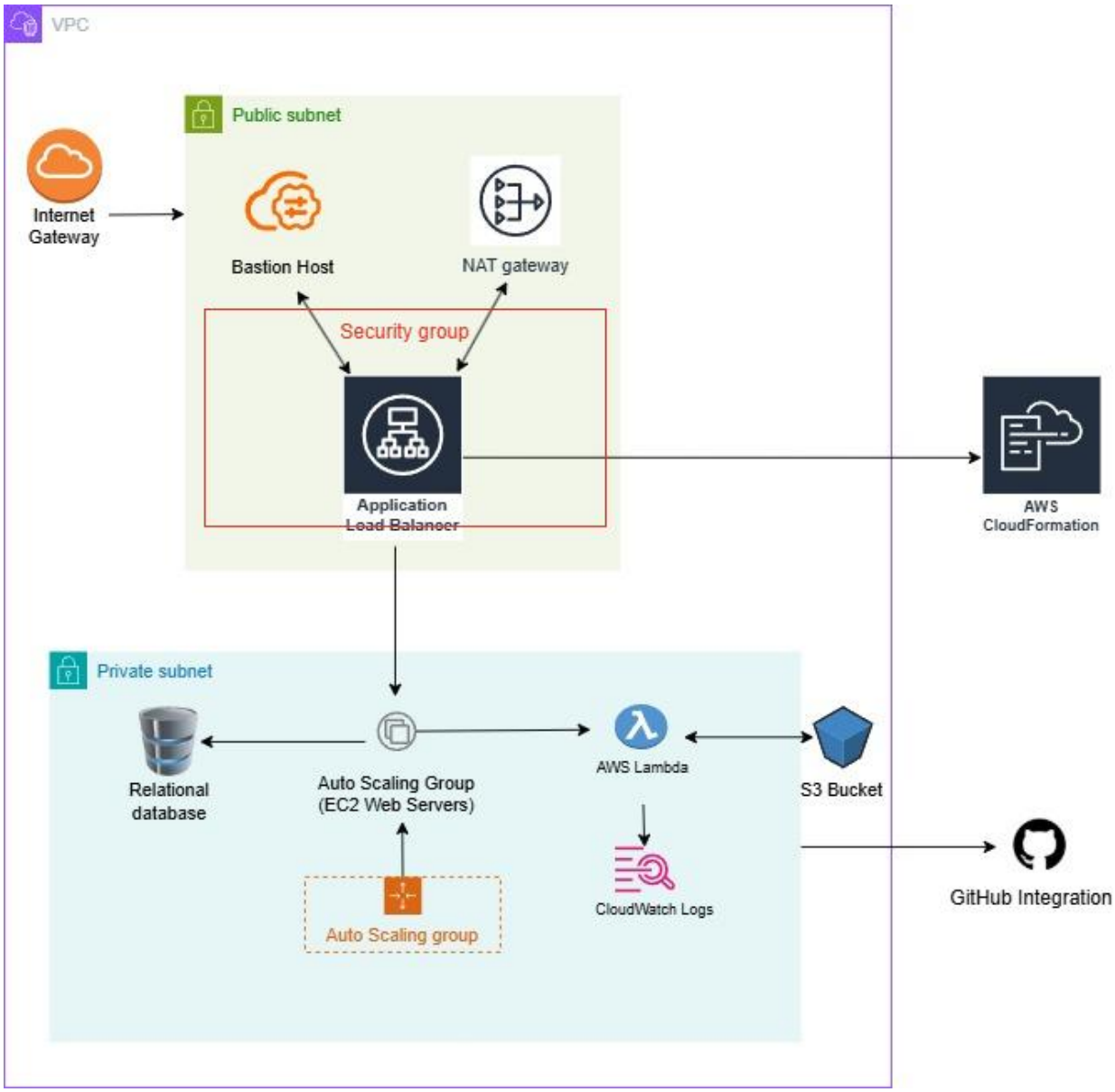
Project:

Tartela Tabassum
5/15/2025

1. Design an AWS Architecture

Students must design and **draw an architecture diagram** for their cloud application, which should include:

- **AWS VPC** with at least two subnets (public and private).
- **AWS EC2** instances behind an **Application Load Balancer (ALB)**.
- **Relational database** (RDS or MySQL/PostgreSQL on EC2).
- **S3 bucket** for storing files (e.g., logs, backups, static content).
- **AWS Lambda** to log S3 file uploads to **CloudWatch Logs**.
- **Autoscaling** for the web server layer.
- **AWS CloudFormation** to deploy infrastructure.
- **Security Group** configurations to control network access.
- **GitHub integration** for version control.



Architecture Diagram

The accompanying architecture diagram illustrates a secure, highly available AWS deployment housed within a single VPC that's split into a **public subnet**—which contains an Internet Gateway, NAT Gateway, Bastion Host, and an Application Load Balancer (ALB) guarded by its own ALB-SG—and a **private subnet** hosting an Auto Scaling group of EC2 web servers (protected by Web-SG) alongside an RDS relational database instance (behind DB-SG). All static assets, logs, and backups are stored in an S3 bucket; each **ObjectCreated** event there triggers a Lambda function (in Lambda-SG) that writes detailed entries to CloudWatch Logs. The entire stack is defined as code in AWS CloudFormation templates stored in GitHub, with GitHub Actions automatically validating and deploying updates on every push—ensuring

network isolation, autoscaling, fault tolerance, and end-to-end version-controlled infrastructure provisioning.

2. Implementation

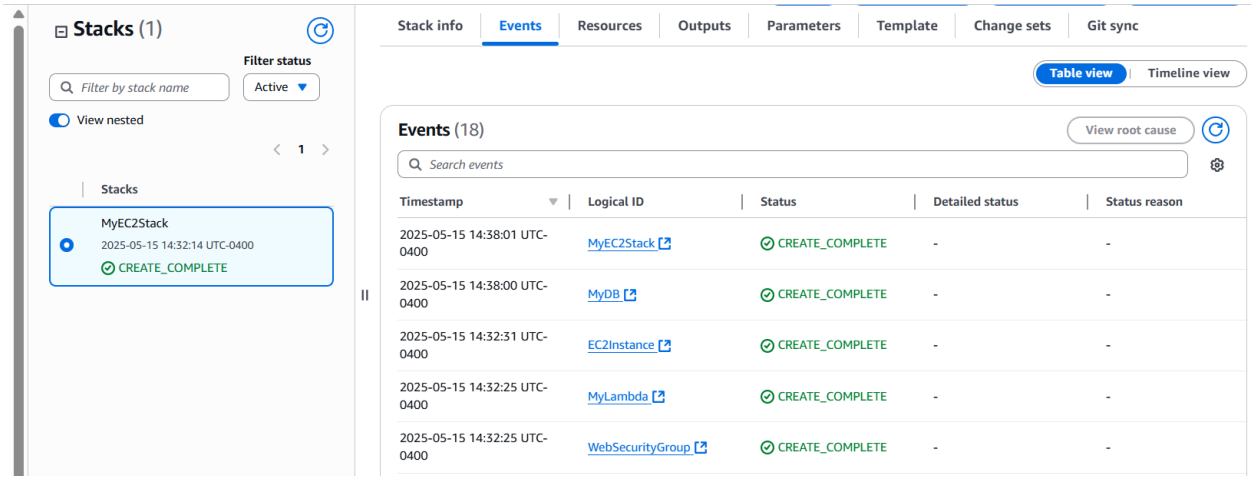
A. Infrastructure Deployment

- **Use Terraform** to create networking components (VPC, subnets, security groups).

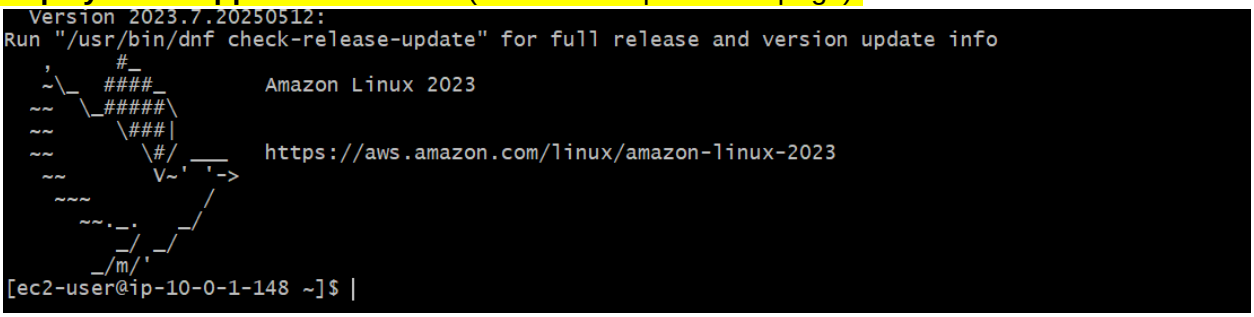
```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\User\Documents\terraform-ec2-lab\terraform_1.11.1_windows_amd64>
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.main: Creating...
aws_vpc.main: Creation complete after 1s [id=vpc-00093d985a3289efc]
aws_internet_gateway.gw: Creating...
aws_subnet.public: Creating...
aws_security_group.web_sg: Creating...
aws_internet_gateway.gw: Creation complete after 0s [id=igw-026f2e5cd349a6173]
aws_route_table.public: Creating...
aws_route_table.public: Creation complete after 1s [id=rtb-06e865bc356f20d38]
aws_security_group.web_sg: Creation complete after 3s [id=sg-024946b702581d6e5]
aws_subnet.public: Still creating... [10s elapsed]
aws_subnet.public: Creation complete after 11s [id=subnet-02985573b62ab9966]
aws_route_table_association.public_assoc: Creating...
aws_route_table_association.public_assoc: Creation complete after 0s [id=rtbassoc-0d8609ce18256e58d]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

public_subnet_id = "subnet-02985573b62ab9966"
security_group_id = "sg-024946b702581d6e5"
vpc_id = "vpc-00093d985a3289efc"
PS C:\Users\User\Documents\terraform-ec2-lab\terraform_1.11.1_windows_amd64>
```

- **Use CloudFormation** to deploy EC2 instances, RDS, and Lambda functions.



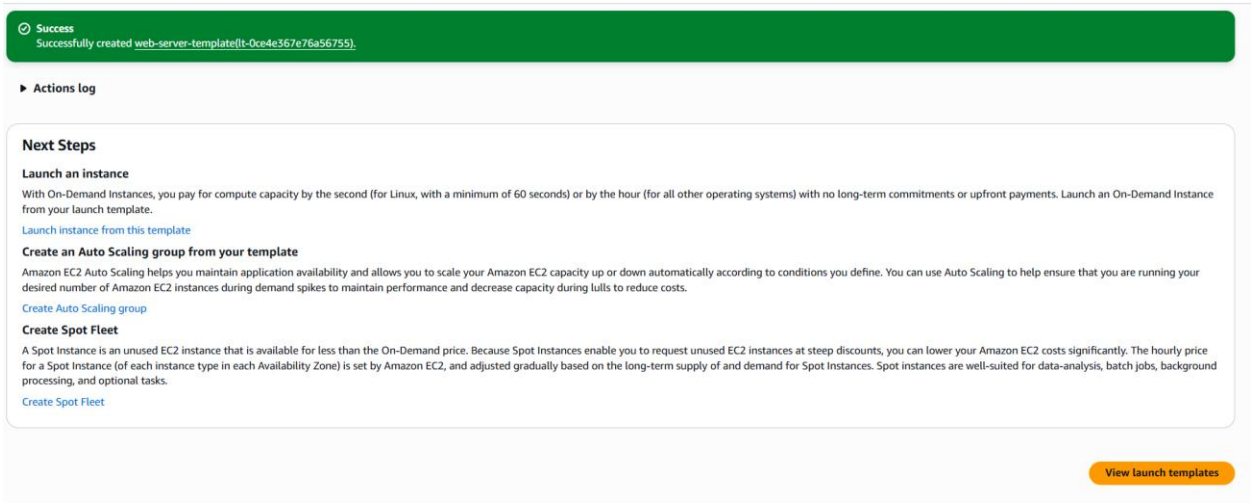
- **Deploy a web application** on EC2 (can be a simple HTML page).

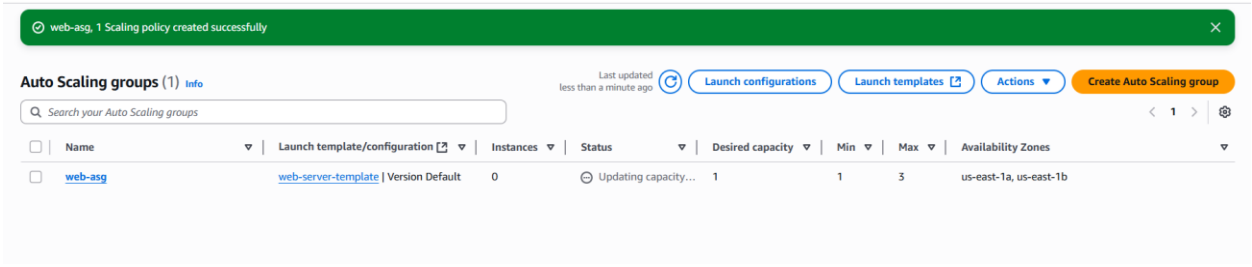


- **Configure the database** for the web application.

```
[ec2-user@ip-10-0-1-148 ~]$ sudo mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.5.23-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE myapp;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON myapp.* TO 'myuser'@'localhost';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.000 sec)

MariaDB [(none)]> EXIT;
Bye
[ec2-user@ip-10-0-1-148 ~]$
```

- **Implement autoscaling** to manage web server load.



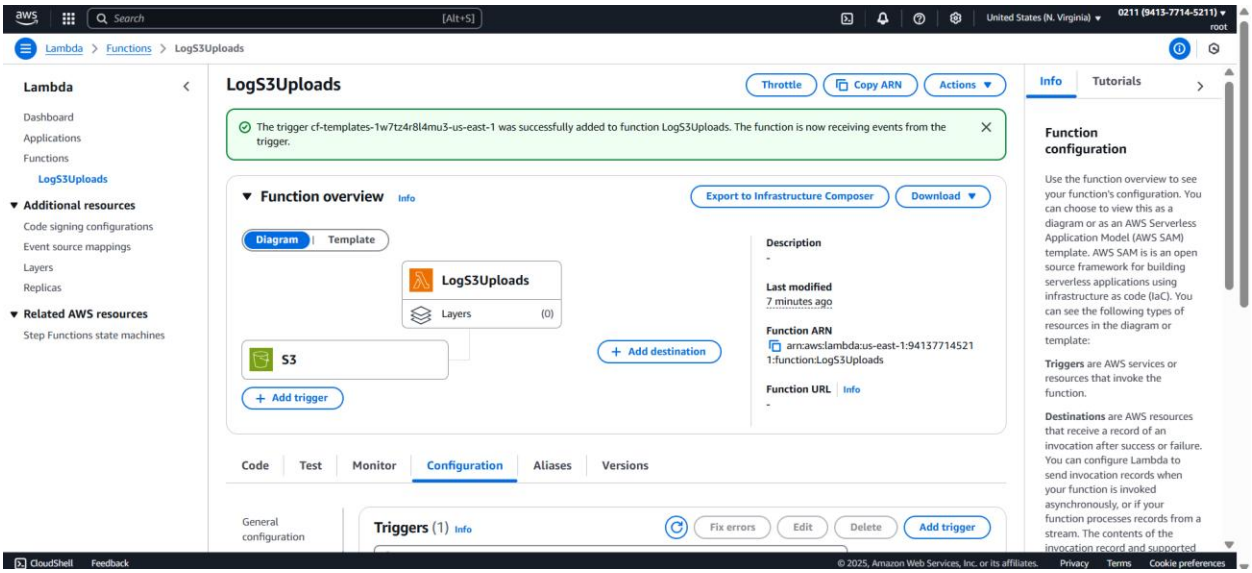


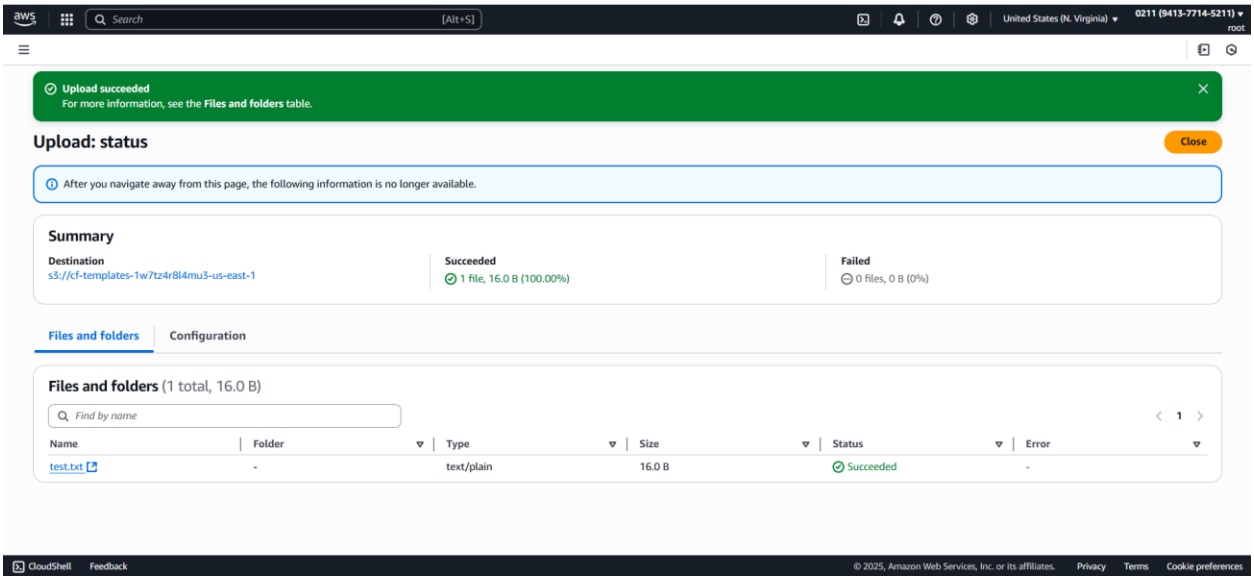
B. AWS Lambda for Logging S3 Uploads

- **Create an AWS Lambda function** that logs S3 file uploads to CloudWatch Logs.
- **Use the following Python script for the Lambda function:**

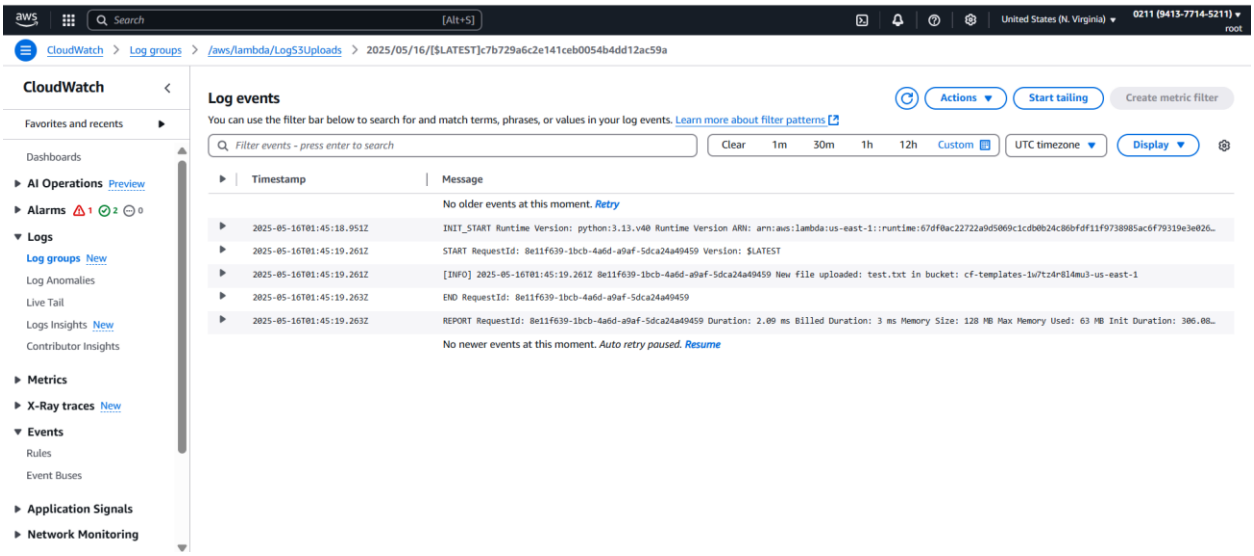
```
import json
import boto3
import logging

# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        # Extract bucket name and object key from the event
        bucket_name = event['Records'][0]['s3']['bucket']['name']
        object_key = event['Records'][0]['s3']['object']['key']

        # Log the upload event
        log_message = f"New file uploaded: {object_key} in bucket:
{bucket_name}"
        logger.info(log_message)

        return {
            "statusCode": 200,
            "body": json.dumps(log_message)
        }

    except Exception as e:
        logger.error(f"Error processing S3 event: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps(f"Error: {str(e)}")
        }
```

- **Set up an S3 event trigger** so the function executes whenever a new file is uploaded.



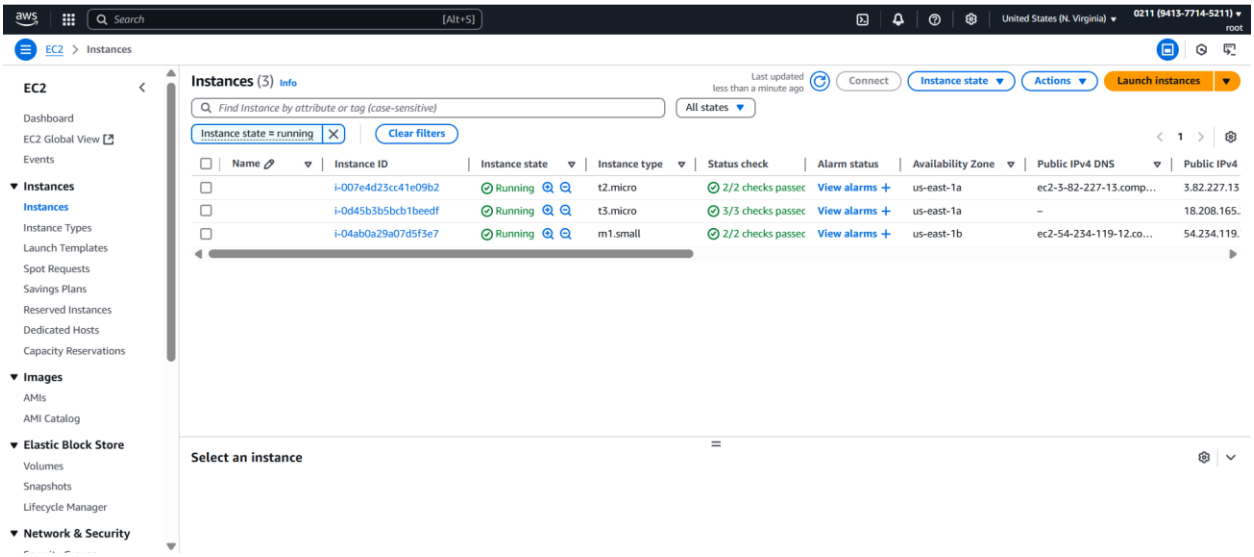


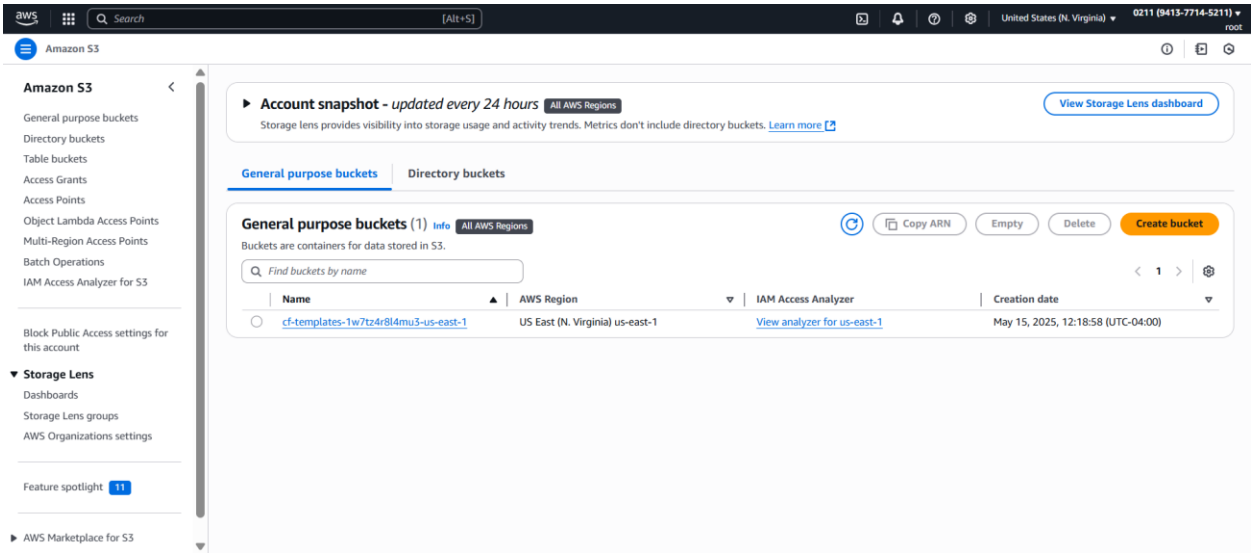
- **Verify logs in CloudWatch** to ensure it captures the uploaded file's name and bucket details.

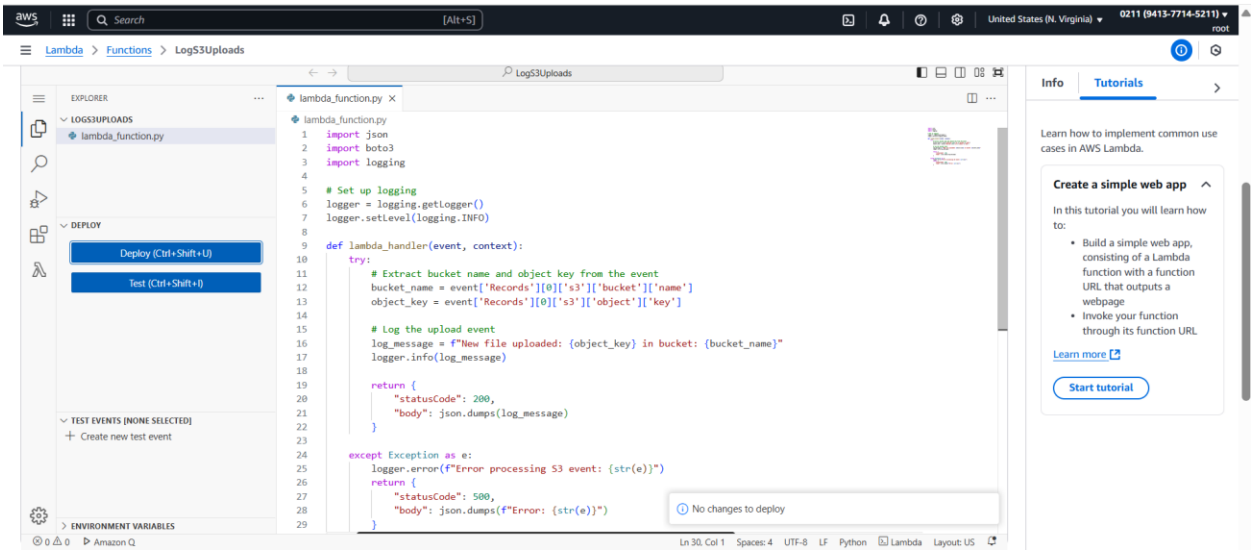


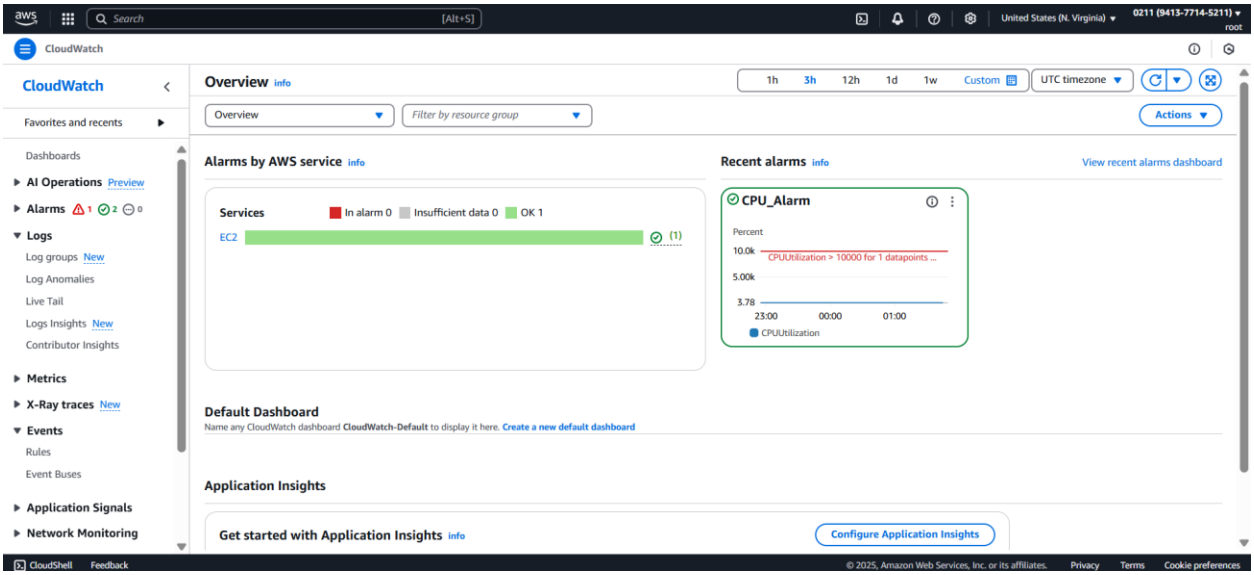
C. Interaction with AWS

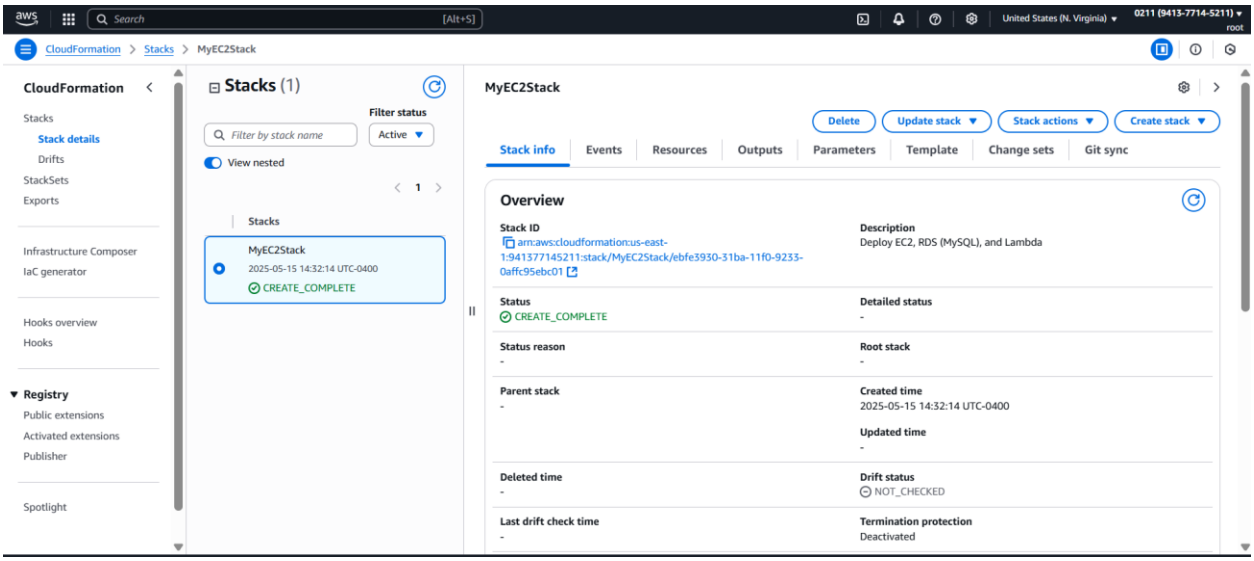
- **Use AWS Console** to verify infrastructure deployment.




Search
[Alt+S]
United States (N. Virginia)
0211 (9413-7714-5211)
root
Amazon S3
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3
Account snapshot - updated every 24 hours
All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info
All AWS Regions
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
cf-templates-1w7tz4r8l4mu3-us-east-1
US East (N. Virginia) us-east-1
View analyzer for us-east-1
May 15, 2025, 12:18:58 (UTC-04:00)


Search
[Alt+S]
United States (N. Virginia)
0211 (9413-7714-5211)
root
Lambda
Functions
LogS3Uploads
LogS3Uploads
EXPLORER
LOGS3UPLOADS
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
import json
import boto3
import logging
# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
try:
# Extract bucket name and object key from the event
bucket_name = event['Records'][0]['s3']['bucket']['name']
object_key = event['Records'][0]['s3']['object']['key']
# Log the upload event
log_message = f"New file uploaded: {object_key} in bucket: {bucket_name}"
logger.info(log_message)
return {
"statusCode": 200,
"body": json.dumps(log_message)
}
except Exception as e:
logger.error(f"Error processing S3 event: {str(e)}")
return {
"statusCode": 500,
"body": json.dumps(f"Error: {str(e)}")
}
No changes to deploy
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
Lambda
Layout: US
Info
Tutorials
Learn how to implement common use cases in AWS Lambda.
Create a simple web app
In this tutorial you will learn how to:
Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
Invoke your function through its function URL
Learn more
Start tutorial





- **Use AWS CLI** to interact with EC2, S3, and Lambda.

```
User@DESKTOP-AQNR9SO MINGW64 ~
$ aws ec2 describe-instances \
  --instance-ids i-0d45b3b5bcb1beedf \
  --query 'Reservations[0].Instances[0].PublicIpAddress' \
  --output text
18.208.165.243
```

```
User@DESKTOP-AQNR9SO MINGW64 ~
$ aws s3 ls
2025-05-15 12:18:58 cf-templates-1w7tz4r8l4mu3-us-east-1
```

```
User@DESKTOP-AQNR9SO MINGW64 ~
$ aws lambda invoke \
  --function-name LogS3Uploads \
  --payload '{}' \
  output.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

- **Write Python scripts using Boto3** to:
  - Create an S3 bucket and upload a file.

```
PS C:\Users\User\Downloads> python create_s3_and_upload.py
File test.txt uploaded to bucket tartela-bucket
PS C:\Users\User\Downloads>
```

  - Retrieve EC2 instance metadata.

```
ec2.py  metadata.py
[ec2-user@ip-10-0-1-148 ~]$ nano metadata.py
[ec2-user@ip-10-0-1-148 ~]$ python3 metadata.py
i-0d45b3b5bcb1beedf:
t3.micro:
18.208.165.243:
ami-000d41719f4ab3d23:
[ec2-user@ip-10-0-1-148 ~]$
```

  - List running EC2 instances.

```
PS C:\Users\User\Downloads> python list_running_ec2.py
Instance ID: i-007e4d23cc41e09b2, Public IP: 3.82.227.13
Instance ID: i-0d45b3b5bcb1beedf, Public IP: 18.208.165.243
Instance ID: i-04ab0a29a07d5f3e7, Public IP: 54.234.119.12
PS C:\Users\User\Downloads>
```

- Invoke the AWS Lambda function manually.