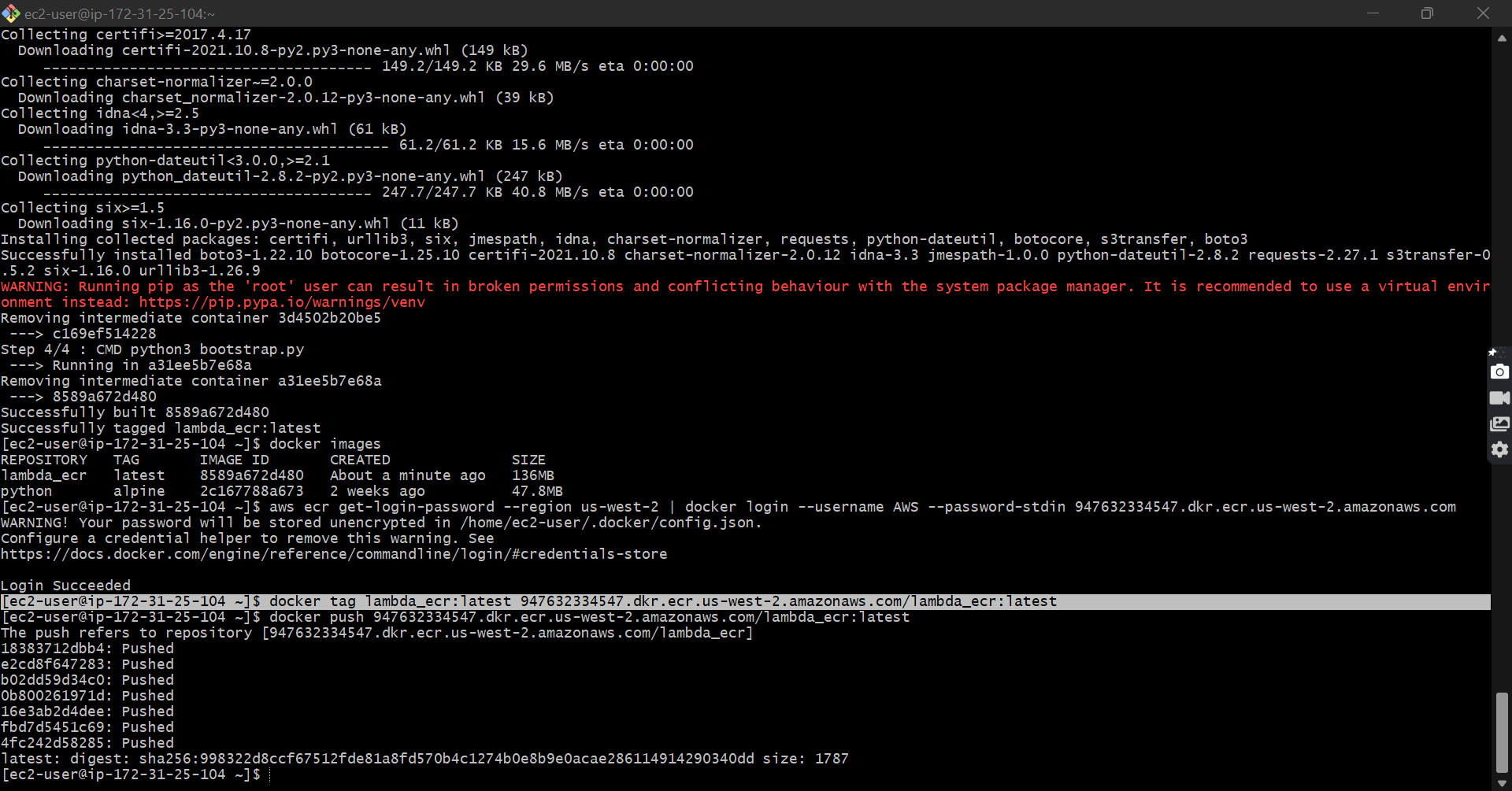
**<Insert screenshot for b(1) here>**

****

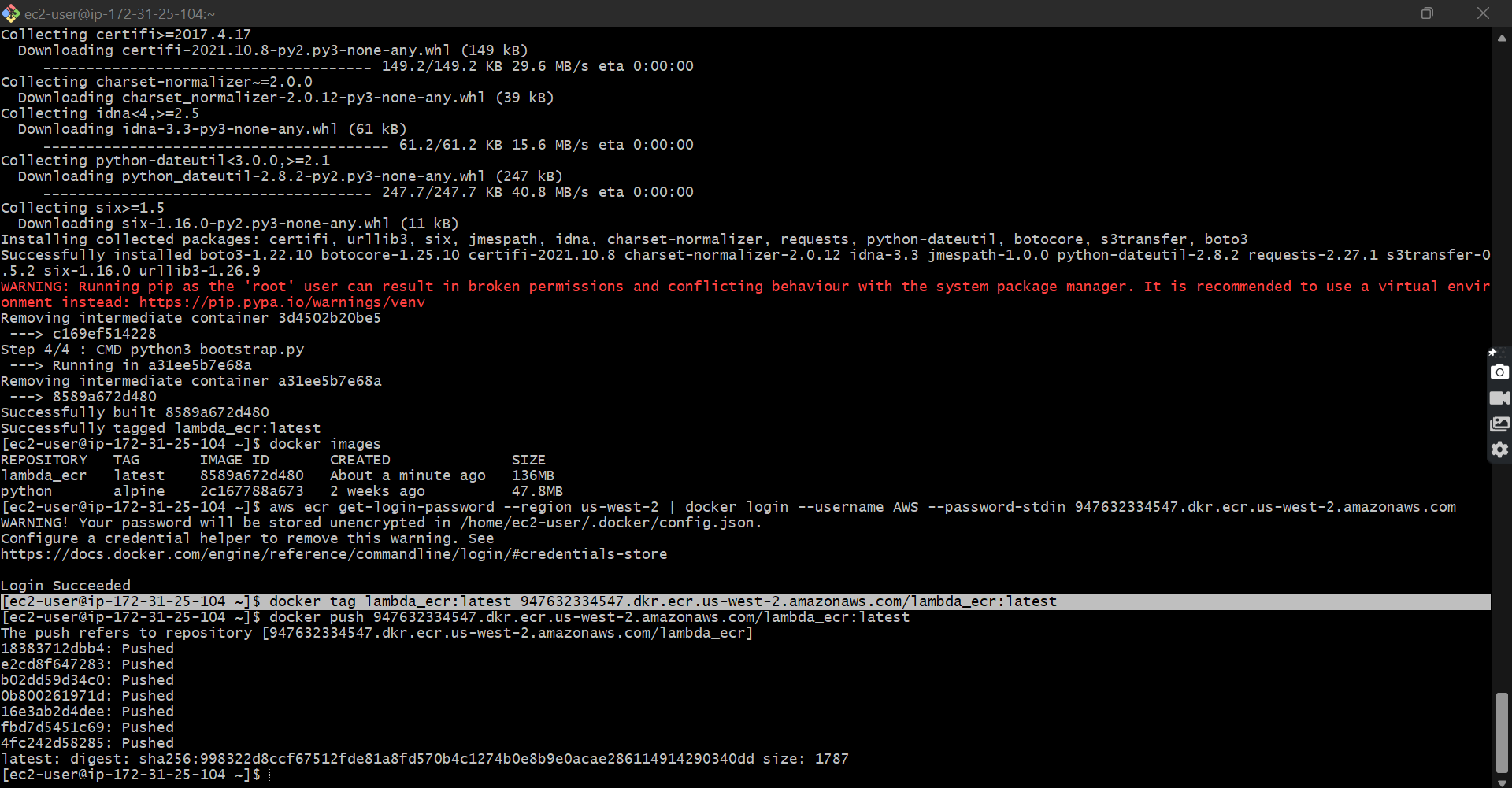
**<Insert screenshot for b(2) here>**

****

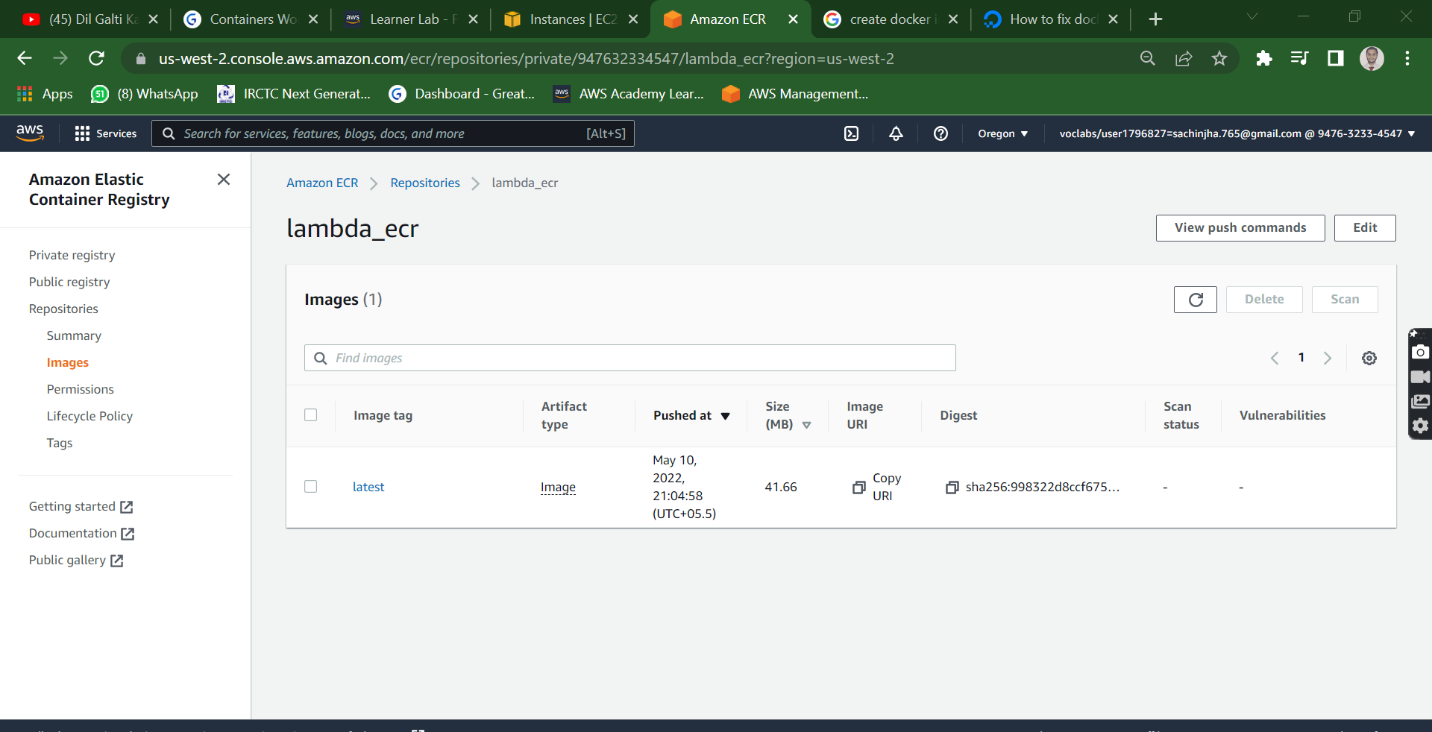
**<Insert screenshot for b(3) here>**

****

**<Insert screenshot for b(4) here>**

****

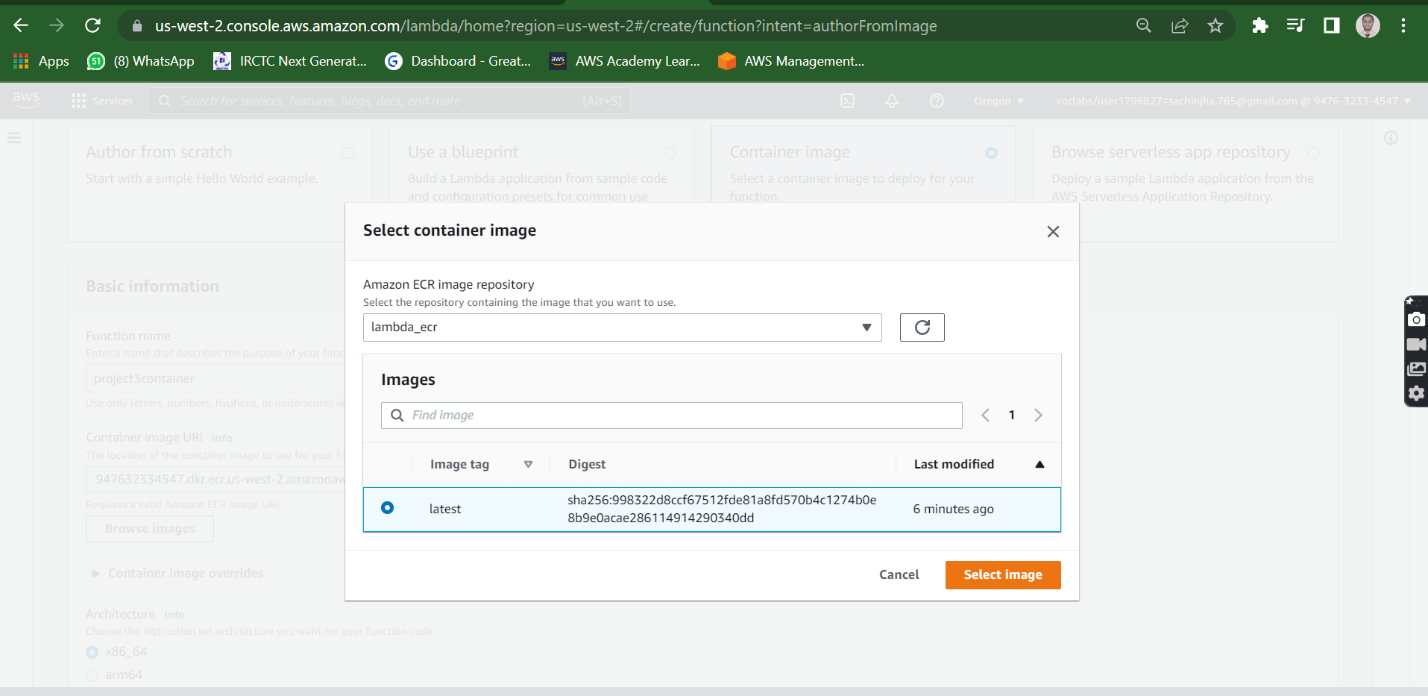
**<Insert screenshot for b(5) here>**

****

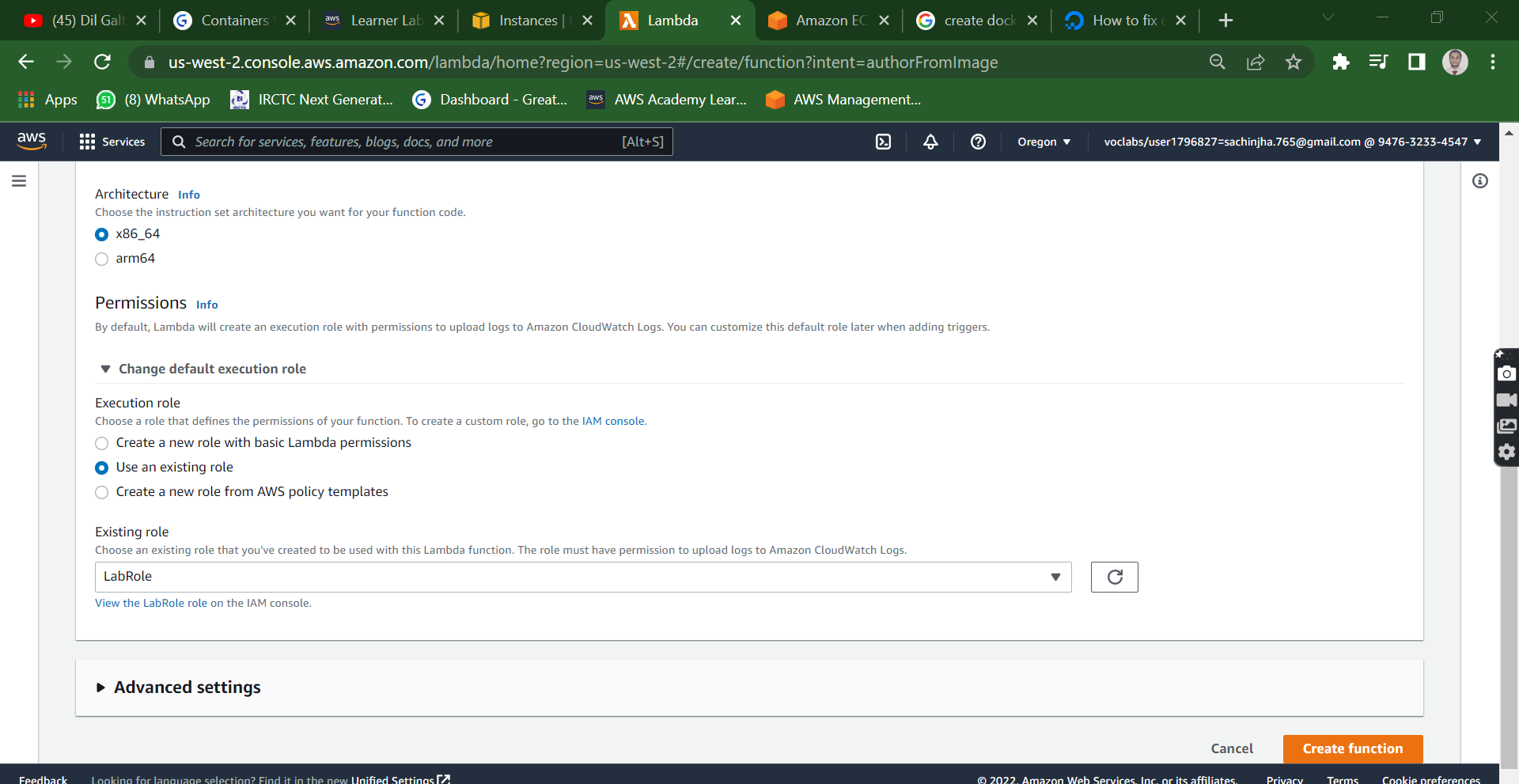
**Step 3: Creation of Lambda function to test the image**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step number | a |  |  |  |
| Step name | Create the Lambda function and test the image |  |  |  |
| Instructions | 1. Navigate to the AWS Lambda service using the AWS Console 2. Click on **Create Function** 3. Under Create Function page select the ‘Container image’ option and enter a function name of your choice 4. For ‘Container image URI’ Click on “Browse Images” and select the repository and the image 5. Use the existing IAM role – LabRole. 6. Click on Create 7. Wait a few minutes for the function to be created 8. Test the function with the default “Hello World” test to see the result. | | | |
| Expected screenshots | 1) Container image selection 2) Execution role selection 3) Created function 4)Test result of function | | | |

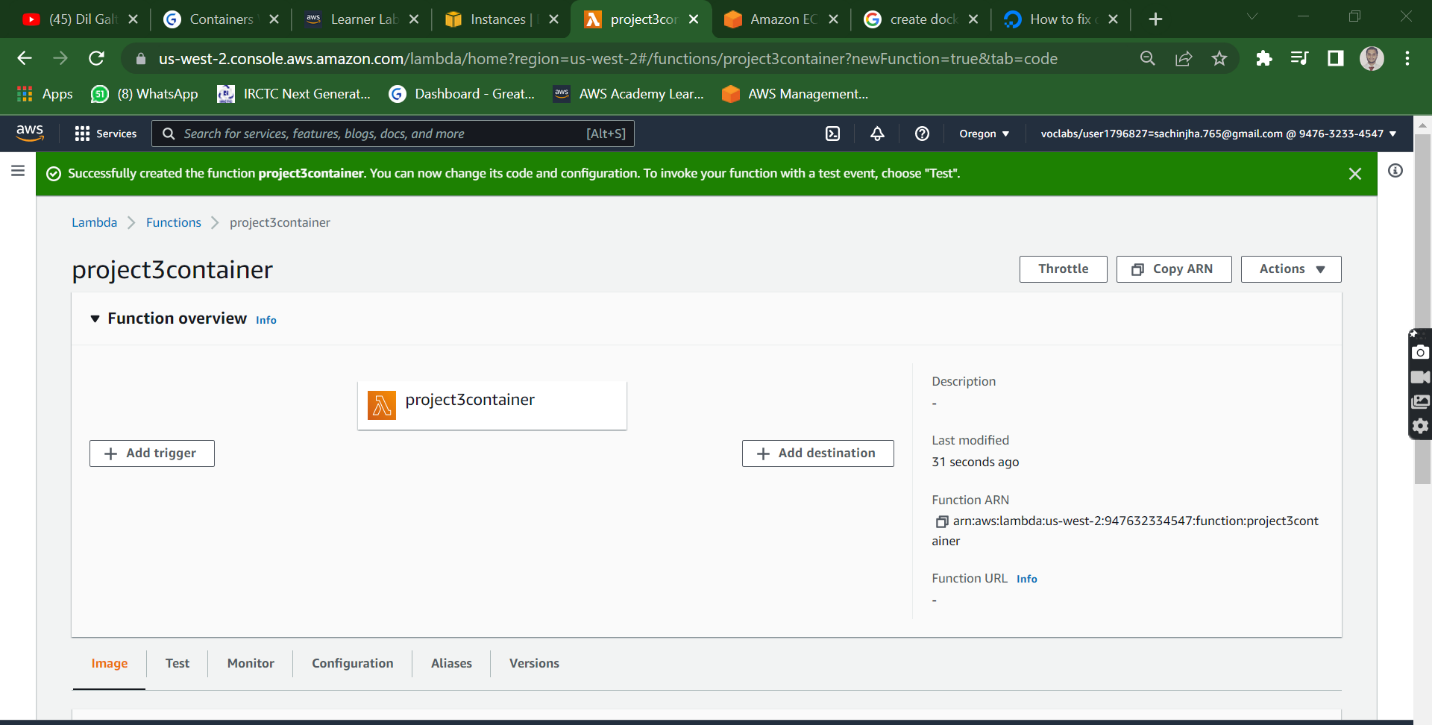
**<Insert screenshot for a(1) here>**

****

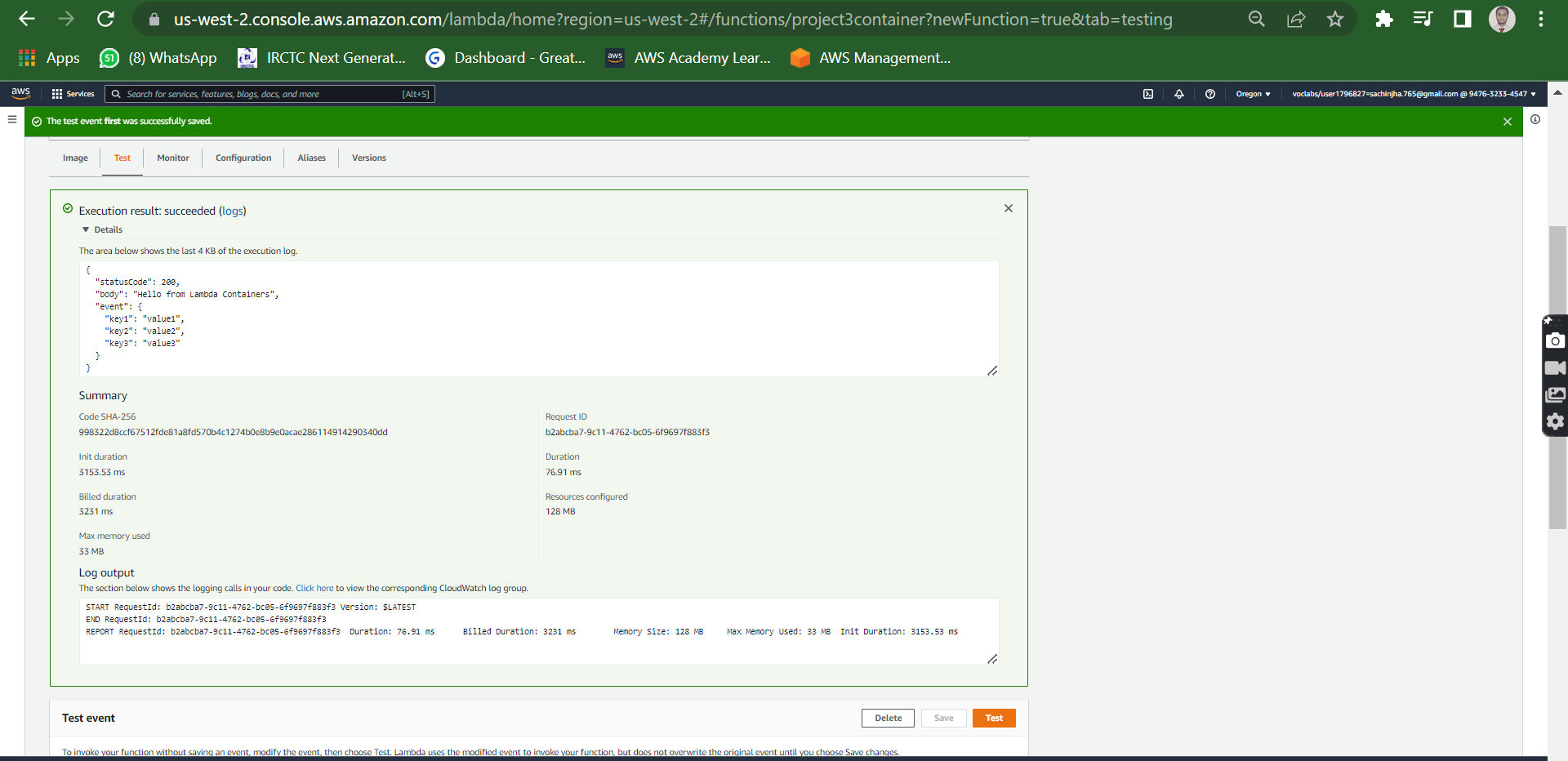
**<Insert screenshot for a(2) here>**

****

**<Insert screenshot for a(3) here>**

****

**<Insert screenshot for a(4) here>**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Answer the following questions** | | | | **Points** |
| Q1 | How long does a container stay in the running state if it is not manually halted? | | | 1 |
|  | a) As long as the container’s PID 1 is running |  |  |  |
|  | b) Has a set timeout after which it pauses |  |  |  |
|  | c) Until its container is expunged |  |  |  |
|  | d) Docker daemon process scheduler decides on load |  |  |  |
|  | Enter your answer here | a |  |  |
|  |  |  |  |  |
| Q2 | Which of the following best illustrates the relationship between an image and a container? | | | 1 |
|  | a) Executable and its hard link |  |  |  |
|  | b) Executable and process |  |  |  |
|  | c) Parent and child process |  |  |  |
|  | d) Many to one |  |  |  |
|  | Enter your answer here | b |  |  |
|  |  |  |  |  |
| Q3 | What is the maximum amount of RAM a container can consume if the memory flag is not used? | | | 1 |
|  | a) 8GiB |  |  |  |
|  | b) 32GiB |  |  |  |
|  | c) None of these |  |  |  |
|  | d) As much as the host instance has free |  |  |  |
|  | Enter your answer here | d |  |  |
|  |  |  |  |  |
| Q4 | Which of the following will happen in the same Docker image is pushed to Docker Hub multiple times with different tags | | | 1 |
|  | a) Dockerhub will refuse to upload the image |  |  |  |
|  | b) The layers in the first image (if unchanged) will be reused in subsequent pushes |  |  |  |
|  | c) Dockerhub will merge the images |  |  |  |
|  | d) The same image cannot have multiple tags |  |  |  |
|  | Enter your answer here | d |  |  |
|  |  |  |  |  |
| Q5 | Which of the following will run a Docker container in interactive mode? | | | 1 |
|  | a) -v |  |  |  |
|  | b) -it |  |  |  |
|  | c) -b |  |  |  |
|  | d) -u |  |  |  |
|  | Enter your answer here | b |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Q6 | How would data persistence be handled in a container environment set up for autoscaling? | | | 4 |
|  |  | | |  |
|  | The data persistence be handled in a container environment set up for autoscaling. We have to choose the following things for the correct conatainer environment for autoscaling.   * Choosing the right storage type for your containers * Amazon EFS volumes * Docker volumes * FSx for Windows File Server   Applications that are running in an Amazon ECS cluster can use a variety of AWS storage services and third-party products to provide persistent storage for stateful workloads. You should choose your storage backend for your containerized application based on the architecture and storage requirements of your application.  Suppose you have an application such as a transactional database that requires sub-millisecond latency and doesn’t need a shared filesystem when it's scaled horizontally. For such an application, we recommend using Amazon EBS volumes for persistent storage. Currently, Amazon ECS supports Amazon EBS volumes for tasks hosted on Amazon EC2 only. Support for Amazon EBS volumes isn't available for tasks on Fargate. Before using Amazon EBS volumes with Amazon ECS tasks, you must first attach Amazon EBS volumes to container instances and manage volumes separately from the lifecycle of the task.  For clusters that contain Windows instances, FSx for Windows File Server provides persistent storage for containers. FSx for Windows File Server filesystems supports multi-AZ deployments. Through these deployments, you can share a filesystem with Amazon ECS tasks running across multiple Availability Zones. | | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Q7 | Why is this statement false? "Docker is the only popular choice for microservices deployment". | | | 3 |
|  |  |
|  | Docker is built on Unix/Linux containerization. There are other microservice tools out there but they all use this containerization layer. So, no, containers are the most popular choice.  Docker is one of the most microservices due to the following reason.   1. Easy to manage with the help of docker commands 2. You can easily setup an environment which is suitable or as required for your piece of code to run with the help of docker file 3. And also, it’s an independent application or demon which uses CGroups from the OS kernel it’s independent from the OS 4. Also, with the docker-compose you can maintain multiple stacks of your application (you can run multi-tier application) 5. Very easy to versioning based on the development testing and production requirements. 6. Last but not the least docker container can easily ship from one server to another. you can push your container to the docker registry or AWS registry Thanks. Let me know if you have any other queries | | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | **Max points** | **12** |

|  |  |
| --- | --- |
| **Grades distribution** |  |
| MCQs | 5 (1 point each) |
| Subjective questions | 8 points (4+4) |
| Implementation screenshots | 12 points (1 point each) |
| Total | 25 points |