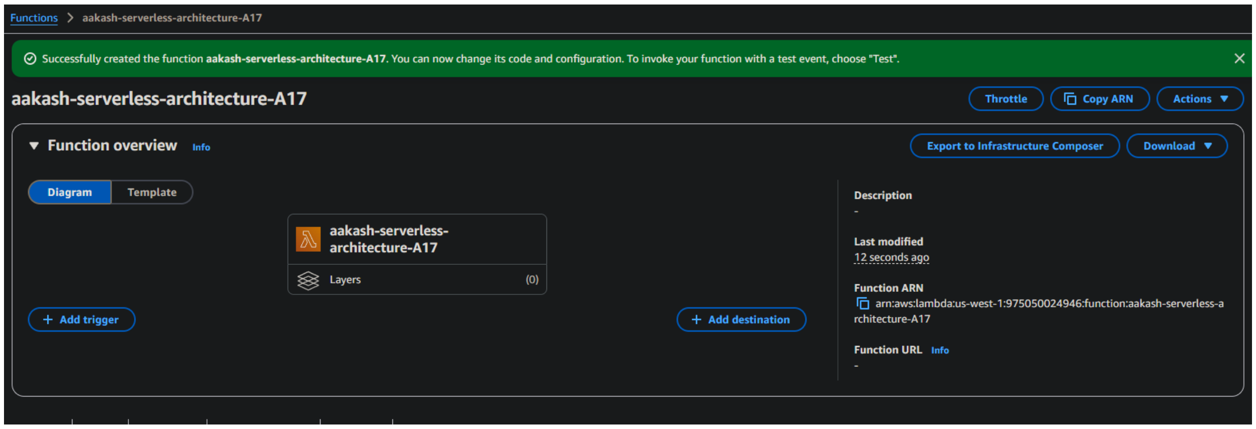
# Graded Assignment on Serverless Architecture

Assignment 17: Restore EC2 Instance from Snapshot

Objective: Automate the process of creating a new EC2 instance from the latest snapshot using a Lambda function.

## Create a Lambda function



Successfully created the function **aakash-serverless-architecture-A17**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

aakash-serverless-architecture-A17

aakash-serverless-architecture-A17

Description
-

Last modified
12 seconds ago

Function ARN
arn:aws:lambda:us-west-1:975050024946:function:aakash-serverless-architecture-A17

Function URL
-

## Boto3 Function

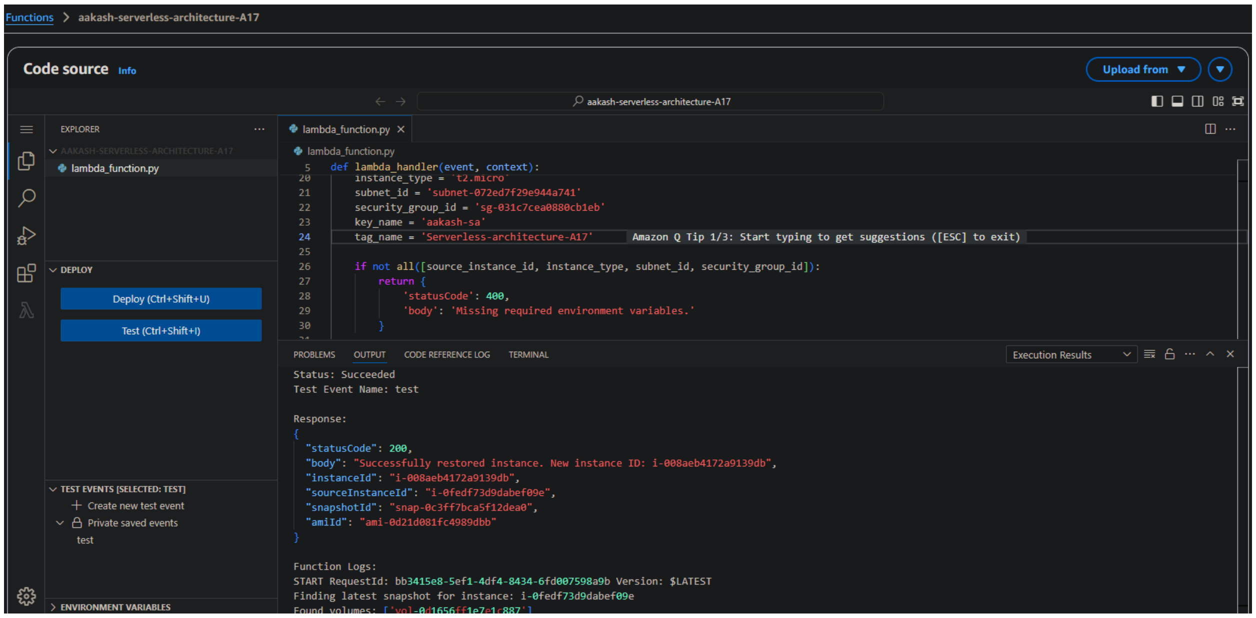
Restore EC2 Instance from Snaps

```
import boto3
import os
import datetime

def lambda_handler(event, context):
    """
    Lambda function to restore an EC2 instance from the latest snapshot of a
given instance.
```

```python
    Environment Variables:
    - SOURCE_INSTANCE_ID: The ID of the original EC2 instance (e.g., i-
1234567890abcdef0)
    - INSTANCE_TYPE: The type of the new EC2 instance (e.g., t2.micro)
    - SUBNET_ID: The subnet ID where the new instance will be launched
    - SECURITY_GROUP_ID: The security group ID to attach to the new instance
    - KEY_NAME: (Optional) The key pair name to use for the new instance
    - TAG_NAME: The name tag for the new EC2 instance
    """

    # Get environment variables
    source_instance_id = 'i-0fedf73d9dabef09e'
    instance_type = 't2.micro'
    subnet_id = 'subnet-072ed7f29e944a741'
    security_group_id = 'sg-031c7cea0880cb1eb'
    key_name = 'aakash-sa'
    tag_name = 'Serverless-architecture-A17'

    if not all([source_instance_id, instance_type, subnet_id,
security_group_id]):
        return {
            'statusCode': 400,
            'body': 'Missing required environment variables.'
        }

    # Initialize AWS clients
    ec2_client = boto3.client('ec2')
    ec2_resource = boto3.resource('ec2')

    try:
        # Step 1: Get the latest snapshot for the source instance
        print(f"Finding latest snapshot for instance: {source_instance_id}")

        # First, identify all volumes attached to the source instance
        instance_volumes = []
        instance_response =
ec2_client.describe_instances(InstanceIds=[source_instance_id])

        for reservation in instance_response['Reservations']:
            for instance in reservation['Instances']:
                for block_device in instance['BlockDeviceMappings']:
                    if 'Ebs' in block_device:
                        instance_volumes.append(block_device['Ebs']['VolumeId'])

        if not instance_volumes:
            return {
                'statusCode': 404,
                'body': f'No volumes found for instance {source_instance_id}'
            }

        print(f"Found volumes: {instance_volumes}")

        # Find snapshots associated with these volumes
        latest_snapshot = None
        latest_snapshot_time = datetime.datetime(1970, 1, 1,
tzinfo=datetime.timezone.utc)

        for volume_id in instance_volumes:
            snapshots_response = ec2_client.describe_snapshots(
                Filters=[
                    {
                        'Name': 'volume-id',
                        'Values': [volume_id]
                    },
                    {
                        'Name': 'status',
                        'Values': ['completed']
                    }
                ]
            )

            for snapshot in snapshots_response['Snapshots']:
                if snapshot['StartTime'] > latest_snapshot_time:
                    latest_snapshot_time = snapshot['StartTime']
                    latest_snapshot = snapshot['SnapshotId']

        if not latest_snapshot:
            return {
                'statusCode': 404,
                'body': f'No snapshots found for instance {source_instance_id}'
            }

        print(f"Latest snapshot found: {latest_snapshot}, created at
{latest_snapshot_time}")

        # Step 2: Create an AMI from the snapshot
        snapshot = ec2_resource.Snapshot(latest_snapshot)

        # Get snapshot details
```

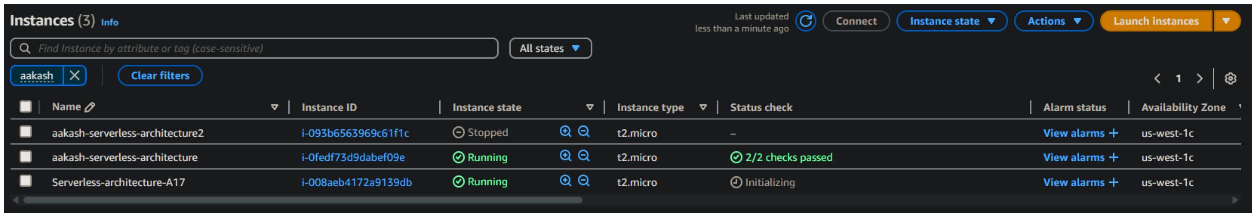
```python
        snapshot_description = 
ec2_client.describe_snapshots(SnapshotIds=[latest_snapshot])
        volume_size = snapshot_description['Snapshots'][0]['VolumeSize']

        # Create an AMI from the snapshot
        current_time = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
        ami_name = f"restored-{source_instance_id}-{current_time}"

        ami_response = ec2_client.register_image(
            Name=ami_name,
            Architecture='x86_64',  # Adjust as needed
            RootDeviceName='/dev/sda1',
            BlockDeviceMappings=[
                {
                    'DeviceName': '/dev/sda1',
                    'Ebs': {
                        'SnapshotId': latest_snapshot,
                        'VolumeSize': volume_size,
                        'DeleteOnTermination': True,
                        'VolumeType': 'gp2'
                    }
                }
            ],
            VirtualizationType='hvm'
        )

        ami_id = ami_response['ImageId']
        print(f"Created AMI: {ami_id}")

        # Wait for the AMI to be available
        waiter = ec2_client.get_waiter('image_available')
        waiter.wait(ImageIds=[ami_id])
        print(f"AMI {ami_id} is now available")

        # Step 3: Launch a new instance using the AMI
        run_instances_args = {
            'ImageId': ami_id,
            'InstanceType': instance_type,
            'MaxCount': 1,
            'MinCount': 1,
            'SubnetId': subnet_id,
            'SecurityGroupIds': [security_group_id],
            'TagSpecifications': [
                {
                    'ResourceType': 'instance',
                    'Tags': [
                        {
                            'Key': 'Name',
                            'Value': tag_name
                        },
                        {
                            'Key': 'SourceInstance',
                            'Value': source_instance_id
                        },
                        {
                            'Key': 'RestoreDate',
                            'Value': current_time
                        }
                    ]
                }
            ]
        }

        # Add key pair if provided
        if key_name:
            run_instances_args['KeyName'] = key_name

        new_instance = ec2_client.run_instances(**run_instances_args)

        new_instance_id = new_instance['Instances'][0]['InstanceId']
        print(f"Launched new instance: {new_instance_id}")

        return {
            'statusCode': 200,
            'body': f'Successfully restored instance. New instance ID: 
{new_instance_id}',
            'instanceId': new_instance_id,
            'sourceInstanceId': source_instance_id,
            'snapshotId': latest_snapshot,
            'amiId': ami_id
        }

    except Exception as e:
        print(f"Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': f'Error creating instance from snapshot: {str(e)}'
        }
```

# Trigger this Lambda function



```
def lambda_handler(event, context):
    instance_type = 't2.micro'
    subnet_id = 'subnet-072ed7f29e944a741'
    security_group_id = 'sg-031c7cea0880cb1eb'
    key_name = 'aakash-sa'
    tag_name = 'Serverless-architecture-A17'    Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)

    if not all([source_instance_id, instance_type, subnet_id, security_group_id]):
        return {
            'statusCode': 400,
            'body': 'Missing required environment variables.'
        }
```

```
Status: Succeeded
Test Event Name: test

Response:
{
  "statusCode": 200,
  "body": "Successfully restored instance. New instance ID: i-008aeb4172a9139db",
  "instanceId": "i-008aeb4172a9139db",
  "sourceInstanceId": "i-0fedf73d9dabef09e",
  "snapshotId": "snap-0c3ff7bca5f12dea0",
  "amiId": "ami-0d21d081fc4989dbb"
}

Function Logs:
START RequestId: bb3415e8-5ef1-4df4-8434-6fd007598a9b Version: $LATEST
Finding latest snapshot for instance: i-0fedf73d9dabef09e
Found volumes: ['vol-0d1656ff1e7e1c887']
```



| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone |
|---|---|---|---|---|---|---|
| aakash-serverless-architecture2 | i-093b6563969c61f1c | Stopped | t2.micro | – | View alarms + | us-west-1c |
| aakash-serverless-architecture | i-0fedf73d9dabef09e | Running | t2.micro | 2/2 checks passed | View alarms + | us-west-1c |
| Serverless-architecture-A17 | i-008aeb4172a9139db | Running | t2.micro | Initializing | View alarms + | us-west-1c |