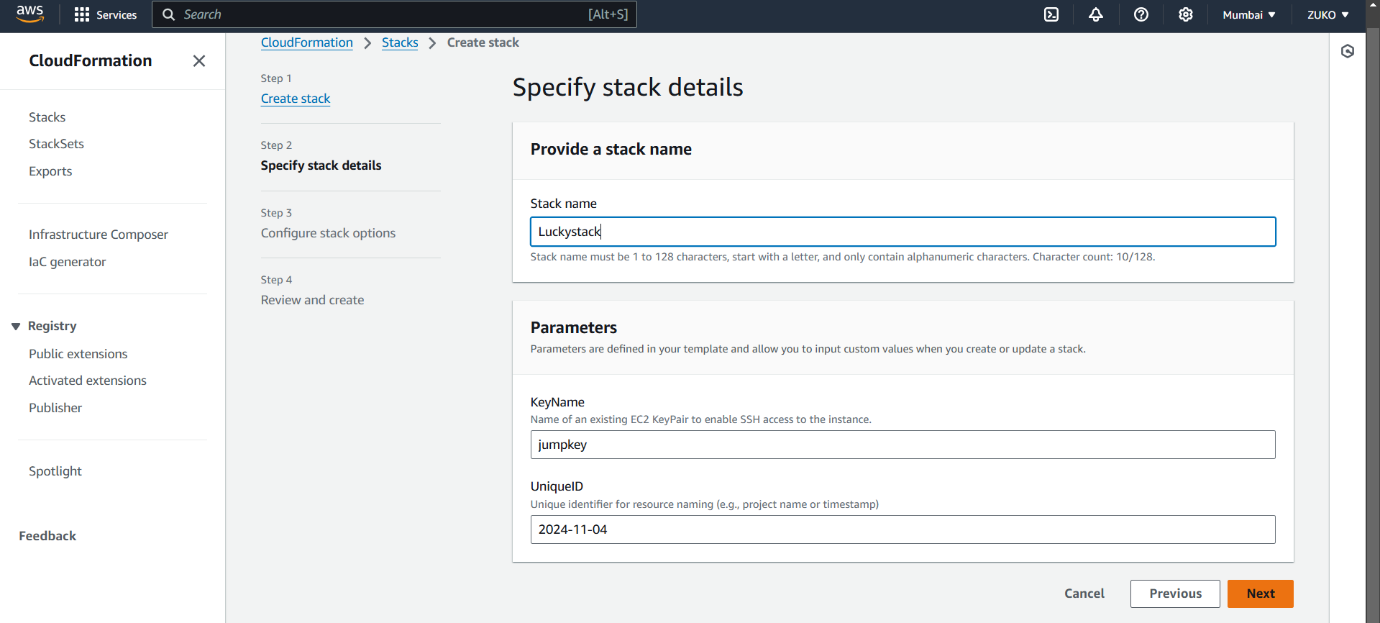
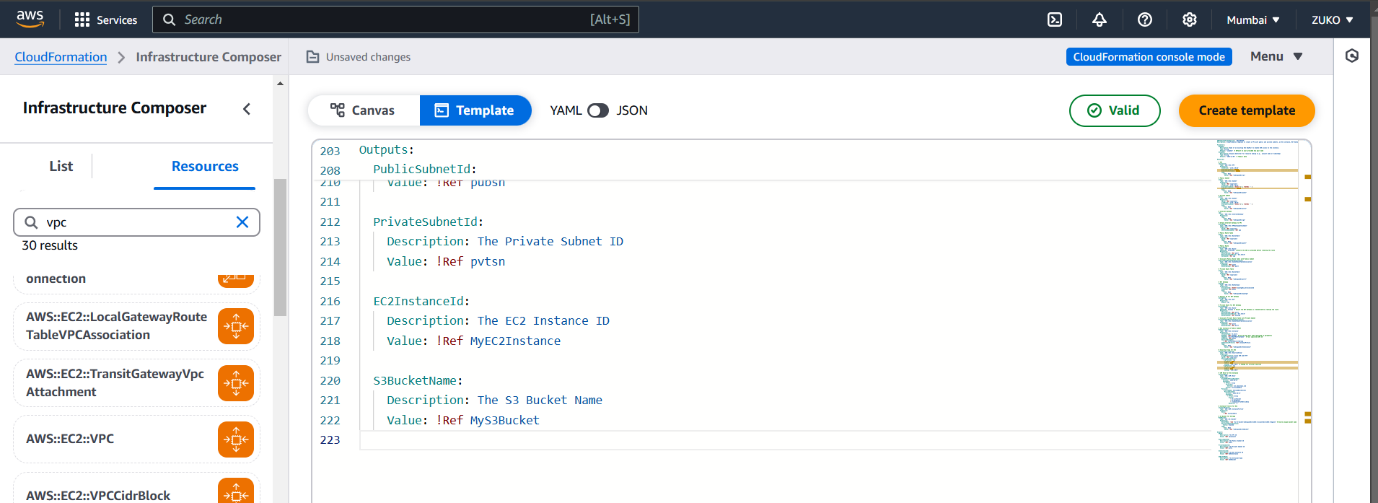
**TASK 17- MANAGING INFRASTRUCTURES USING AWS CFT**

AWS CLOUDFORMATION IS AN IAAC TOOL THAT ACTS AS AN INTERFACE BETWEEN THE USER AND THE CLOUD USING CODE TEMPLATES WRITTEN IN JSON OR YAML WHICH TRIGGERS APICALLS TO AWS TO UNDERSTAND THE REQUEST OF THE USER. TEMPLATES SHOULD BE DECLARATIVE AND VERSIONED.

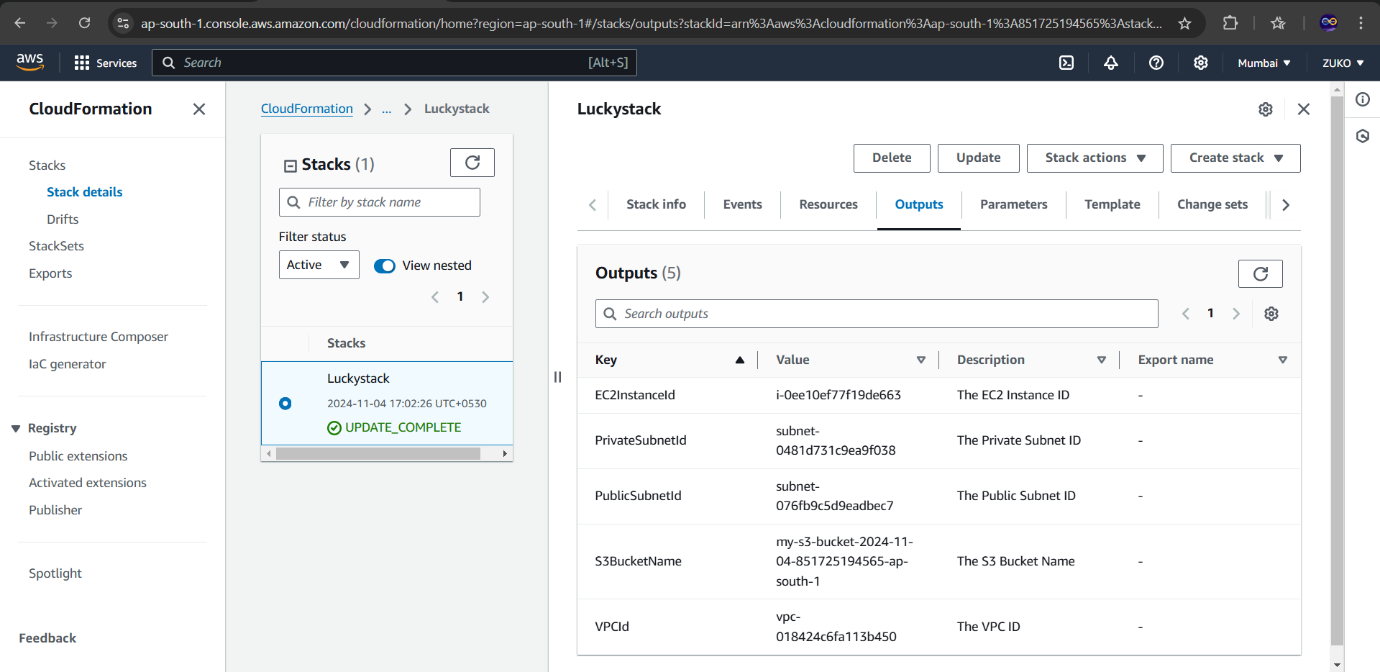
Step1- CREATED A STACK ON AWS CFT



STEP2- COMPOSED CODE TEMPLATE AND VALIDATED



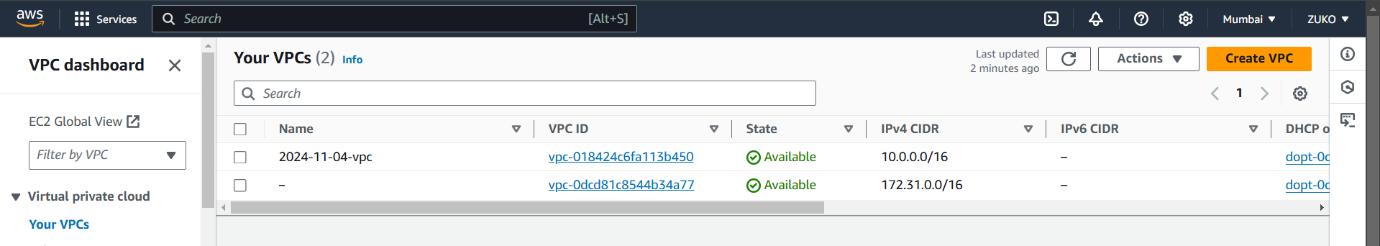
STACK UPDATED



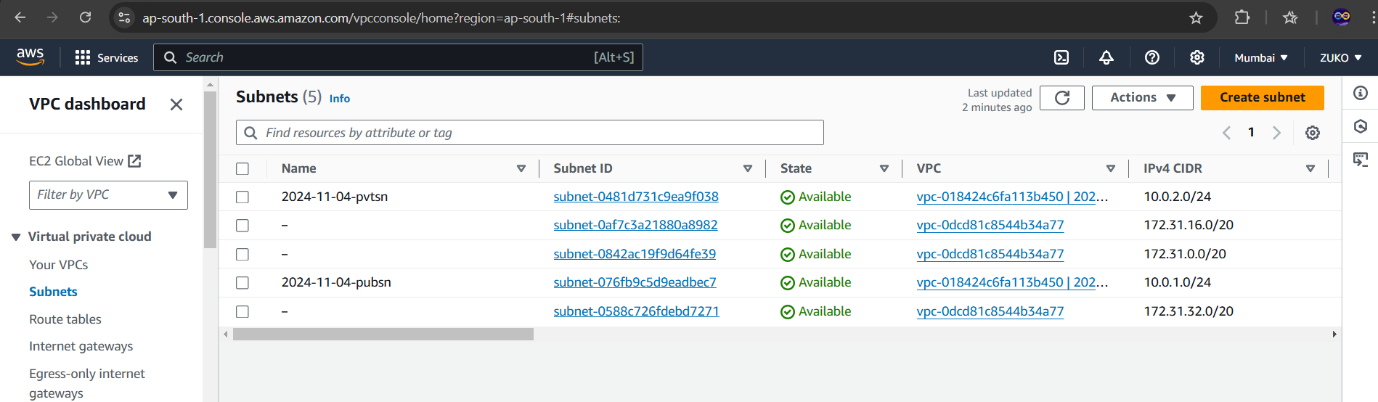
STEP3- VPC CREATED ALONG WITH PUB/PVT SUBNETS, PUB/PVT ROUTE TABLES, NAT GATEWAY, EIP, IGW,

IAM ROLE, EC2, S3.

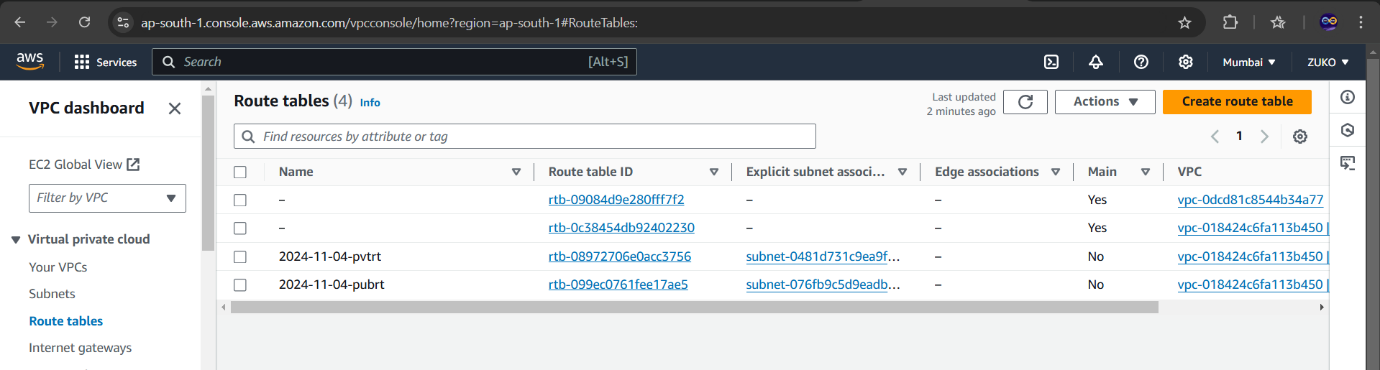
(VPC)



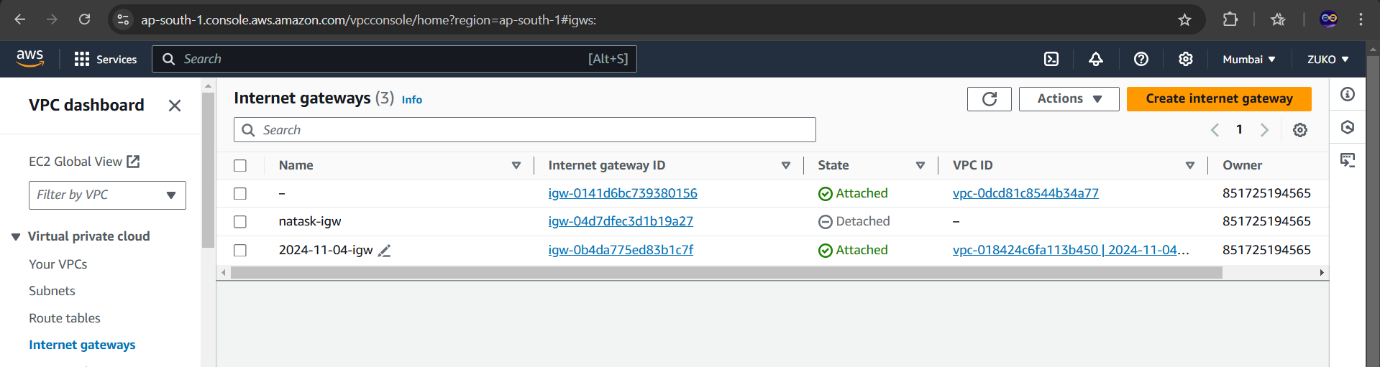
SUBNETS



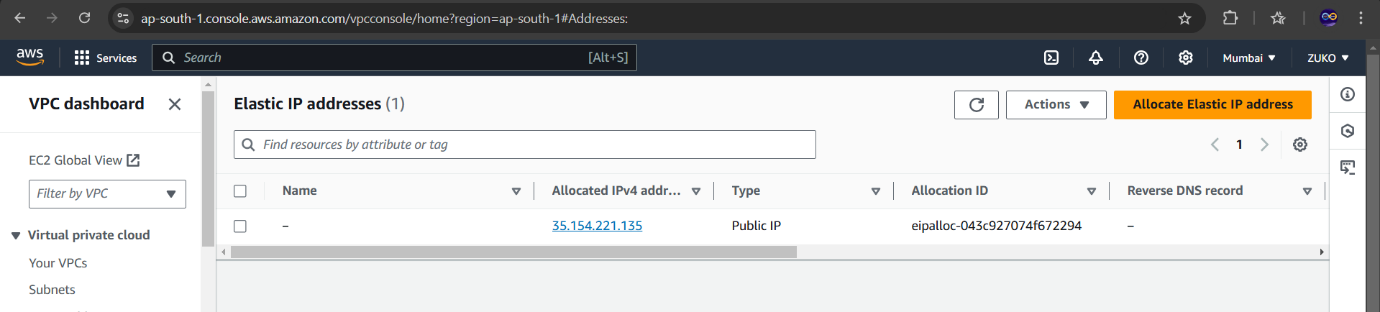
ROUTE TABLES



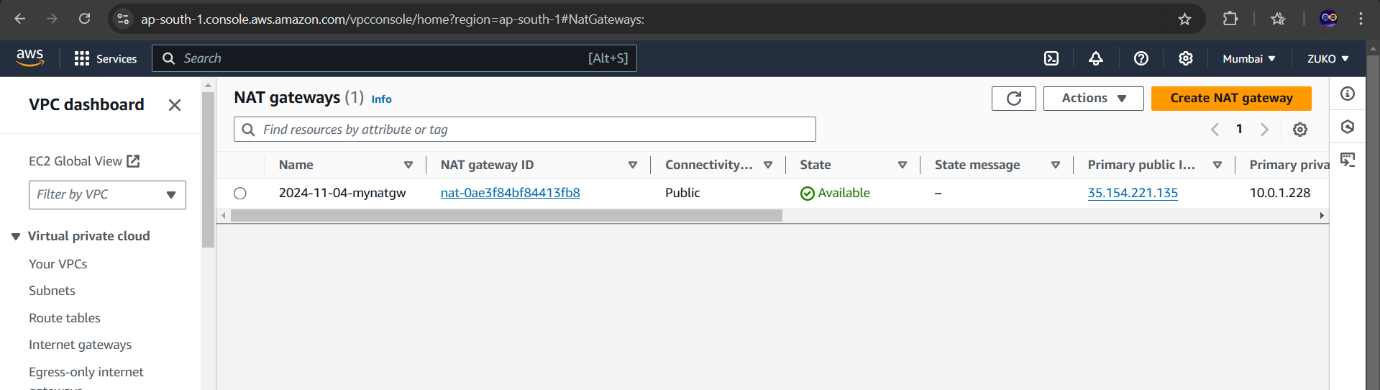
IGW



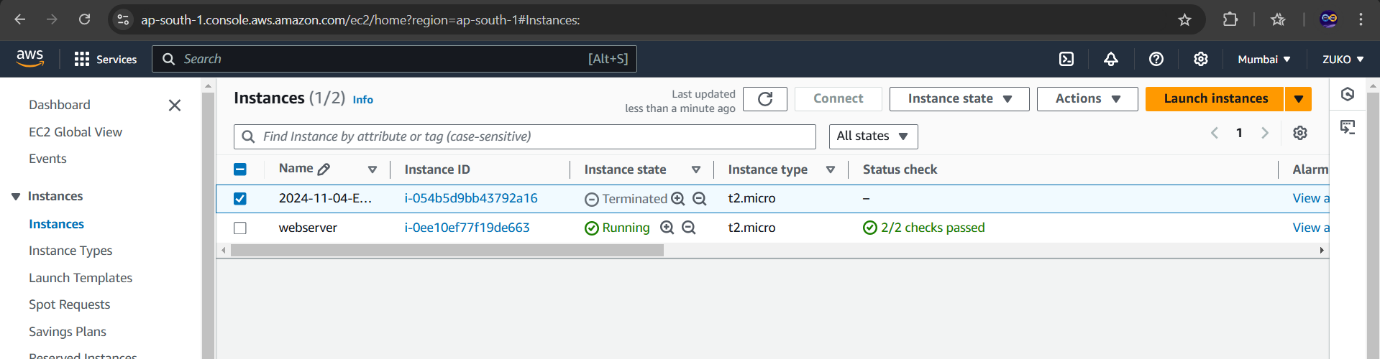
EIP



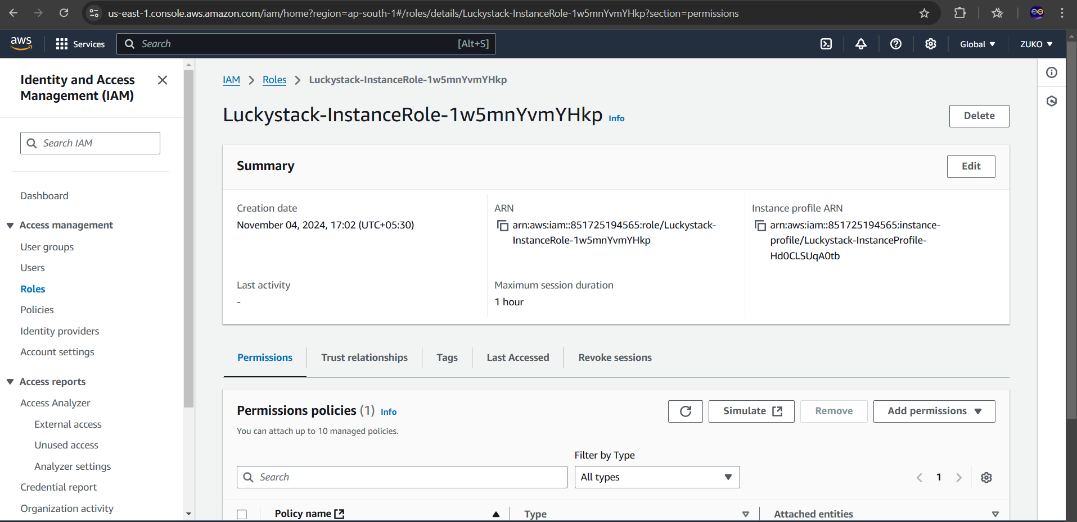
NATGW



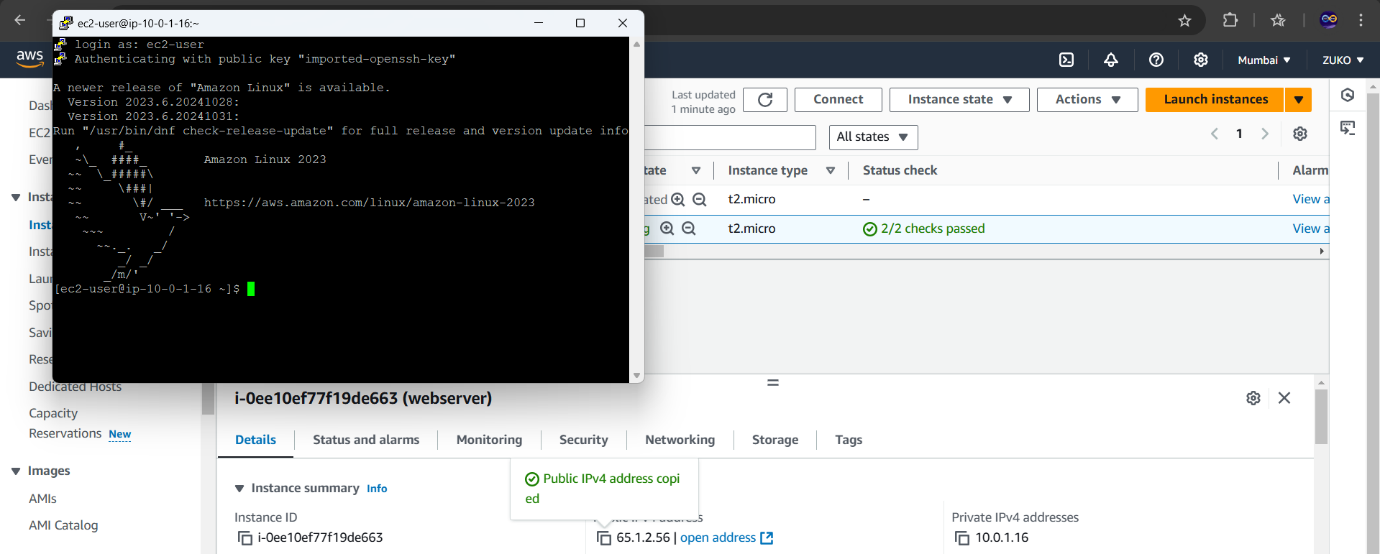
EC2



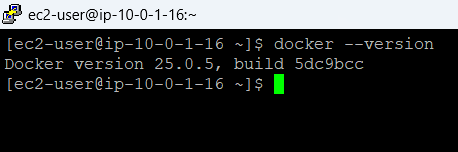
IAM ROLE



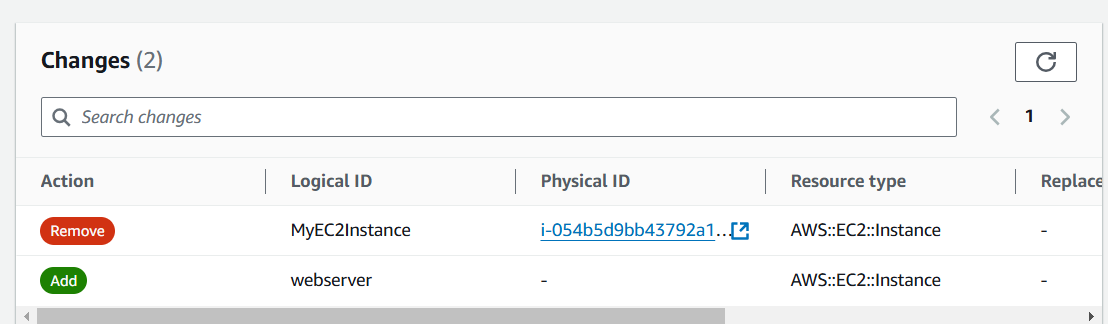
EC2 LAUNCHED



Internet connection was checked and I installed docker on the webserver instance. I also tried updating the code template to change the name of EC2 instance, but AWS CFT terminated the existing EC2 and launched a new instance with the declared name.



CHANGES

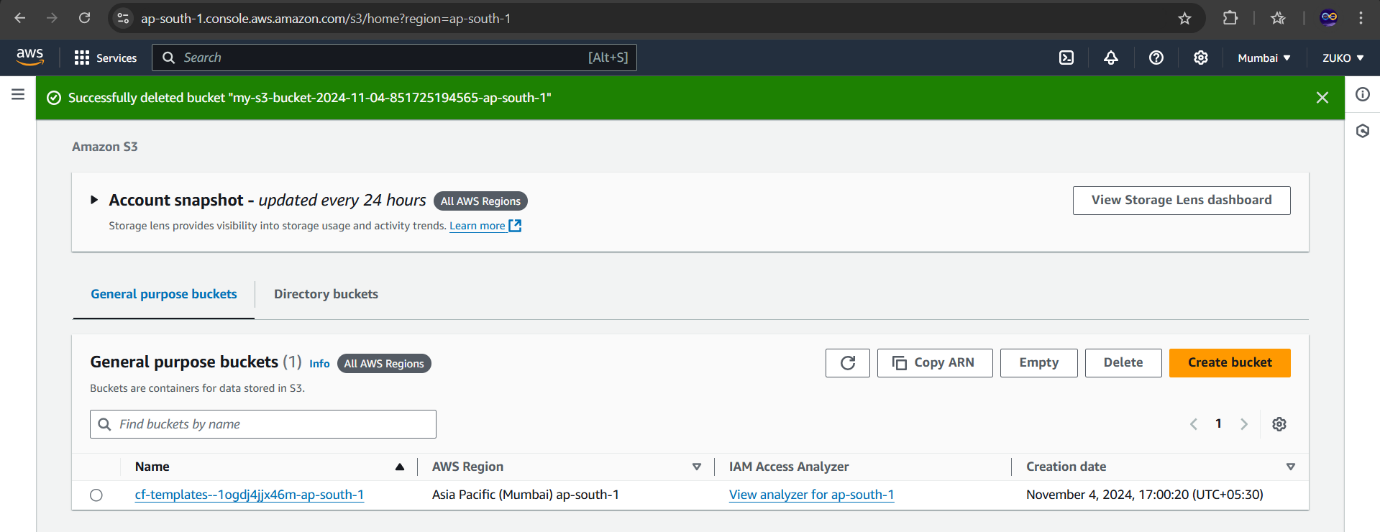


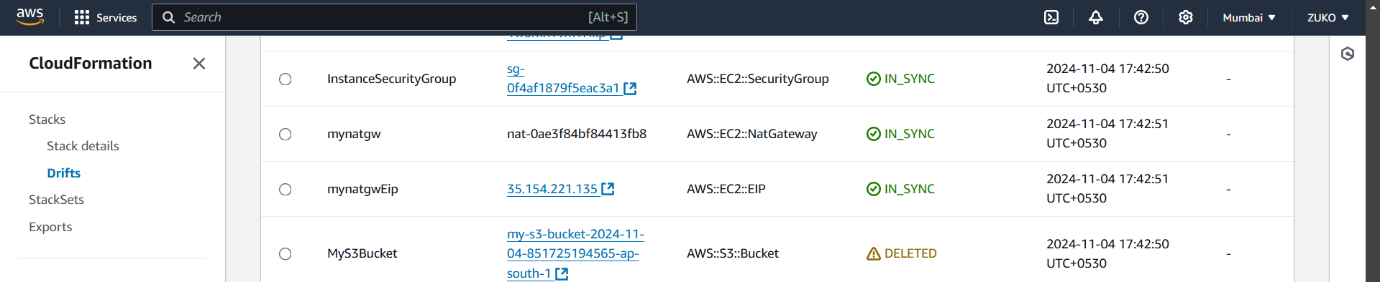
DRIFT DETECTION

Drift Detection in AWS CloudFormation is a feature that identifies changes, or "drift," between the actual configuration of resources in your AWS environment and the expected configuration defined in your CloudFormation template. Drift occurs when resources are manually modified outside of CloudFormation, creating a state that differs from what is specified in the stack template.

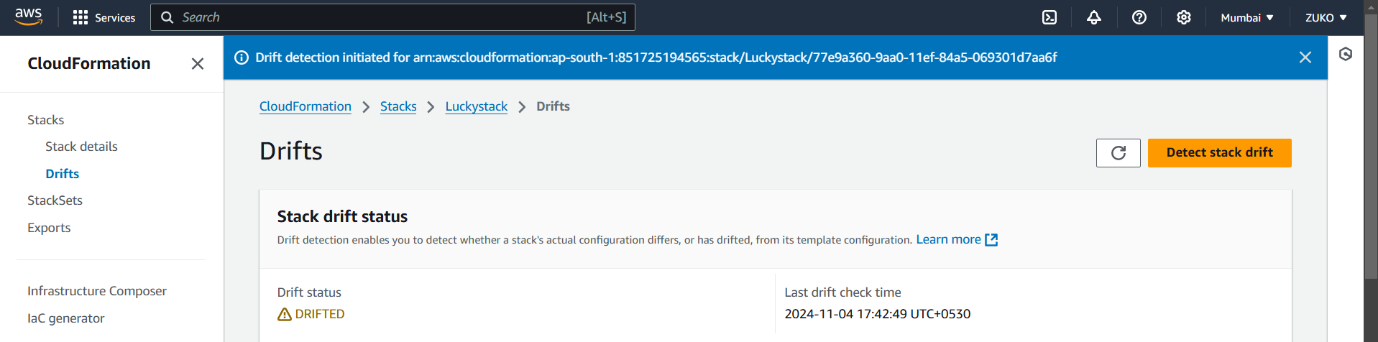
1. Responding to Drift:
   * If drift is detected, you have a few options:
     + Update the Stack: Modify the template to match the actual configuration or revert the manual changes to align with the original template.
     + Rollback Changes: Use CloudFormation to reset resources to the desired state as defined in the stack.

Drift detection ensures consistency and helps align your infrastructure with your code, supporting robust, reliable infrastructure management.





DRIFT DETECTED



Deleting a CloudFormation stack in AWS removes all resources that were created as part of that stack, effectively tearing down the infrastructure. Here’s a breakdown of what happens when a stack is deleted:

Key Outcomes of Deleting a Stack

1. Resource Deletion:

- All resources defined in the stack’s CloudFormation template are deleted. This includes instances like EC2, VPCs, RDS databases, Lambda functions, S3 buckets, and other resources.

- Some resources, such as S3 buckets with manually added data or certain IAM resources, might not delete if configured with deletion protection or if there are custom deletion policies.

2. Dependency Handling:

- CloudFormation deletes resources in an order that respects dependencies, ensuring that resources dependent on others are deleted after those dependencies. For example, an EC2 instance would be deleted before its associated security group if there is a dependency.

3. Deletion Policies:

- Some resources in a stack can be configured with \*\*DeletionPolicy\*\* attributes, which determine how CloudFormation handles them during stack deletion:

- Retain: Keeps the resource instead of deleting it.

- Snapshot: Takes a snapshot of the resource (commonly used for databases like RDS).

- Delete: Deletes the resource (the default behavior).

4. Stack Events and Status:

- CloudFormation records each step of the deletion process in the \*\*Stack Events\*\* tab, which provides details on each resource deletion and any issues encountered.

5. Error Handling and Rollback:

- If a deletion fails, CloudFormation stops the process, marks the stack as \*\*DELETE\_FAILED\*\*, and retains resources that could not be deleted.

- You can then address the issues (e.g., by manually deleting resources that failed or modifying permissions) and retry the deletion.

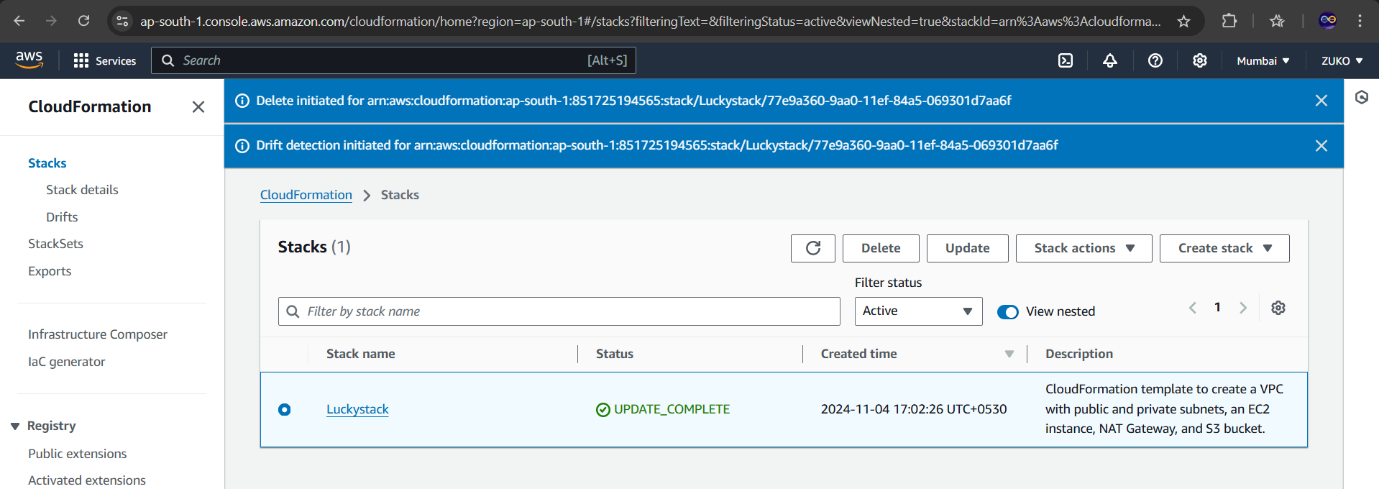
# Use Cases for Stack Deletion

- Cost Management: Deleting a stack removes all associated resources, which helps manage costs by stopping billing for resources you no longer need.

- Environment Clean-Up: After testing, prototyping, or using a temporary environment, deleting a stack helps clean up the resources efficiently.

- Re-Provisioning: You may delete and recreate a stack as part of updating configurations or refreshing resources to a known state.

Deleting a stack is a powerful way to remove an entire environment, and caution is advised to avoid unintended data loss, especially for production resources.



TEMPLATE-YAML

AWSTemplateFormatVersion: "2010-09-09"

Description: CloudFormation template to create a VPC with public and private subnets, an Internet Gateway, two route tables, and an EC2 instance in the public subnet.

Parameters:

VpcCIDR:

Type: String

Default: "10.0.0.0/16"

Description: CIDR block for the VPC

PublicSubnetCIDR:

Type: String

Default: "10.0.1.0/24"

Description: CIDR block for the public subnet

PrivateSubnetCIDR:

Type: String

Default: "10.0.2.0/24"

Description: CIDR block for the private subnet

Resources:

# VPC

VPC:

Type: "AWS::EC2::VPC"

Properties:

CidrBlock: !Ref VpcCIDR

EnableDnsSupport: true

EnableDnsHostnames: true

Tags:

- Key: Name

Value: "vpcproject"

# Internet Gateway

InternetGateway:

Type: "AWS::EC2::InternetGateway"

Properties:

Tags:

- Key: Name

Value: "igw"

AttachGateway:

Type: "AWS::EC2::VPCGatewayAttachment"

Properties:

VpcId: !Ref VPC

InternetGatewayId: !Ref InternetGateway

# Public Subnet

PublicSubnet:

Type: "AWS::EC2::Subnet"

Properties:

VpcId: !Ref VPC

CidrBlock: !Ref PublicSubnetCIDR

MapPublicIpOnLaunch: true

Tags:

- Key: Name

Value: "pubsn"

# Private Subnet

PrivateSubnet:

Type: "AWS::EC2::Subnet"

Properties:

VpcId: !Ref VPC

CidrBlock: !Ref PrivateSubnetCIDR

MapPublicIpOnLaunch: false

Tags:

- Key: Name

Value: "pvtsn"

# Public Route Table

PublicRouteTable:

Type: "AWS::EC2::RouteTable"

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: "pubrt"

PublicRoute:

Type: "AWS::EC2::Route"

Properties:

RouteTableId: !Ref PublicRouteTable

DestinationCidrBlock: "0.0.0.0/0"

GatewayId: !Ref InternetGateway

PublicSubnetRouteTableAssociation:

Type: "AWS::EC2::SubnetRouteTableAssociation"

Properties:

SubnetId: !Ref PublicSubnet

RouteTableId: !Ref PublicRouteTable

# Private Route Table

PrivateRouteTable:

Type: "AWS::EC2::RouteTable"

Properties:

VpcId: !Ref VPC

Tags:

- Key: Name

Value: "pvtrt"

PrivateSubnetRouteTableAssociation:

Type: "AWS::EC2::SubnetRouteTableAssociation"

Properties:

SubnetId: !Ref PrivateSubnet

RouteTableId: !Ref PrivateRouteTable

# NAT Gateway Elastic IP

NatGatewayEIP:

Type: "AWS::EC2::EIP"

DependsOn: AttachGateway

Properties:

Domain: "vpc"

Tags:

- Key: Name

Value: "mynatgwEip"

# NAT Gateway

NatGateway:

Type: "AWS::EC2::NatGateway"

DependsOn: AttachGateway

Properties:

SubnetId: !Ref PublicSubnet

AllocationId: !GetAtt NatGatewayEIP.AllocationId

Tags:

- Key: Name

Value: "mynatgw"

# EC2 Instance in the Public Subnet

PublicEC2Instance:

Type: "AWS::EC2::Instance"

Properties:

InstanceType: "t2.micro"

KeyName: "jumpkey"

ImageId: "ami-04a37924ffe27da53"

SubnetId: !Ref PublicSubnet

Tags:

- Key: Name

Value: "PublicEC2Instance"

SecurityGroupIds:

- !Ref PublicInstanceSecurityGroup

# Security Group for Public EC2 Instance

PublicInstanceSecurityGroup:

Type: "AWS::EC2::SecurityGroup"

Properties:

VpcId: !Ref VPC

GroupDescription: "Allow SSH and HTTP access"

SecurityGroupIngress:

- IpProtocol: "tcp"

FromPort: 22

ToPort: 22

CidrIp: "0.0.0.0/0" # SSH access from anywhere

- IpProtocol: "tcp"

FromPort: 80

ToPort: 80

CidrIp: "0.0.0.0/0" # HTTP access from anywhere

Tags:

- Key: Name

Value: "PublicInstanceSecurityGroup"

Outputs:

VPCId:

Description: "VPC ID"

Value: !Ref VPC

PublicSubnetId:

Description: "Public Subnet ID"

Value: !Ref PublicSubnet

PrivateSubnetId:

Description: "Private Subnet ID"

Value: !Ref PrivateSubnet

InternetGatewayId:

Description: "Internet Gateway ID"

Value: !Ref InternetGateway

PublicEC2InstanceId:

Description: "EC2 Instance ID in the Public Subnet"

Value: !Ref PublicEC2Instance