CLOUD COMPUTING

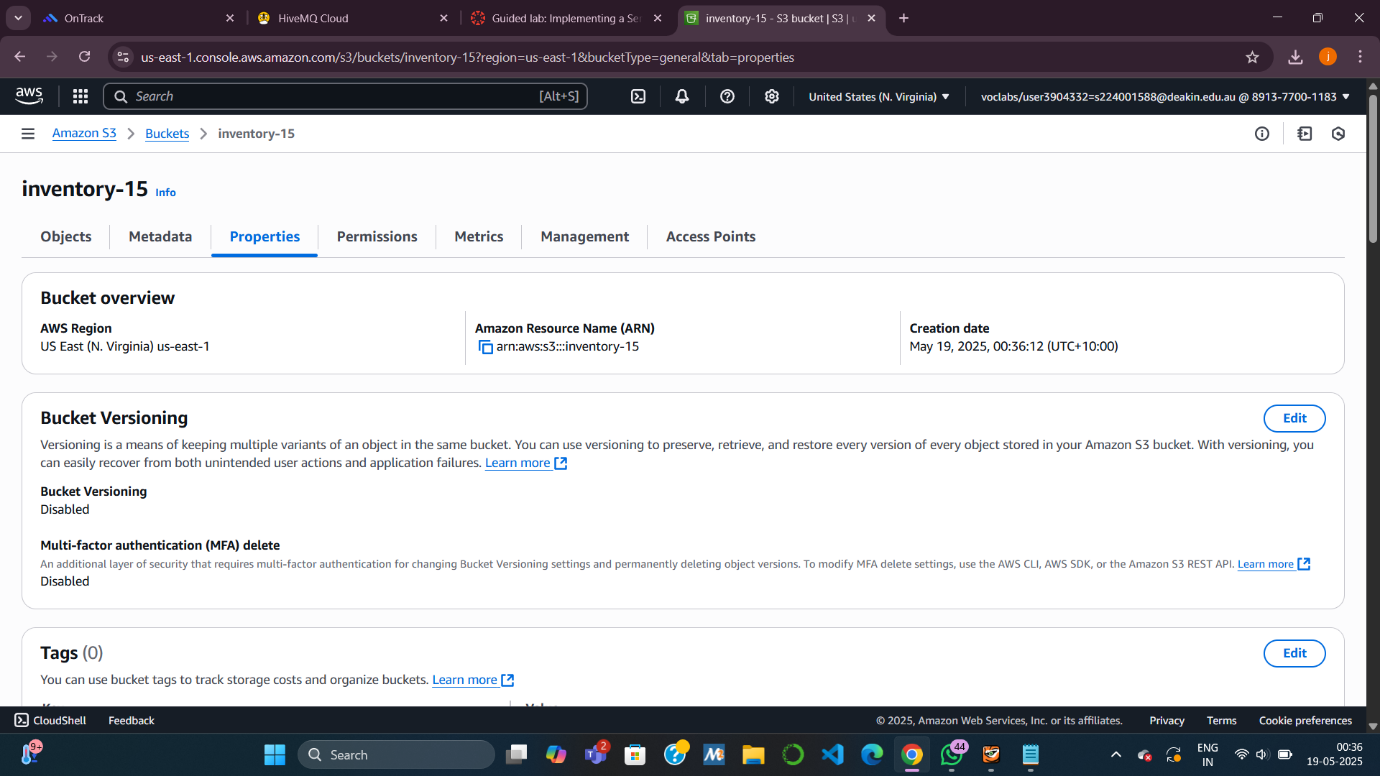
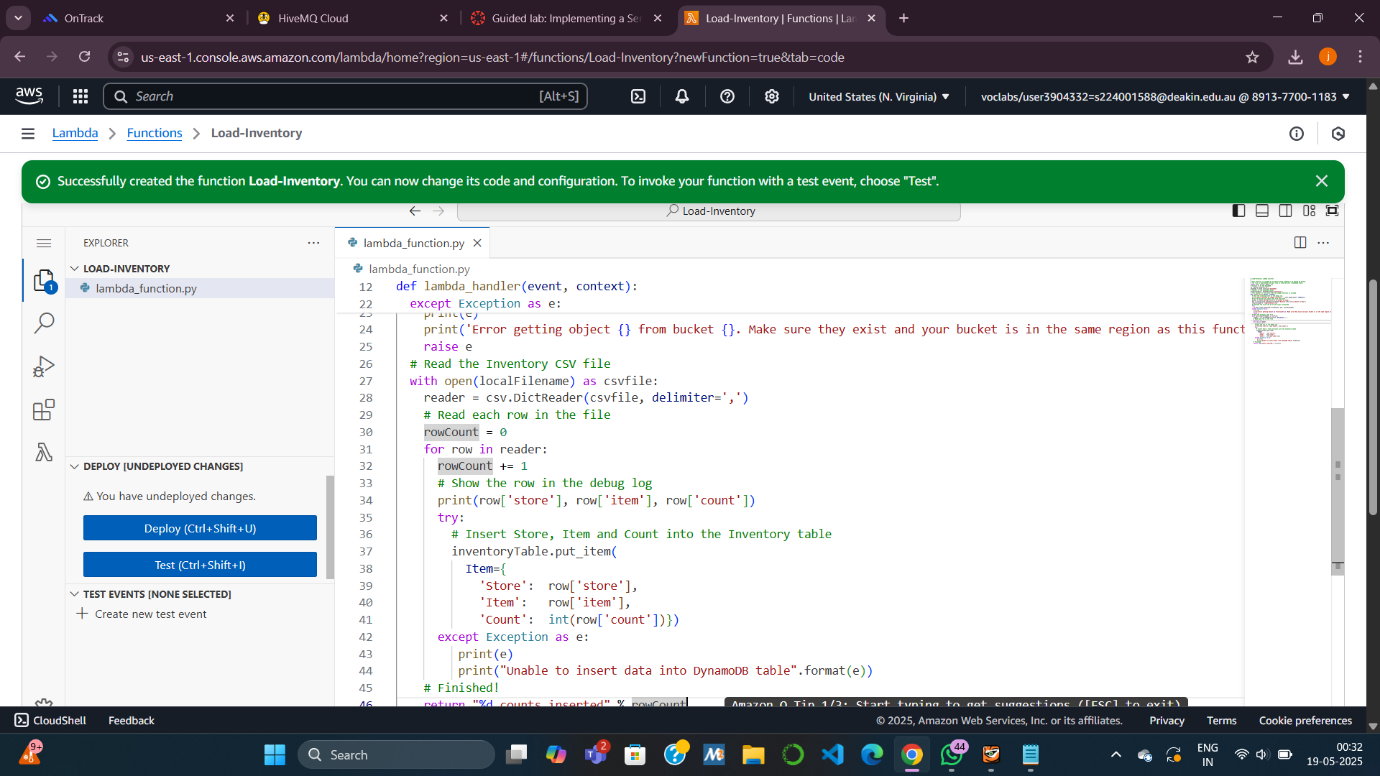
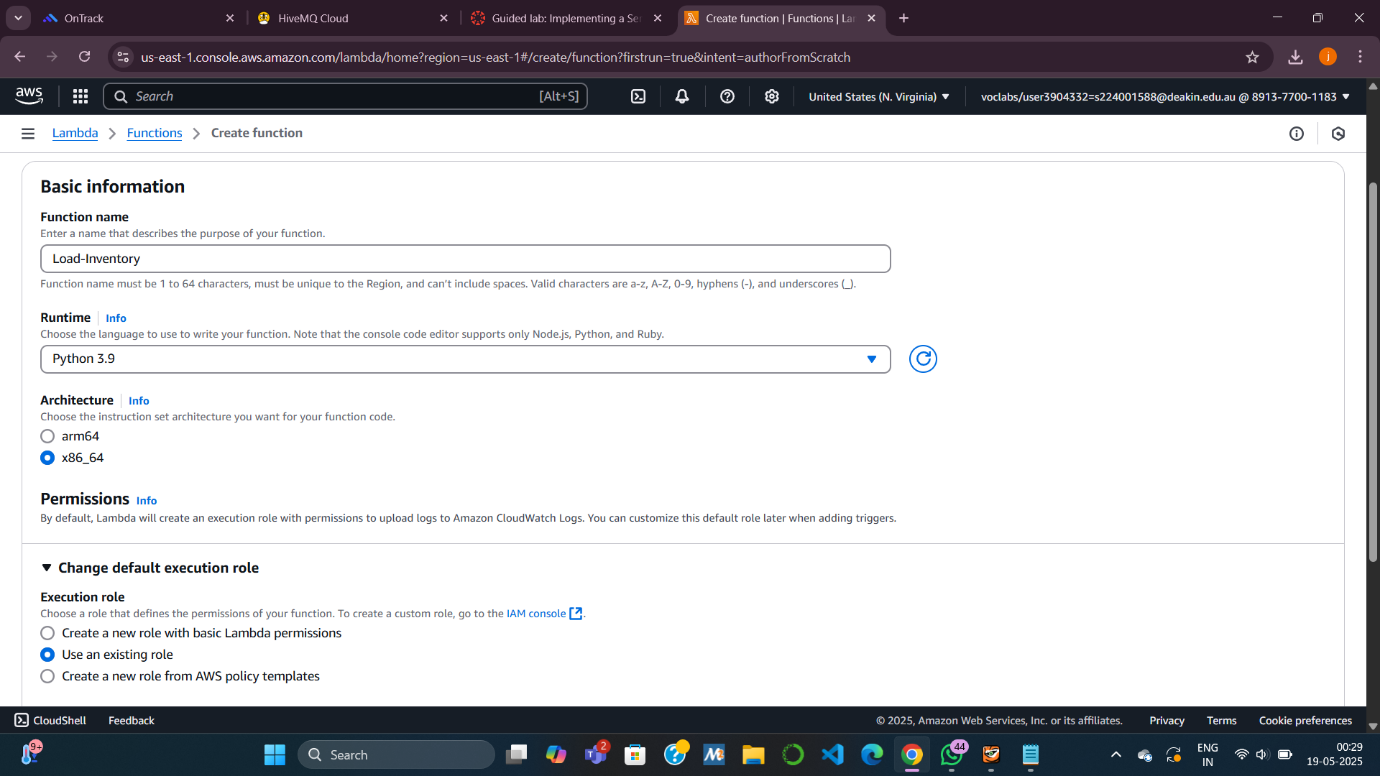
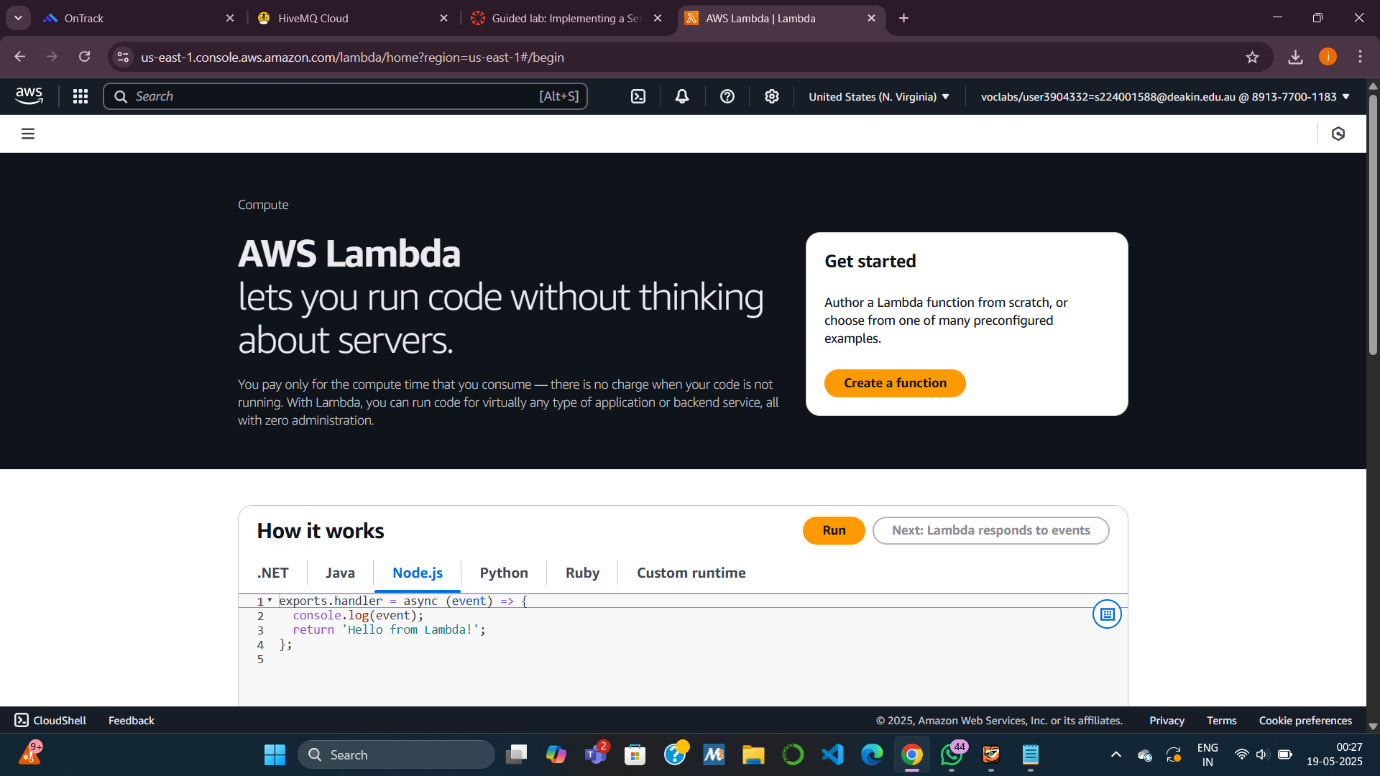
TASK 10.2D

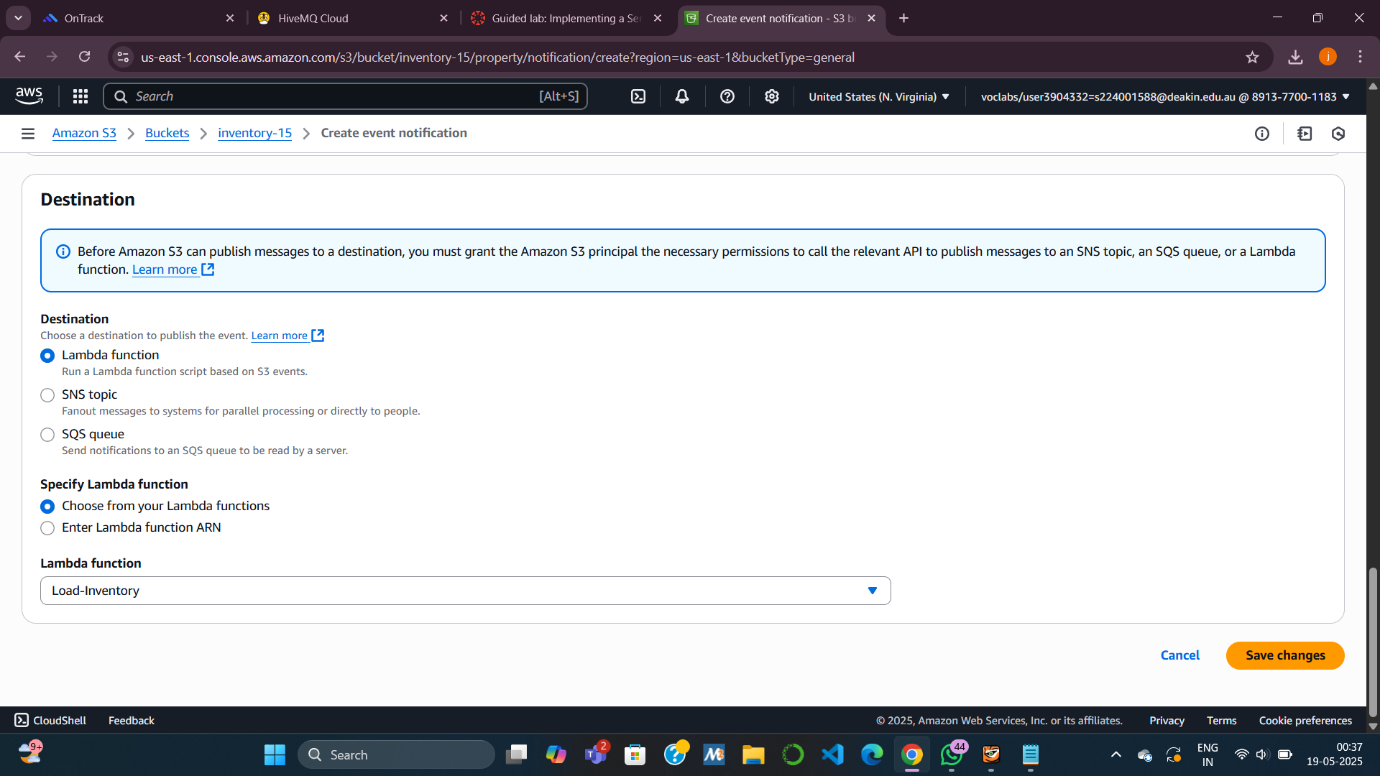
Implementing a Serverless Architecture with AWS Lambda

**Task 1: Creating a Lambda Function to Load Data**

**Goal:** Process a CSV inventory file uploaded to S3 and insert its contents into DynamoDB.

* **Create Lambda function (Load-Inventory)**:  
  Uses Python 3.8, triggered by S3, inserts inventory rows into a DynamoDB table.
* **Attach existing role**:  
  Lambda-Load-Inventory-Role gives permissions to access S3 and DynamoDB.
* **Paste provided code**:
  + Downloads file from S3.
  + Reads CSV line-by-line.
  + Adds each record to the Inventory DynamoDB table.
* **Deploy the function**:  
  Save and deploy the code to make it active.

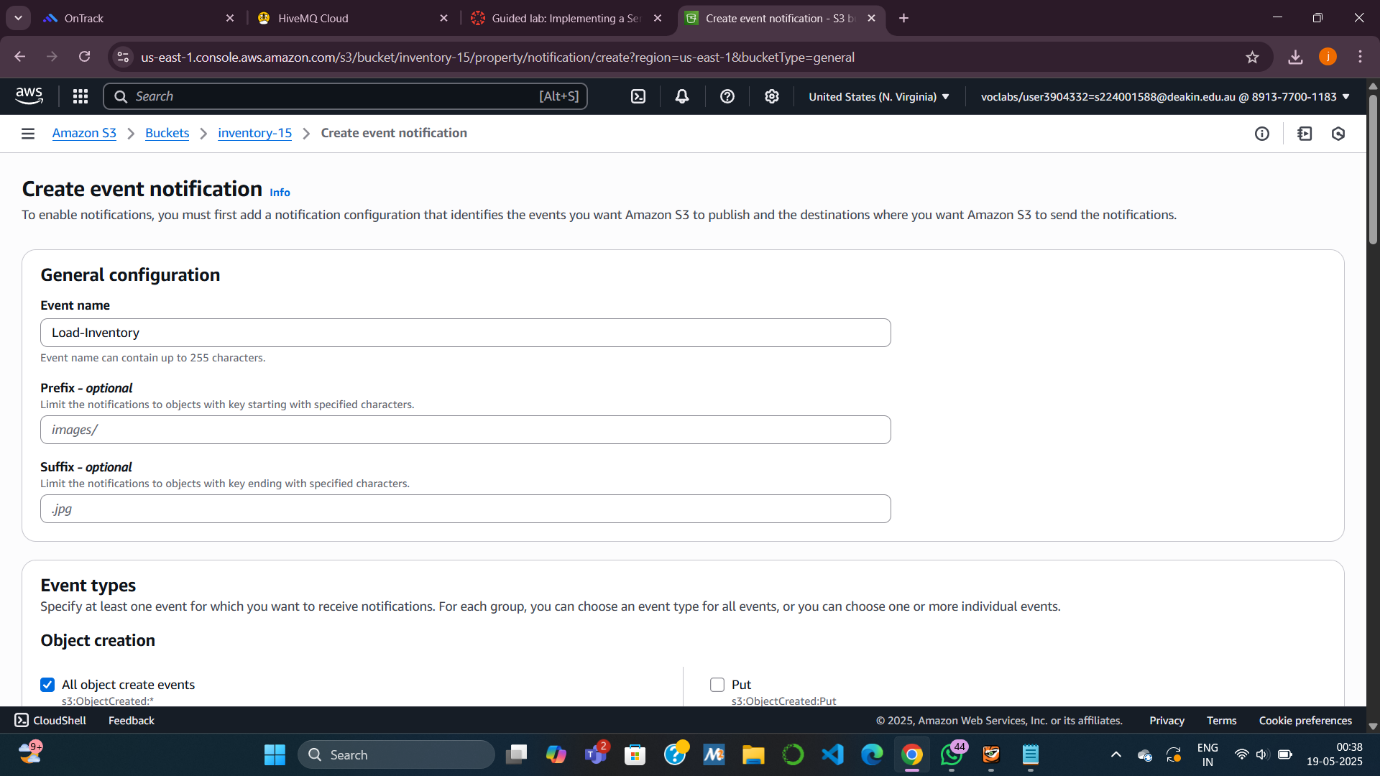




**Task 2: Configuring an Amazon S3 Event**

**Goal:** Automatically trigger the Lambda function when a CSV file is uploaded.

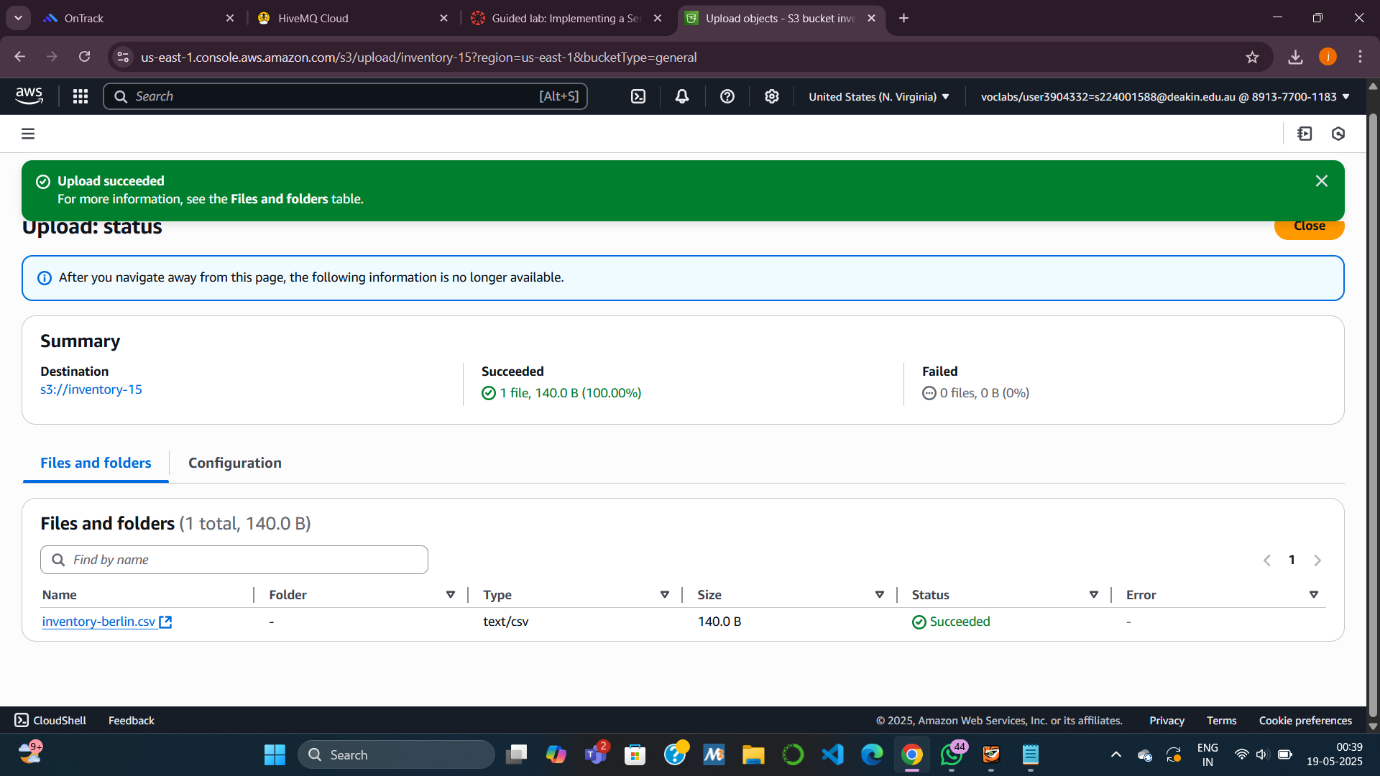
* **Create S3 bucket**:  
  Use a unique name like inventory-123.
* **Add event notification**:
  + Trigger type: “All object create events”.
  + Destination: Lambda function (Load-Inventory).

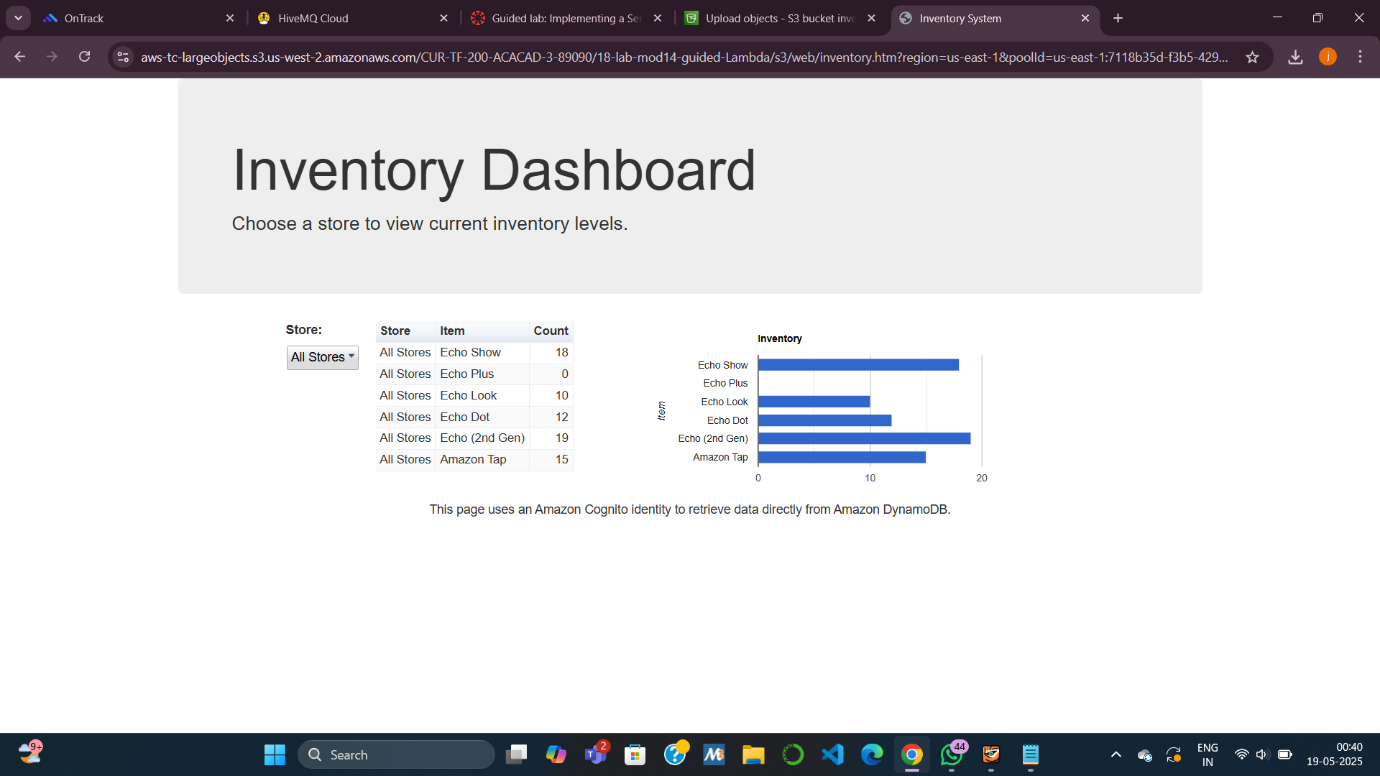
This links file uploads in the S3 bucket to your Lambda function.J

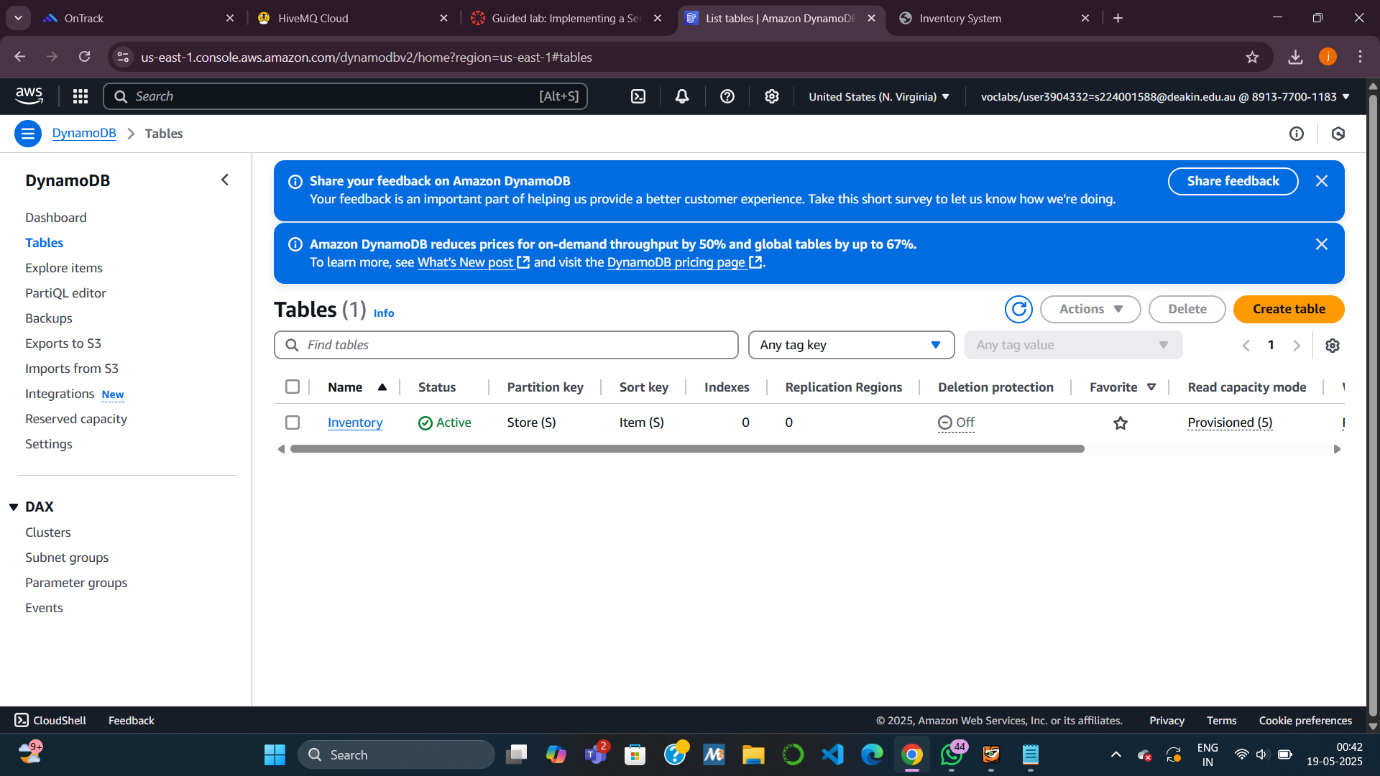
**Task 3: Testing the Loading Process**

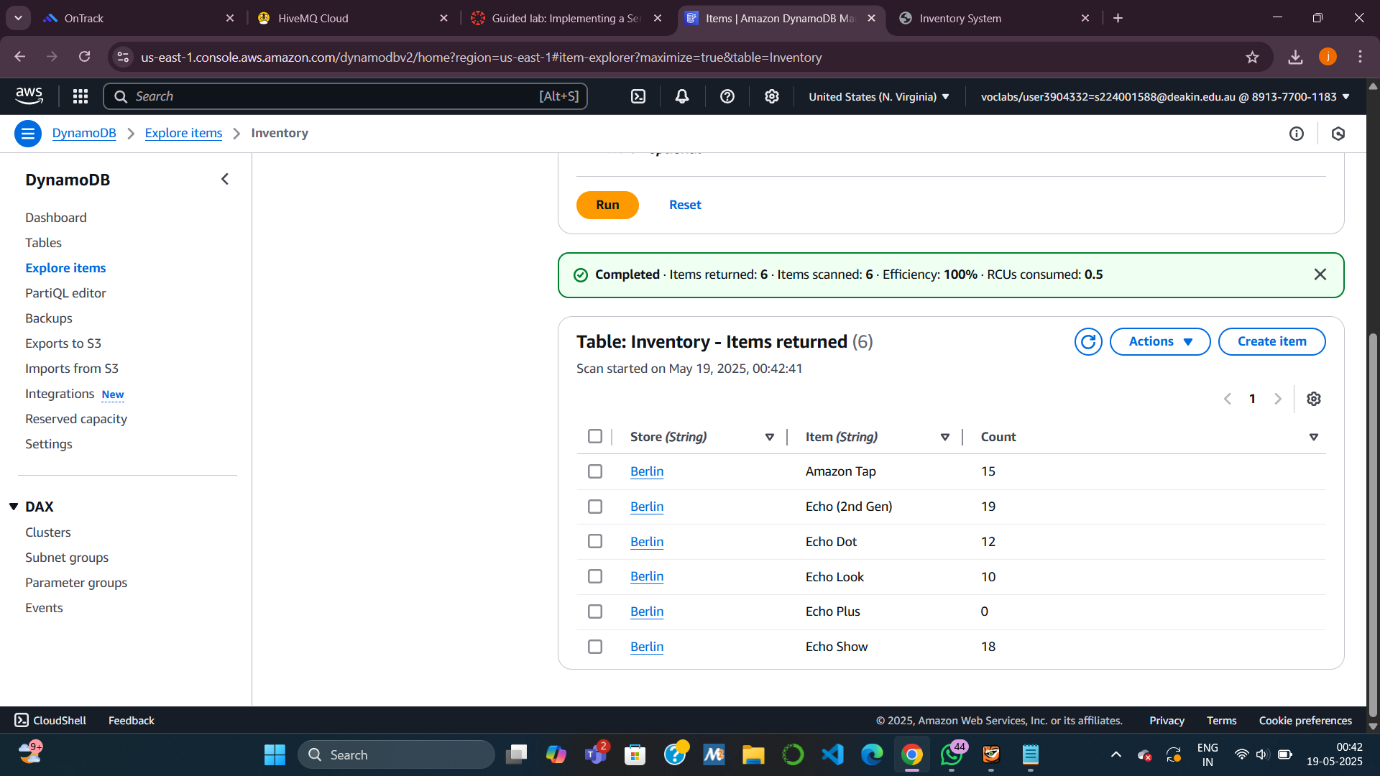
**Goal:** Test the setup by uploading an actual CSV file.

* **Download sample files**:  
  Files like inventory-berlin.csv contain store, item, count.
* **Upload to S3 bucket**:  
  Automatically triggers Load-Inventory.
* **View results**:
  + **Dashboard URL**: Opens a web app showing inventory (read from DynamoDB).
  + **DynamoDB Console**: Use *Explore table items* to confirm entries are inserted.







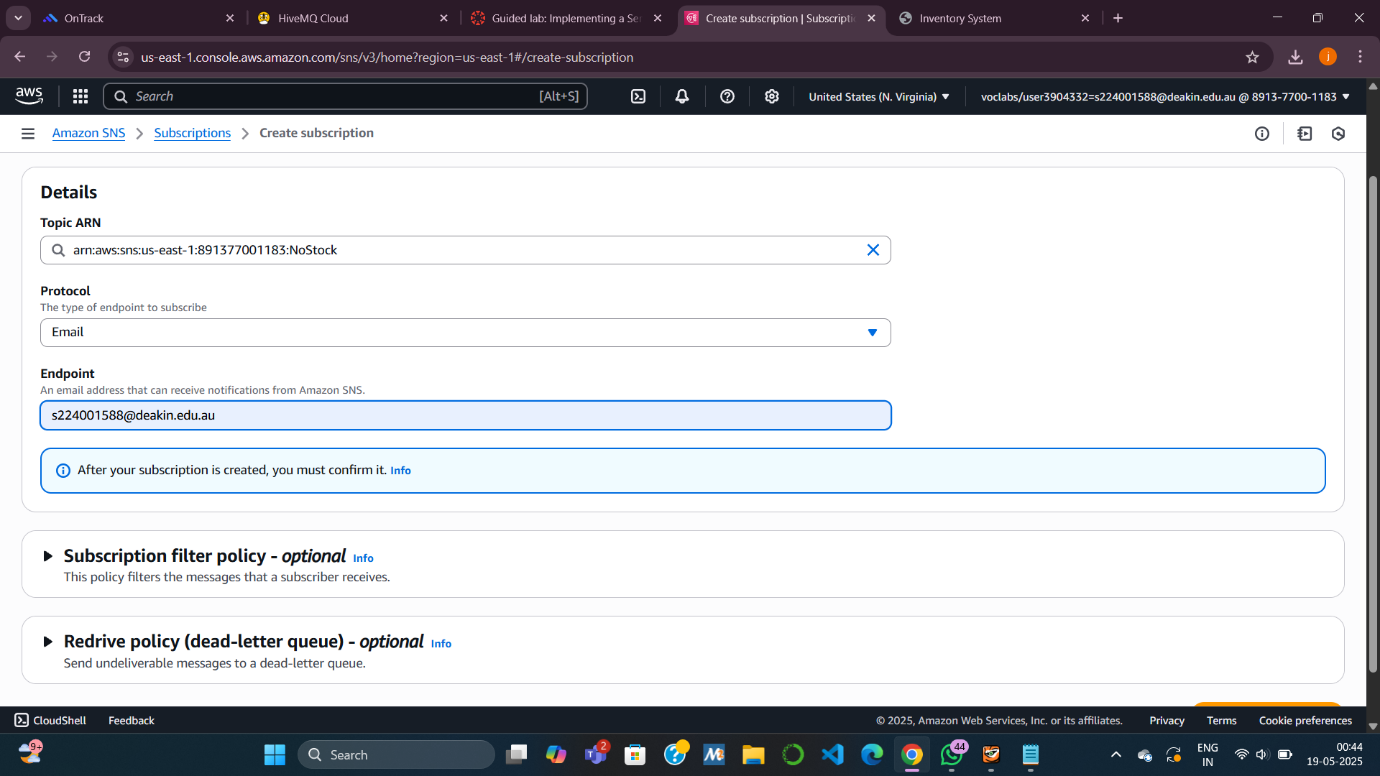
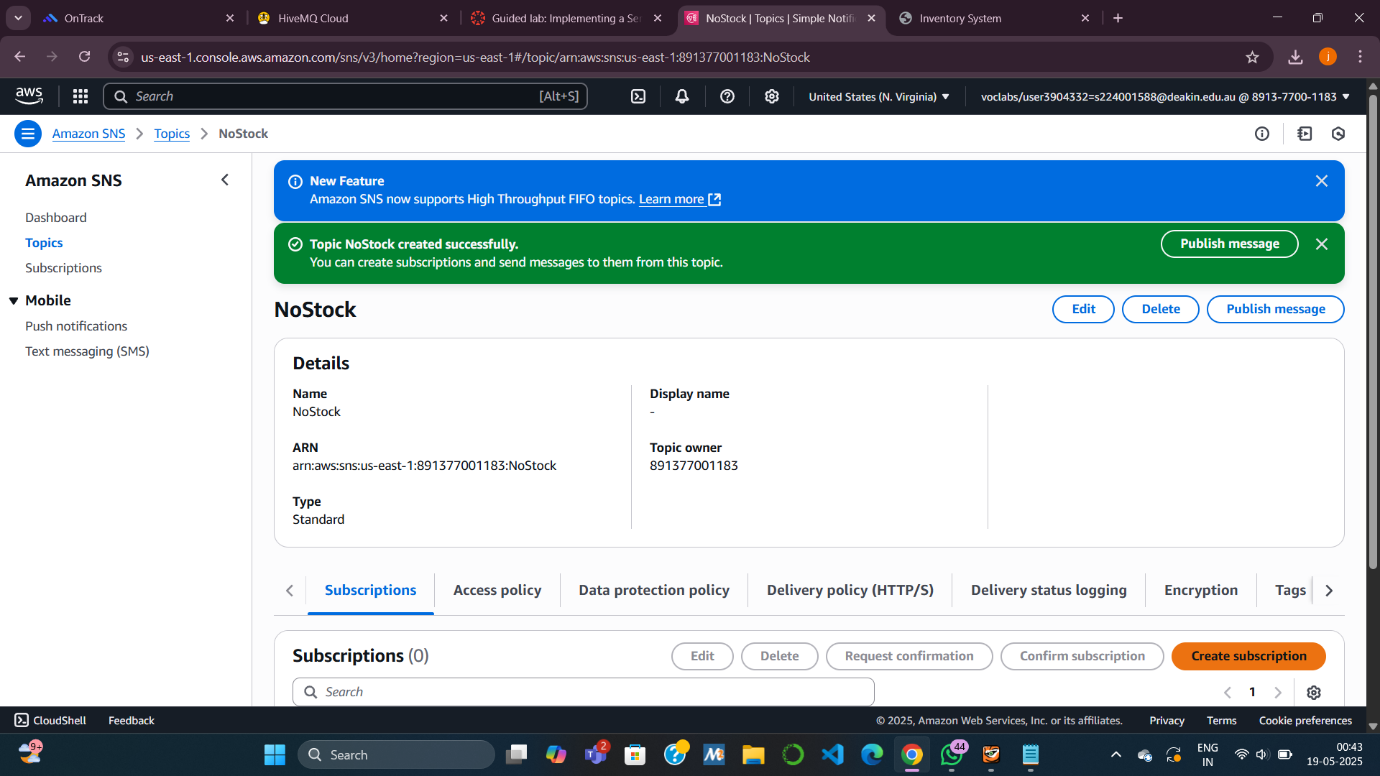


**Task 4: Configuring Notifications with Amazon SNS**

**Goal:** Set up an SNS topic to alert users when inventory is zero.

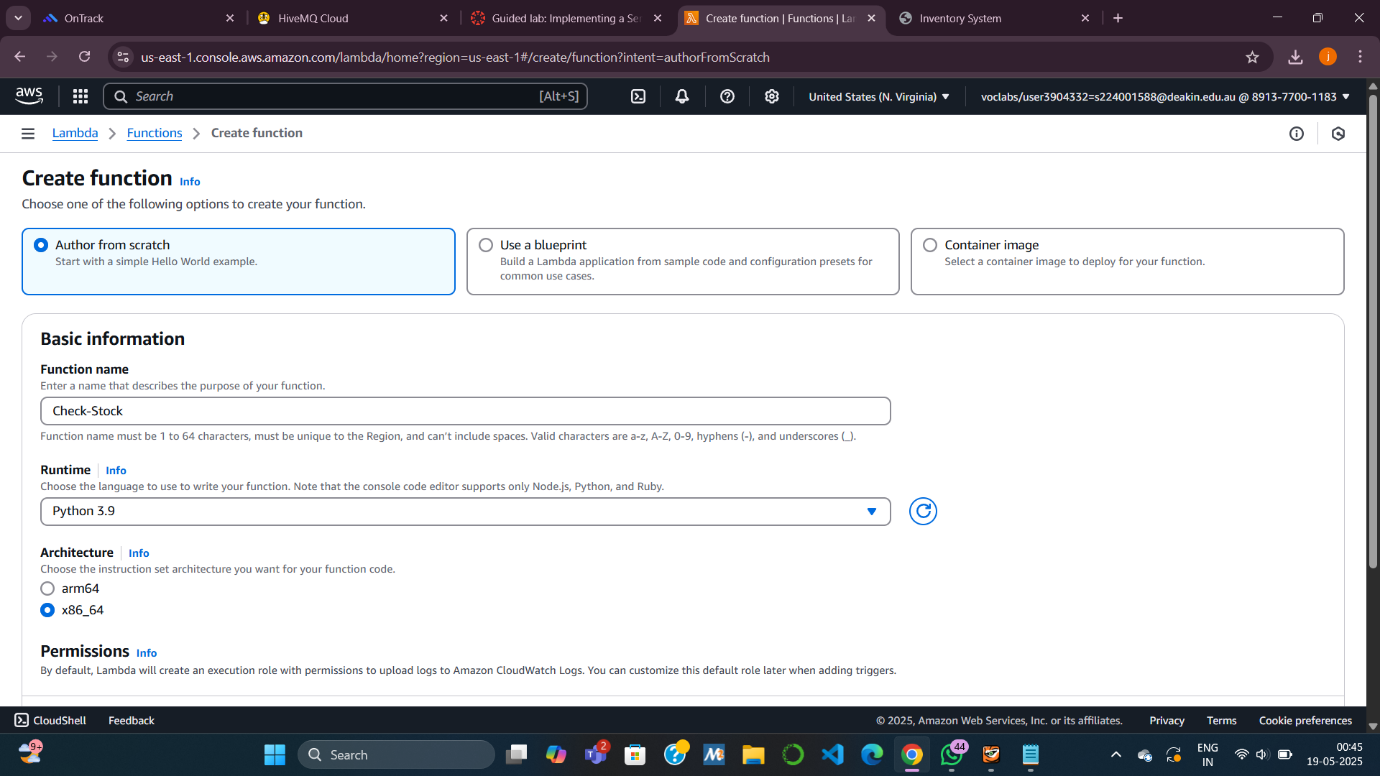
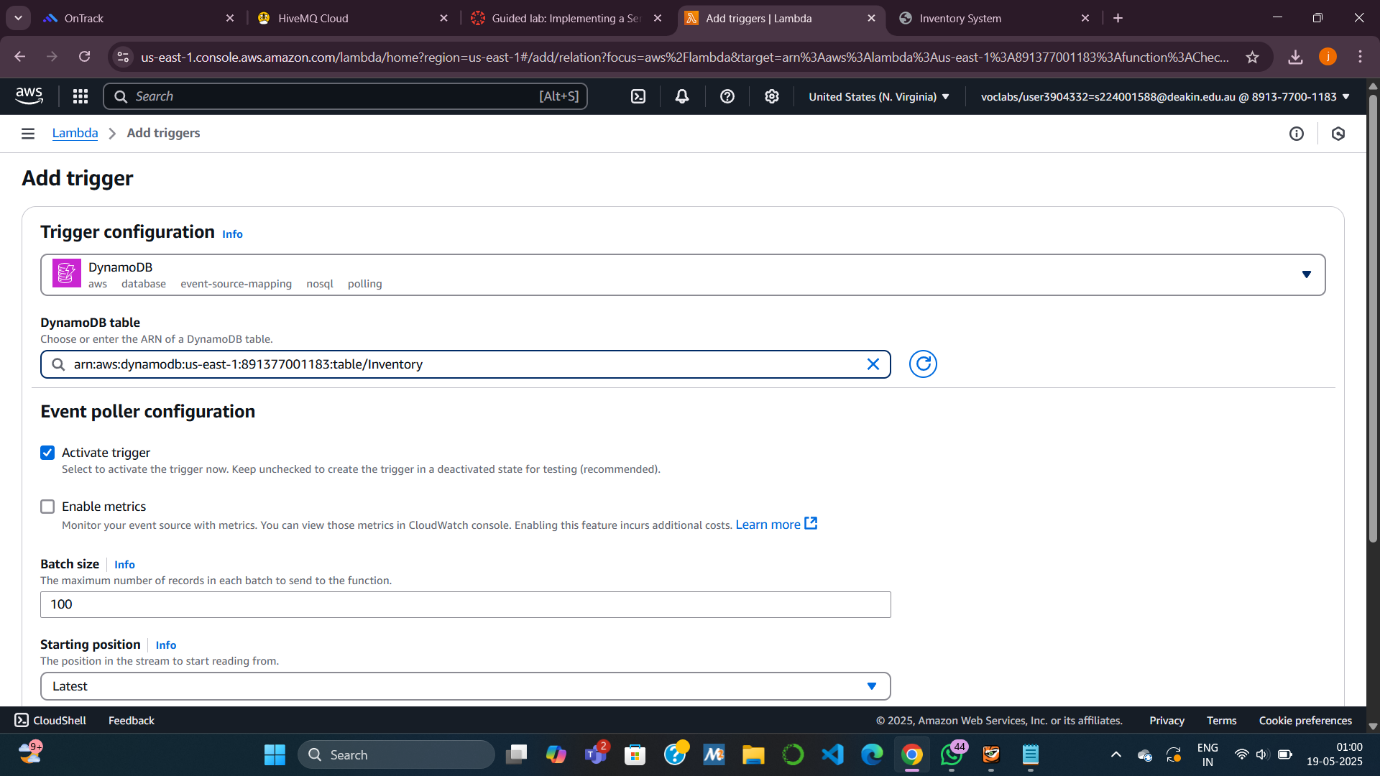
* **Create SNS Topic**:  
  Name it NoStock.
* **Subscribe to topic**:
  + Choose **Email** as protocol.
  + Enter your email → Confirm via email link.

Now SNS is ready to send alerts when items are out of stock.



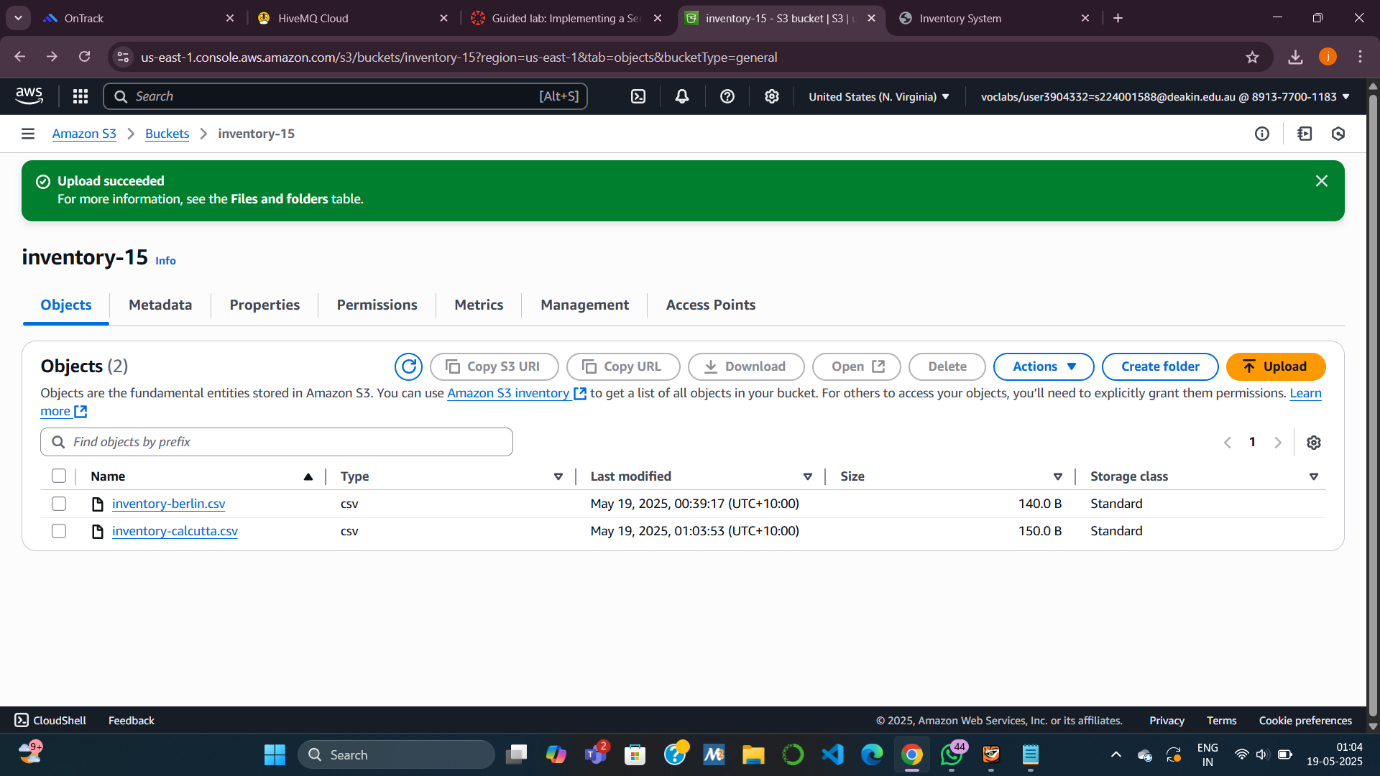
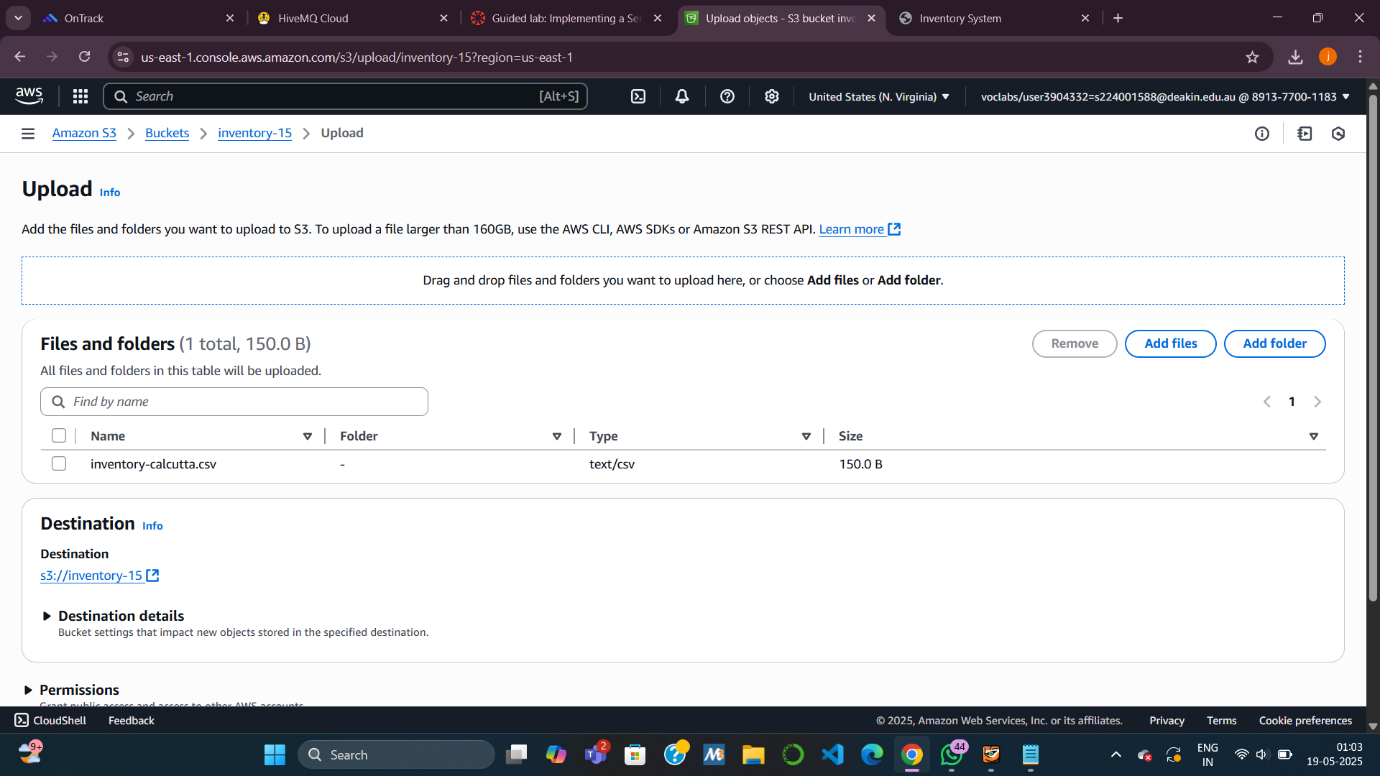
**Task 5: Creating a Lambda Function to Send Notifications**

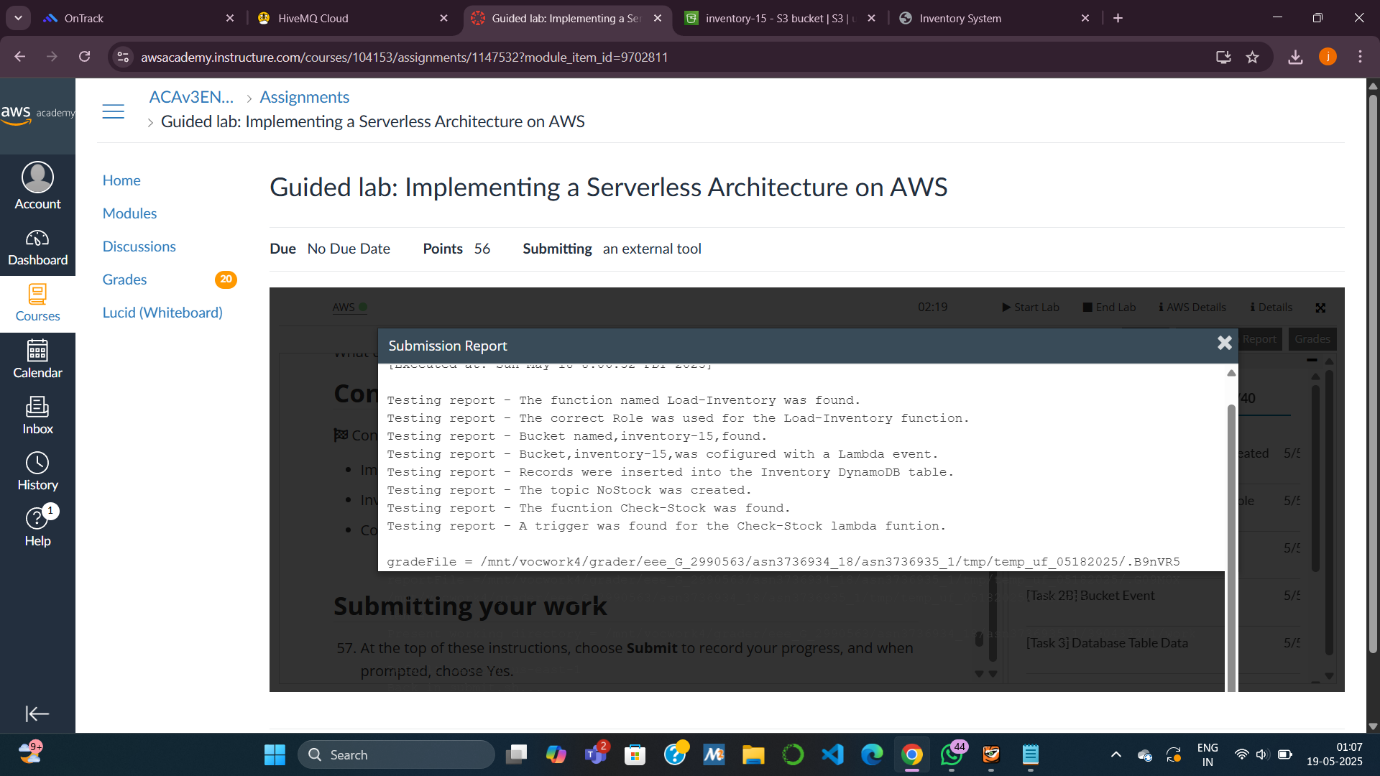
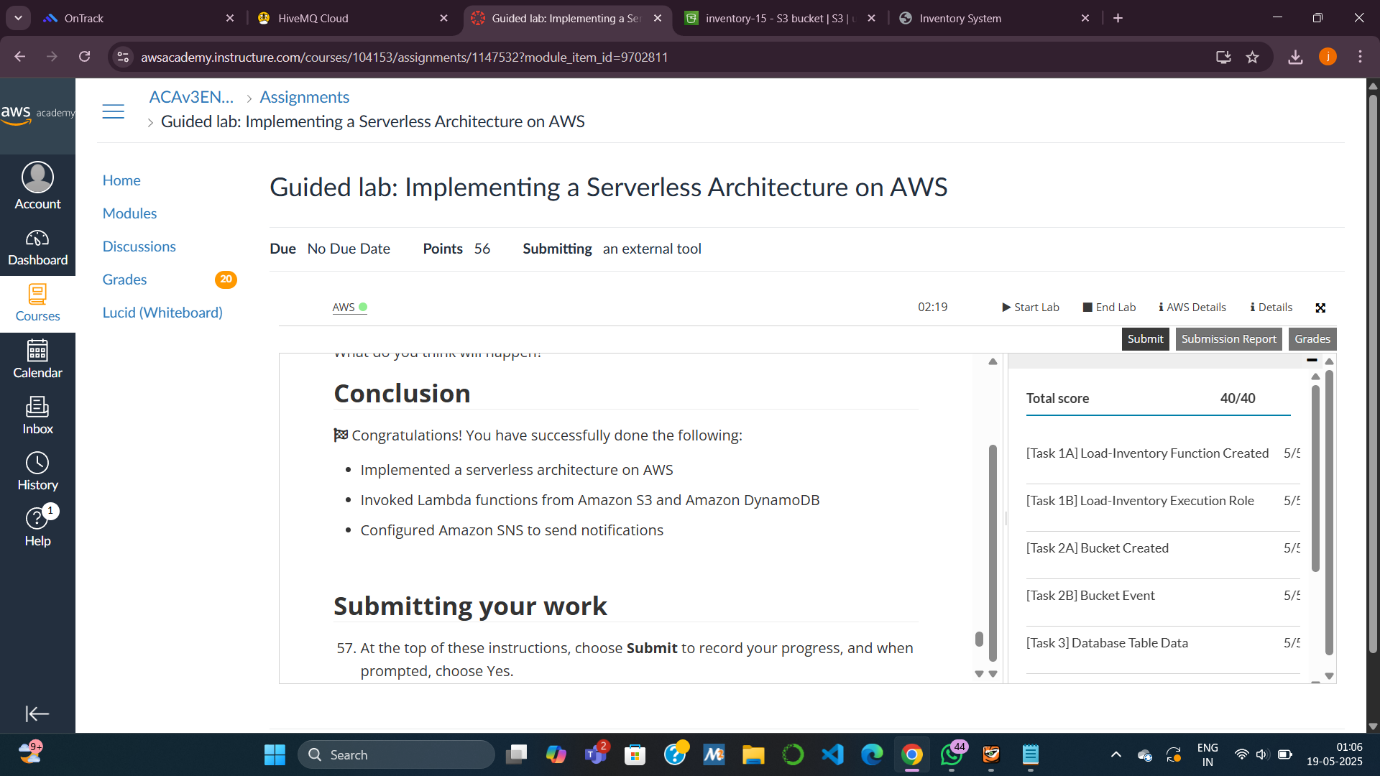
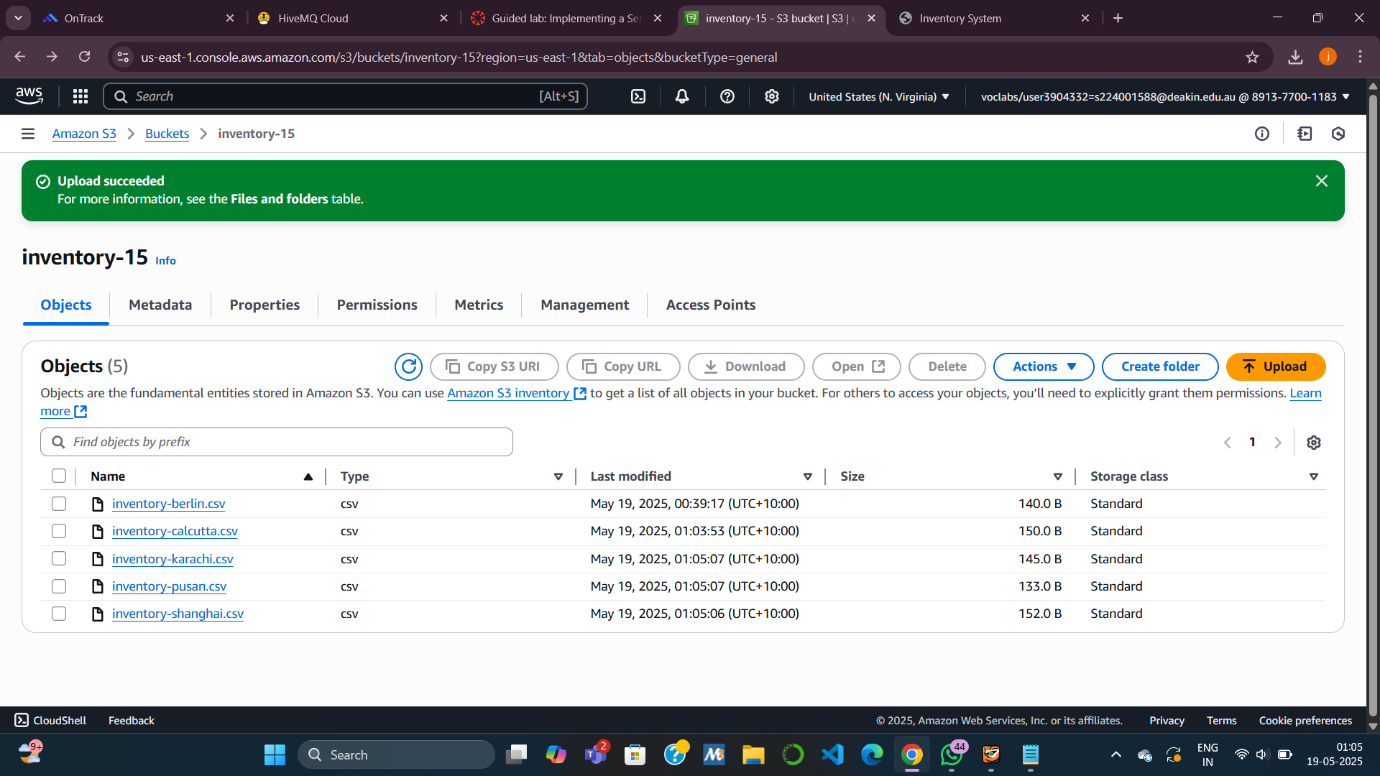
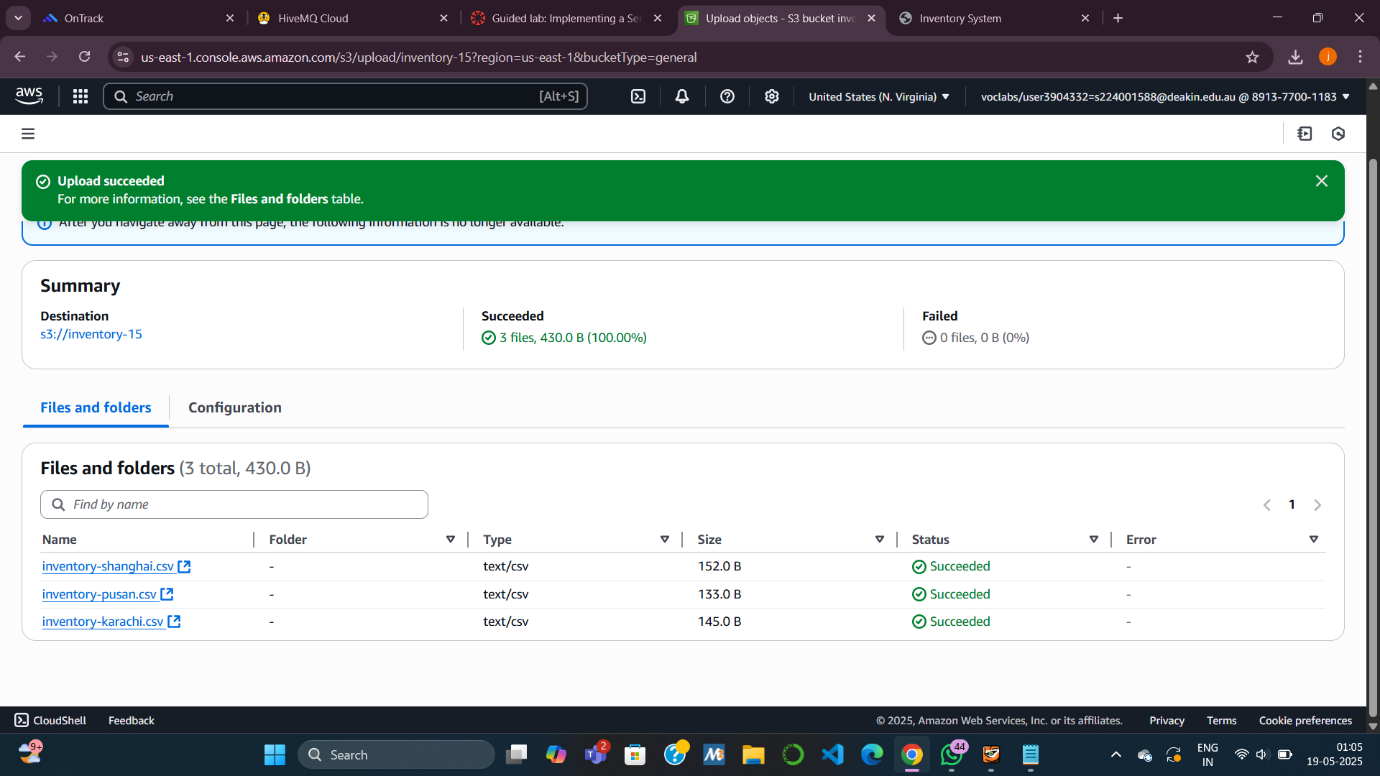
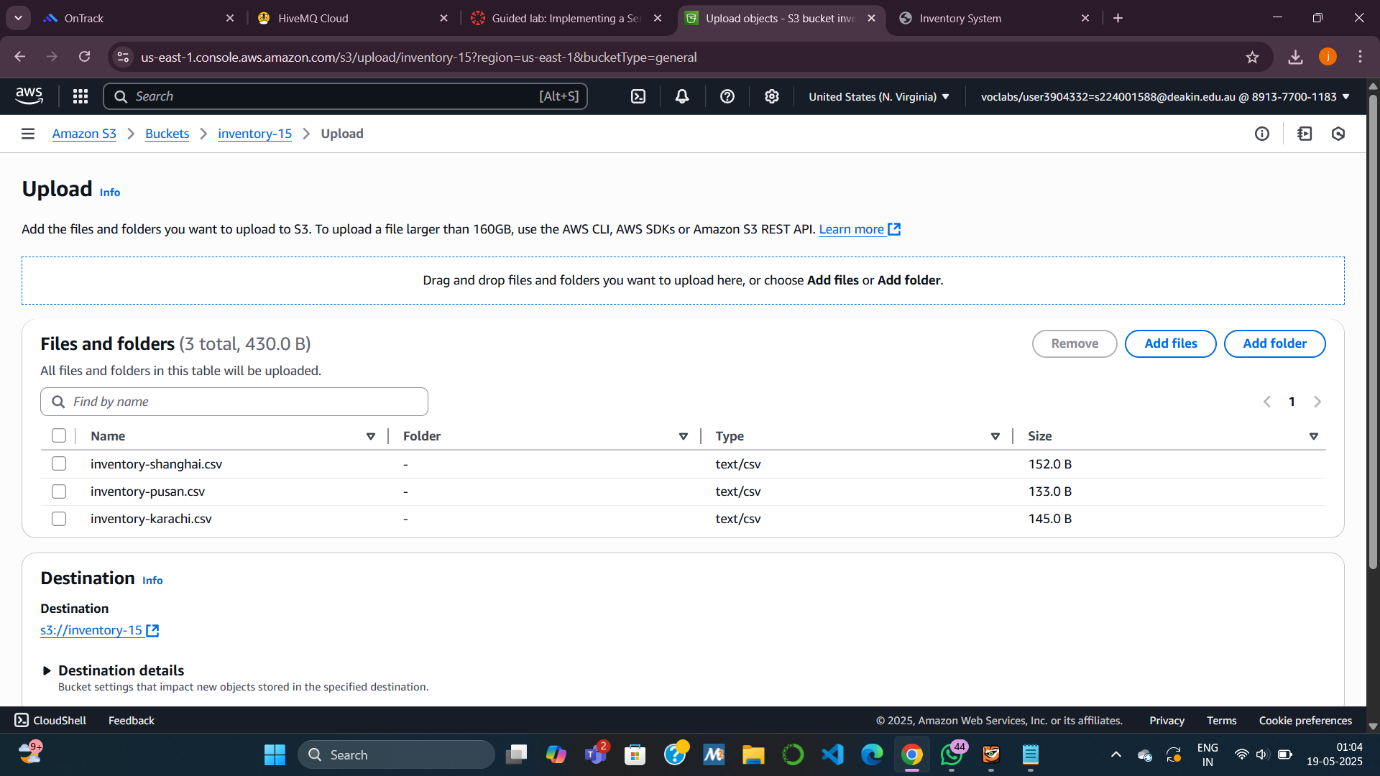
**Goal:** Notify via SNS when item count is 0 in the database.

* **Create Lambda function (Check-Stock)**:
  + Triggered by **DynamoDB Streams** when new item is added.
  + If count == 0, send alert to NoStock SNS topic.
* **Attach execution role**:  
  Lambda-Check-Stock-Role allows access to SNS.
* **Deploy and Add Trigger**:
  + Add DynamoDB Stream trigger on Inventory table.
  + Enables real-time stock checking and alerting.
* 
* 

**Task 6: Testing the System**

**Goal:** Validate the full flow (S3 → Lambda → DynamoDB → SNS).

* **Upload another CSV**:  
  Triggers full pipeline:
  + Data added to DynamoDB.
  + Check-Stock function invoked.
  + Sends email if count is 0.
* **Dashboard update**:  
  Refresh to see new store's inventory.
* **Check email**:  
  You should receive an "Out of Stock" alert.
* **Bonus:**  
  Try uploading **multiple files** together — all will trigger the process in parallel due to scalability of serverless architecture.
* 



**Answering questions about the S3 bucket**

**1. Was an S3 bucket created, even if you did not specify a name for the bucket? If so, what name was it given?**

**Yes**, an S3 bucket was created **even without explicitly specifying a name** in the CloudFormation template.

* When no name is provided, AWS automatically generates a **globally unique name** for the bucket in the format like:

update-cafe-app-bucket-1a2b3c4d5e6f

* This name includes the stack name and a unique string to ensure it doesn't conflict with existing bucket names globally.

**2. What Region was the bucket created in, and why was it created in this Region?**

The bucket was created in the **same Region where the CloudFormation stack was launched**.

* For example: us-east-1 (N. Virginia), ap-southeast-2 (Sydney), etc.
* It is created **in that Region by default** to ensure **low latency and compliance** with resource dependencies and regional policies.

**3. To define an S3 bucket, how many lines of code did you need to enter in the Resources: section of the template file?**

Typically, an S3 bucket definition in a YAML template looks like this:

Resources:

CafeS3Bucket:

Type: AWS::S3::Bucket

* This is **3 lines of code** under the Resources section.
* If you include additional properties (like versioning or lifecycle rules), more lines would be required, but a basic bucket needs just **3 lines**.

**Answering questions about the results of creating an application layer**

**4. Go to the *Parameters* tab of the update-cafe-app stack. What value do you see for the LatestAmiId?**

ami-0abcdef1234567890

* This AMI ID corresponds to the **latest Amazon Linux 2** or another base image being used for the EC2 instance in the application layer.
* This ID is **dynamically referenced** using an SSM parameter like:

LatestAmiId:

Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>

Default: /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2

**5. Go to the *Stack info* tab of the update-cafe-app stack. What is the Amazon Resource Name (ARN) of the IAM role that grants the permissions to create and update the stack?**

we will see an ARN similar to:

arn:aws:iam::123456789012:role/cafe-app-CloudFormationExecutionRole-ABC123XYZ

* This **IAM role** allows CloudFormation to assume necessary permissions (like creating EC2 instances, S3 buckets, etc.).
* It’s specified in the stack's execution role during creation.

**6. In the AWS Management Console, navigate to the CodeCommit repository where your AWS CloudFormation templates are stored. Choose *Commits*, and in the Commits list, open one of the commits by choosing its commit ID. What do you observe?**

Upon opening a commit, you will see:

* **Commit metadata**:
  + Commit ID
  + Commit message (e.g., “Updated EC2 instance type to t3.micro”)
  + Author and timestamp
* **Changed files**:
  + You can view which files were modified, such as template.yaml.
  + A **side-by-side diff** shows the exact line changes (additions, deletions).

**Observation**: This helps track template version history, document changes, and collaborate with others.