Launching an Amazon EC2 Instance Using AWS CloudFormation - Capstone Project

AWS resources can be launched using two methods

1. AWS CloudFormation template (console)
2. AWS CloudFormation template (AWS CLI)

Both involves same sequence of steps to be followed with minor modifications wherever needed as follows:

### 1. Write a CloudFormation Template

Create a CloudFormation template in YAML or JSON format. This template will define the resources needed

### 2. Deploy the Stack

Once the CloudFormation template is ready, we can deploy it using the AWS Management Console, AWS CLI, or AWS SDKs

### 3. Monitor the Stack Creation

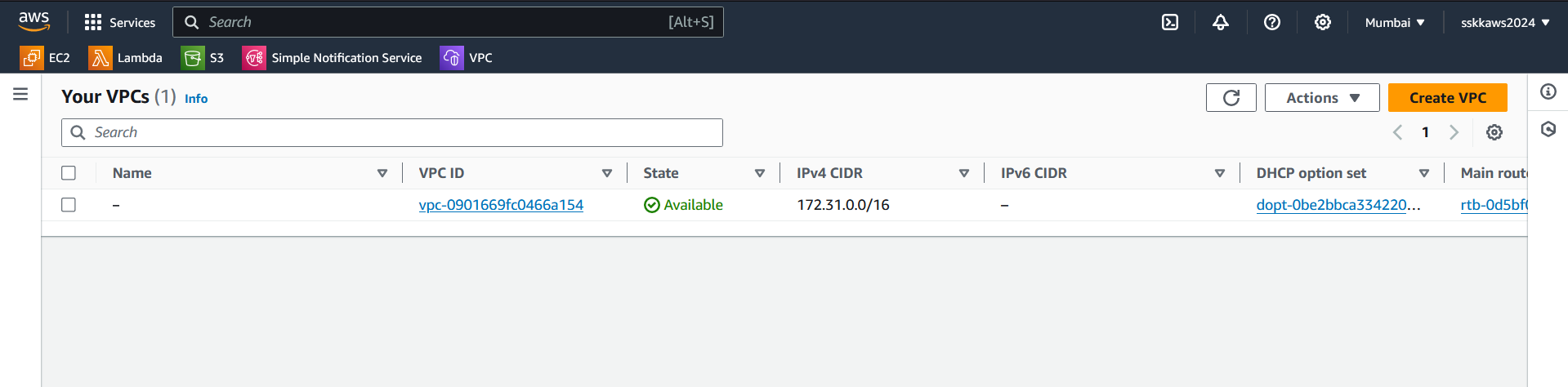
After executing the command, monitor the stack creation process in the AWS Management Console or using the AWS CLI until it reaches the CREATE\_COMPLETE state.

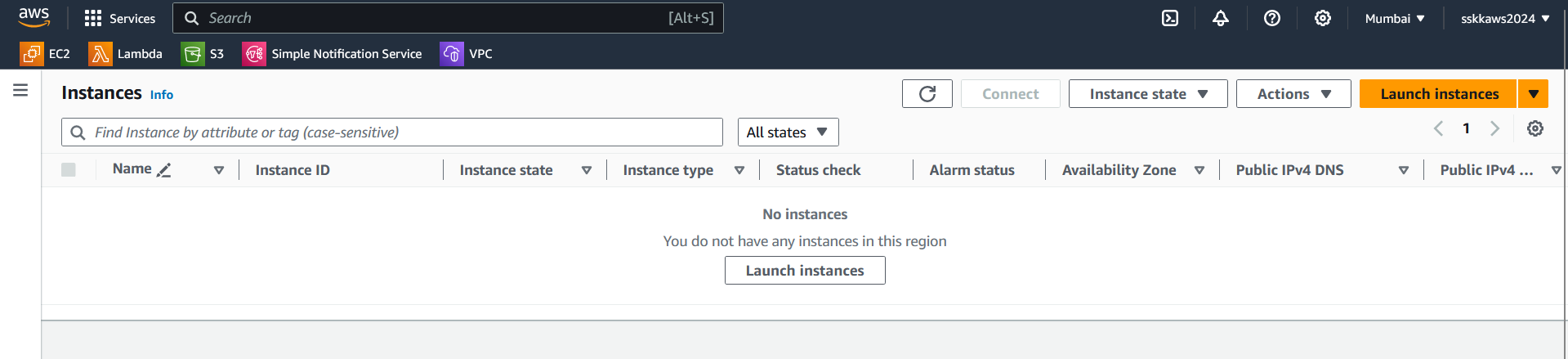
### 4. Access your resources

Once the stack creation is complete, we can access your EC2 instance either through SSH.

### 5. Cleanup

After completing the project, don't forget to delete the CloudFormation stack to avoid incurring unnecessary charges





**Creation of a VPC (Virtual Private Cloud) using cloudformation.**

We shall create a VPC with subnets, Route tables, Internet gateway and NAT gateway. For this demo we will make use of Cloudformation using CLI

1. **CloudFormation Template (**vpc.yaml **):**

|  |
| --- |
| Resources:  MyVPC:  Type: AWS::EC2::VPC  Properties:  CidrBlock: "111.11.0.0/20"  EnableDnsSupport: true  EnableDnsHostnames: true  Tags:  - Key: Name  Value: cloudformation\_VPC  PublicSubnet1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.1.0/24"  AvailabilityZone: !Select [0, !GetAZs '']  Tags:  - Key: Name  Value: public-subnet-1  PublicSubnet2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.2.0/24"  AvailabilityZone: !Select [1, !GetAZs '']  Tags:  - Key: Name  Value: public-subnet-2  PublicSubnet3:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.3.0/24"  AvailabilityZone: !Select [2, !GetAZs '']  Tags:  - Key: Name  Value: public-subnet-3  PrivateSubnet1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.4.0/24"  AvailabilityZone: !Select [0, !GetAZs '']  Tags:  - Key: Name  Value: private-subnet-1  PrivateSubnet2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.5.0/24"  AvailabilityZone: !Select [1, !GetAZs '']  Tags:  - Key: Name  Value: private-subnet-2  PrivateSubnet3:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref MyVPC  CidrBlock: "111.11.6.0/24"  AvailabilityZone: !Select [2, !GetAZs '']  Tags:  - Key: Name  Value: private-subnet-3  InternetGateway:  Type: AWS::EC2::InternetGateway  Properties:  Tags:  - Key: Name  Value: internet-gateway  VPCGatewayAttachment:  Type: AWS::EC2::VPCGatewayAttachment  Properties:  VpcId: !Ref MyVPC  InternetGatewayId: !Ref InternetGateway  PublicRouteTable:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref MyVPC  Tags:  - Key: Name  Value: public-route-table  PrivateRouteTable:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref MyVPC  Tags:  - Key: Name  Value: private-route-table  PublicRoute:  Type: AWS::EC2::Route  DependsOn: VPCGatewayAttachment  Properties:  RouteTableId: !Ref PublicRouteTable  DestinationCidrBlock: "0.0.0.0/0"  GatewayId: !Ref InternetGateway  PublicSubnet1RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PublicSubnet1  RouteTableId: !Ref PublicRouteTable  PublicSubnet2RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PublicSubnet2  RouteTableId: !Ref PublicRouteTable  PublicSubnet3RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PublicSubnet3  RouteTableId: !Ref PublicRouteTable  PrivateSubnet1RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PrivateSubnet1  RouteTableId: !Ref PrivateRouteTable  PrivateSubnet2RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PrivateSubnet2  RouteTableId: !Ref PrivateRouteTable  PrivateSubnet3RouteAssociation:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PrivateSubnet3  RouteTableId: !Ref PrivateRouteTable |

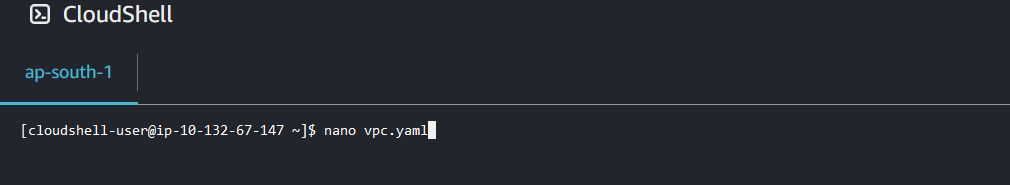
This template creates the following resources:

* VPC named "cloudformation\_VPC" with CIDR block 111.11.0.0/20.
* Three public subnets with CIDR blocks 111.11.1.0/24, 111.11.2.0/24, and 111.11.3.0/24, each in a different availability zone.
* Three private subnets with CIDR blocks 111.11.4.0/24, 111.11.5.0/24, and 111.11.6.0/24, each in a different availability zone.
* An internet gateway attached to the VPC.
* Two route tables, one for public subnets and one for private subnets.
* A route in the public route table to route internet-bound traffic (0.0.0.0/0) through the internet gateway.
* An Elastic IP (EIP) for the NAT gateway.
* A NAT gateway in one of the public subnets.
* A route in the private route table to route internet-bound traffic (0.0.0.0/0) through the NAT gateway.

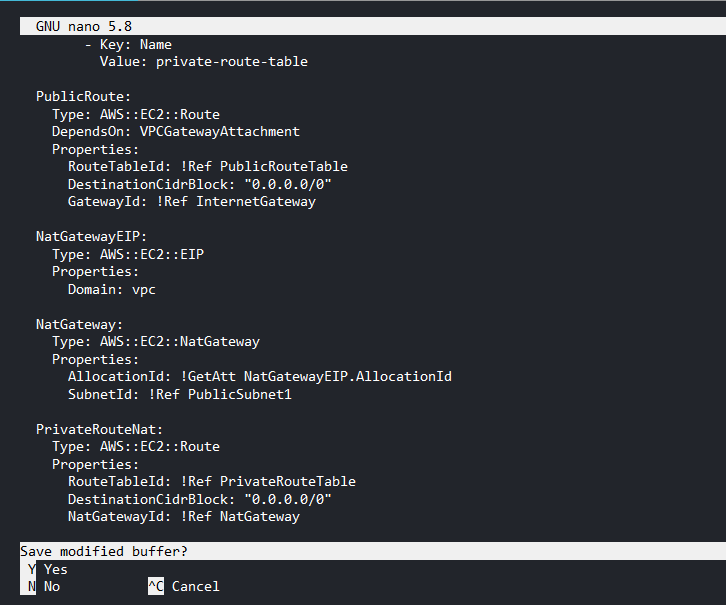
1. **Documentation of the Provisioning Process:**

**Provisioning Process:**

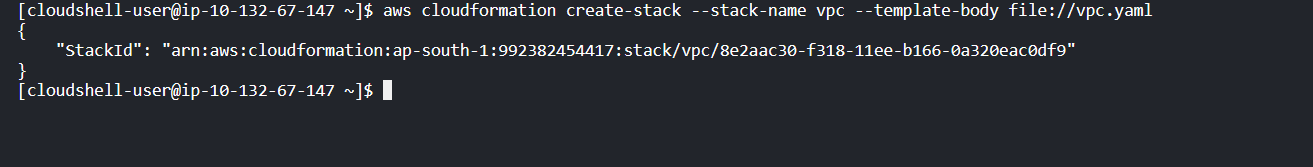
1. Access AWS CloudShell from the AWS Management Console.
2. Use your preferred text editor to create a CloudFormation template



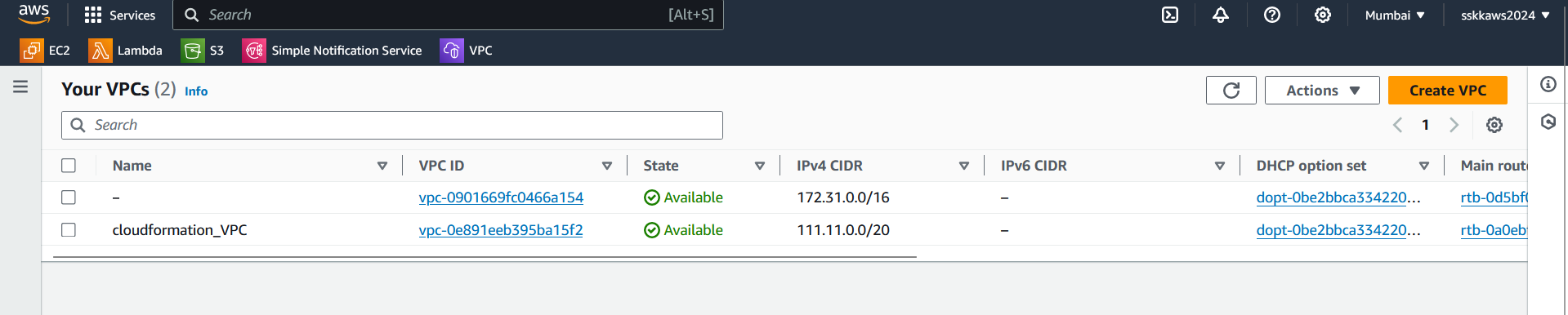
1. Save the CloudFormation template

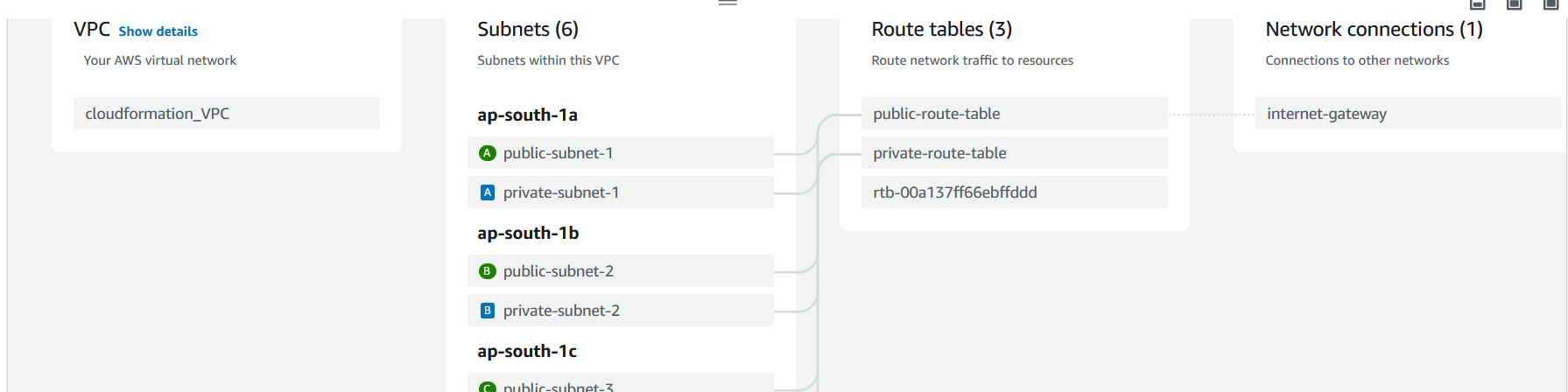


1. Run CloudFormation Stack



1. Once the stack creation is complete, verify that the resource instance is provisioned as expected





**Creation of EC2 instances using cloudformation.**

1. **CloudFormation Template (**vpc.yaml **):**

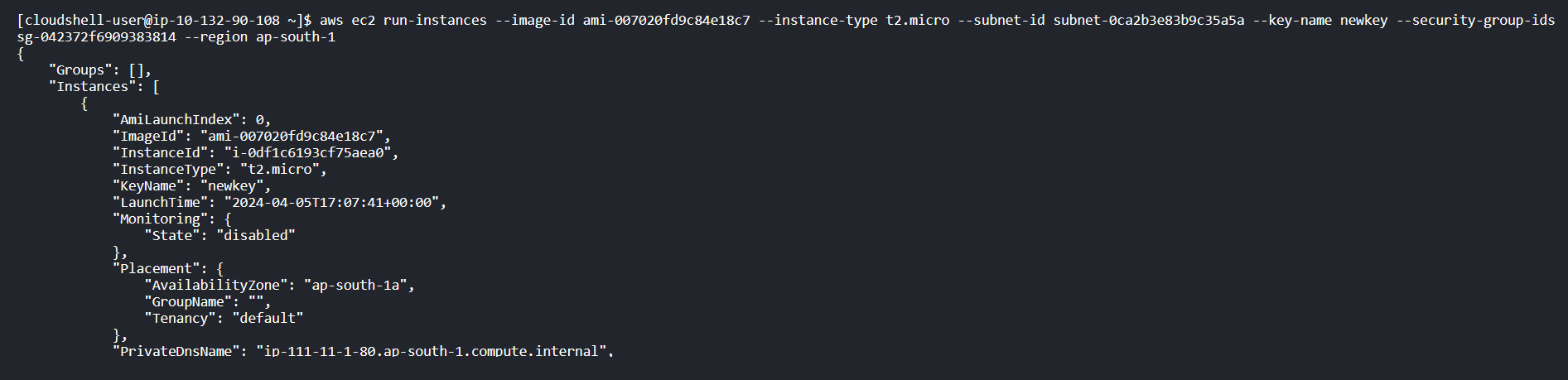
We shall create an instance within the subnets with help of Cloudformation .

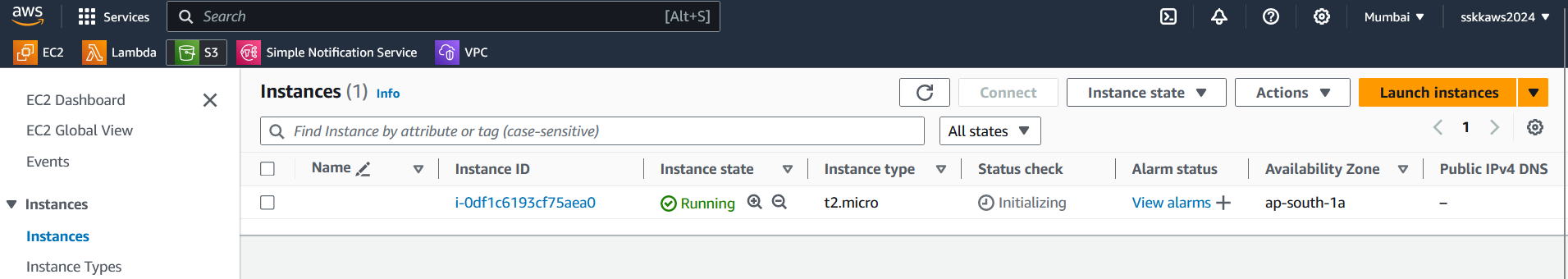
|  |
| --- |
| AWSTemplateFormatVersion: '2010-09-09'  Description: CloudFormation Template for EC2 Instances in Public and Private Subnets  Parameters:  PublicInstanceType:  Description: Instance type for the EC2 instance in the public subnet  Type: String  Default: t2.micro  AllowedValues:  - t2.micro  - t2.small  - t2.medium  ConstraintDescription: Must be a valid EC2 instance type.  PrivateInstanceType:  Description: Instance type for the EC2 instance in the private subnet  Type: String  Default: t2.micro  AllowedValues:  - t2.micro  - t2.small  - t2.medium  ConstraintDescription: Must be a valid EC2 instance type.  ExistingVPCId: # Enter VPC ID here  Description: ID of the existing VPC where the instances will be launched  Type: AWS::EC2::VPC::Id  ExistingPublicSubnetId: # Enter public subnet ID here  Description: ID of the existing public subnet where the instance will be launched  Type: AWS::EC2::Subnet::Id  ExistingPrivateSubnetId: # Enter private subnet ID here  Description: ID of the existing private subnet where the instance will be launched  Type: AWS::EC2::Subnet::Id  Resources:  MyInstanceRole:  Type: AWS::IAM::Role  Properties:  AssumeRolePolicyDocument:  Version: '2012-10-17'  Statement:  - Effect: Allow  Principal:  Service: ec2.amazonaws.com  Action: sts:AssumeRole  PublicSecurityGroup:  Type: 'AWS::EC2::SecurityGroup'  Properties:  GroupDescription: Enable SSH access for public instance  VpcId: !Ref ExistingVPCId  SecurityGroupIngress:  - IpProtocol: tcp  FromPort: '22'  ToPort: '22'  CidrIp: 0.0.0.0/0  PrivateSecurityGroup:  Type: 'AWS::EC2::SecurityGroup'  Properties:  GroupDescription: Allow all traffic from within the VPC  VpcId: !Ref ExistingVPCId  SecurityGroupIngress:  - IpProtocol: -1  FromPort: '0'  ToPort: '65535'  CidrIp: 10.0.0.0/16  PublicEC2Instance:  Type: 'AWS::EC2::Instance'  Properties:  InstanceType: !Ref PublicInstanceType  SecurityGroupIds:  - !GetAtt PublicSecurityGroup.GroupId  SubnetId: !Ref ExistingPublicSubnetId  ImageId: 'ami-xxxxxxxxxxxxxx' # Insert your desired AMI ID here  PrivateEC2Instance:  Type: 'AWS::EC2::Instance'  Properties:  InstanceType: !Ref PrivateInstanceType  SecurityGroupIds:  - !GetAtt PrivateSecurityGroup.GroupId  SubnetId: !Ref ExistingPrivateSubnetId  ImageId: 'ami-xxxxxxxxxxxxxx' # Insert your desired AMI ID here |

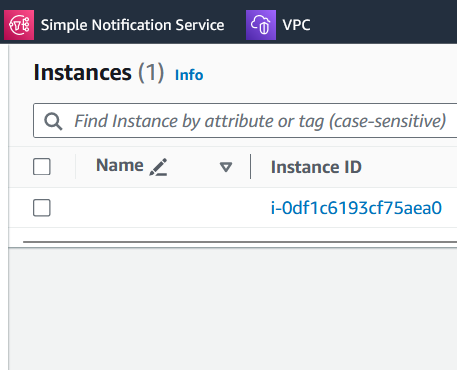
We need to add the VPC id, subnet id and image id to launch a stack and create an EC2 instance following the same procedure as before. Here additionally, we have configured IAM roles to be utilized by the resources.

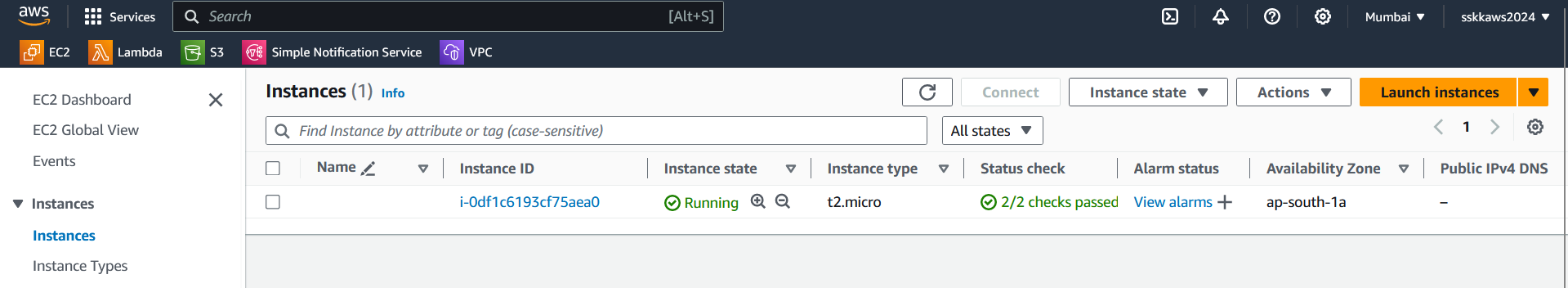
1. **Alternate EC2 creation method using CLI**
   1. Alternatively we can launch an instance directly from CLI, since we have already created the PC, subnets and gateways.

|  |
| --- |
| aws ec2 run-instances \  --image-id ami-1234567890abcdef0 \  --instance-type t2.micro \  --subnet-id subnet-12345678 \  --key-name my-keypair \  --security-group-ids sg-0123456789abcdef0 \  --region your-region  --iam-instance-profile Name=MyInstanceRole |



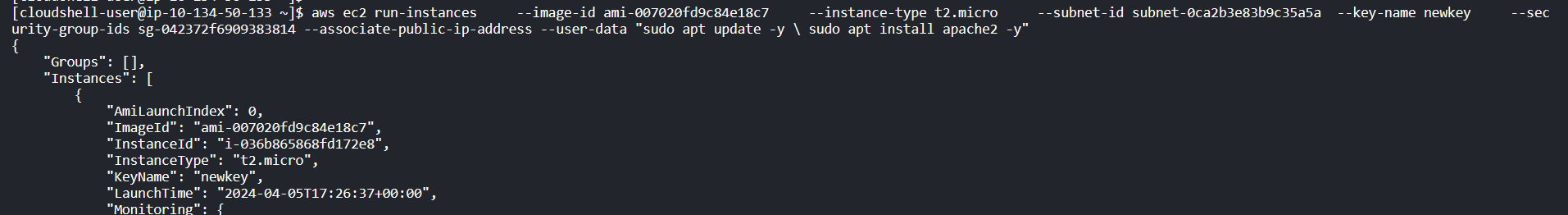


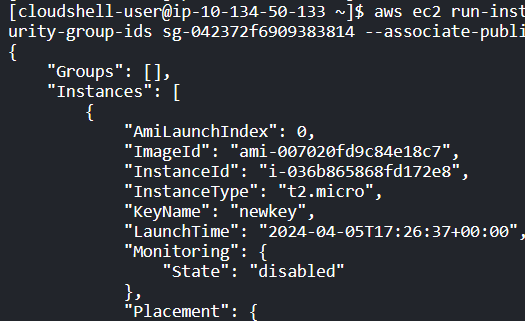
 

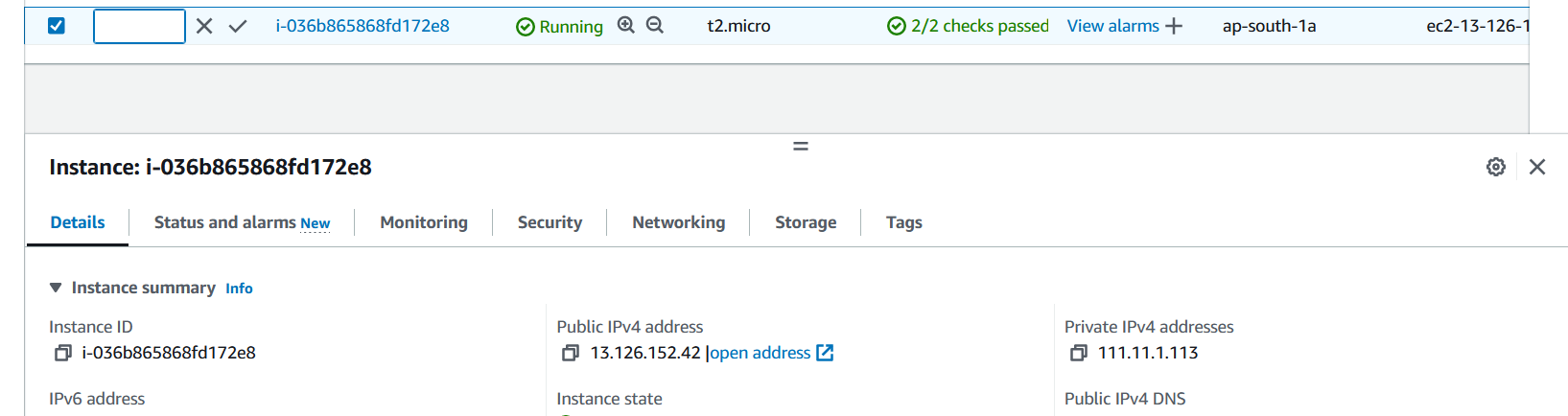


* 1. Add the snippet –associate-public-ip-address to enable default allocation of public ip.

|  |
| --- |
| aws ec2 run-instances \  --image-id ami-1234567890abcdef0 \  --instance-type t2.micro \  --subnet-id subnet-12345678 \  --key-name my-keypair \  --security-group-ids sg-0123456789abcdef0 \  --region your-region  *--associate-public-ip-address* |







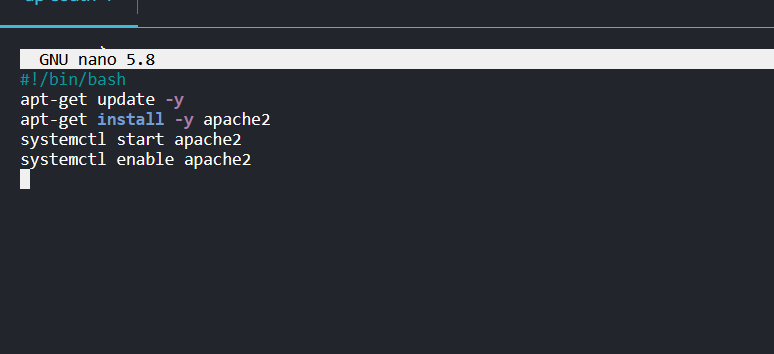
A public ip is added to enable ssh.

* 1. EC2 instance creation with public ip and apache2 provisioned without manual intervention

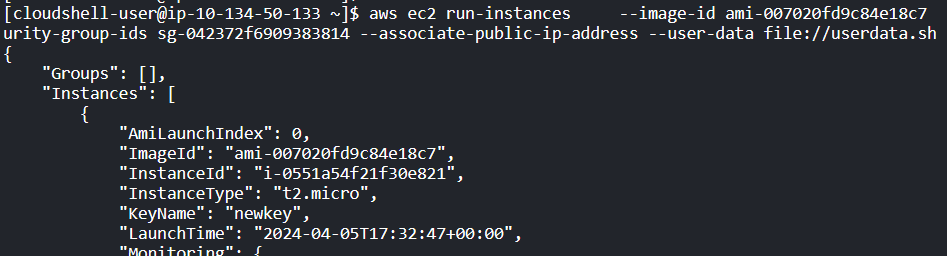
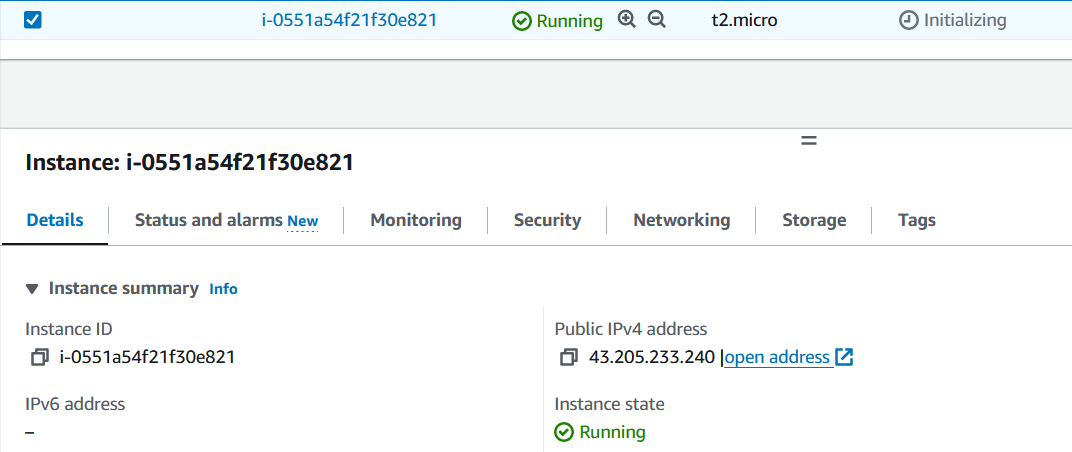
In order to launch a web server, we need to provision the instance with web engine. It can be done by connecting the instance through public ip and manually provisioning. Or we can also provision the instance with a web engine, while launching the instance from the CLI. An example is given below. The userdata.sh is a script which is stored in the cloudshell directory, it can be used to provision the instance with apache2 engine whenever needed.

The user data to install apache2 is stored as a script userdata.sh





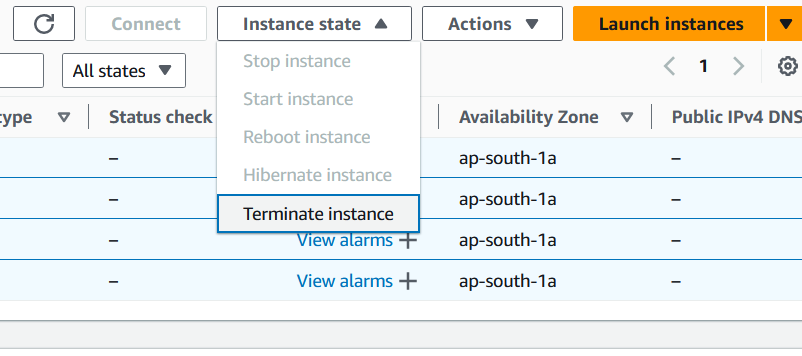
The userdata script is referred at the end of creation of EC2 instance to automatically provision the installation of apache2. Since the script is stored in cloudshell directory, the script can be invoked any number of times as wanted.

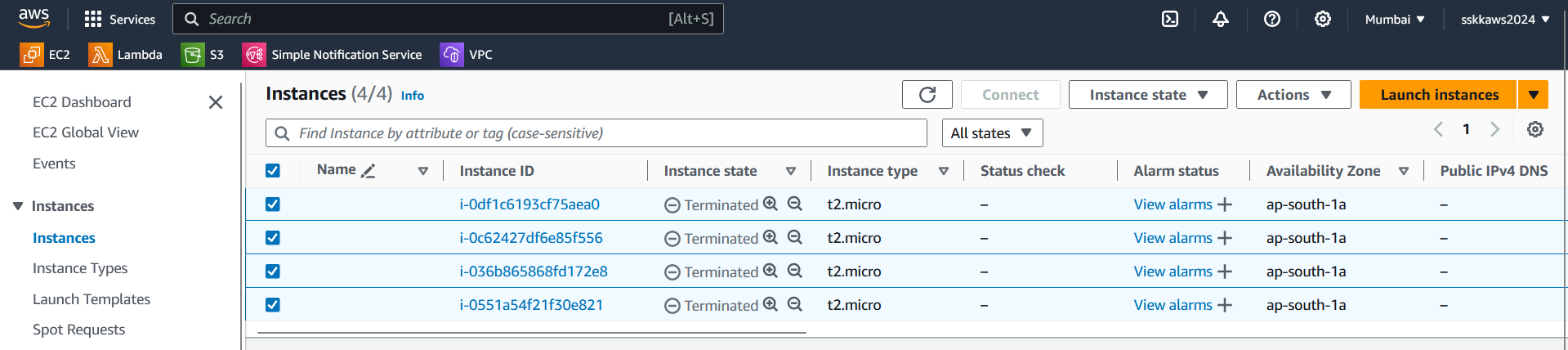
 



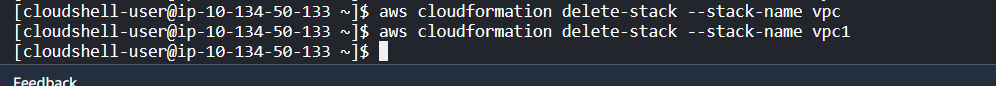
**Cleaning up of resources**

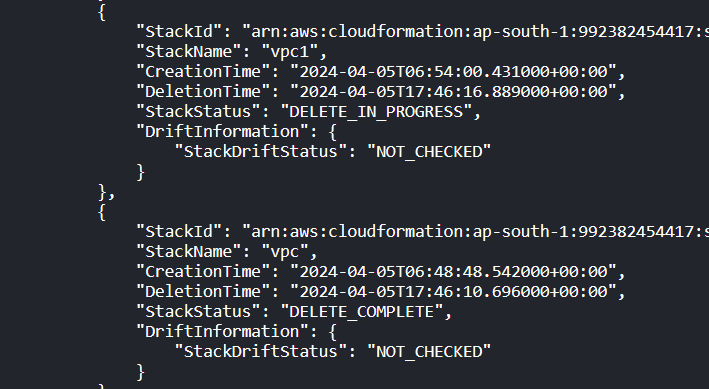
The ec2 instance is a minor resource, so it can be launched using the simple procedure given above. Similarly cleaning up the instances is easier, as we can terminaate the instances directly from the console.

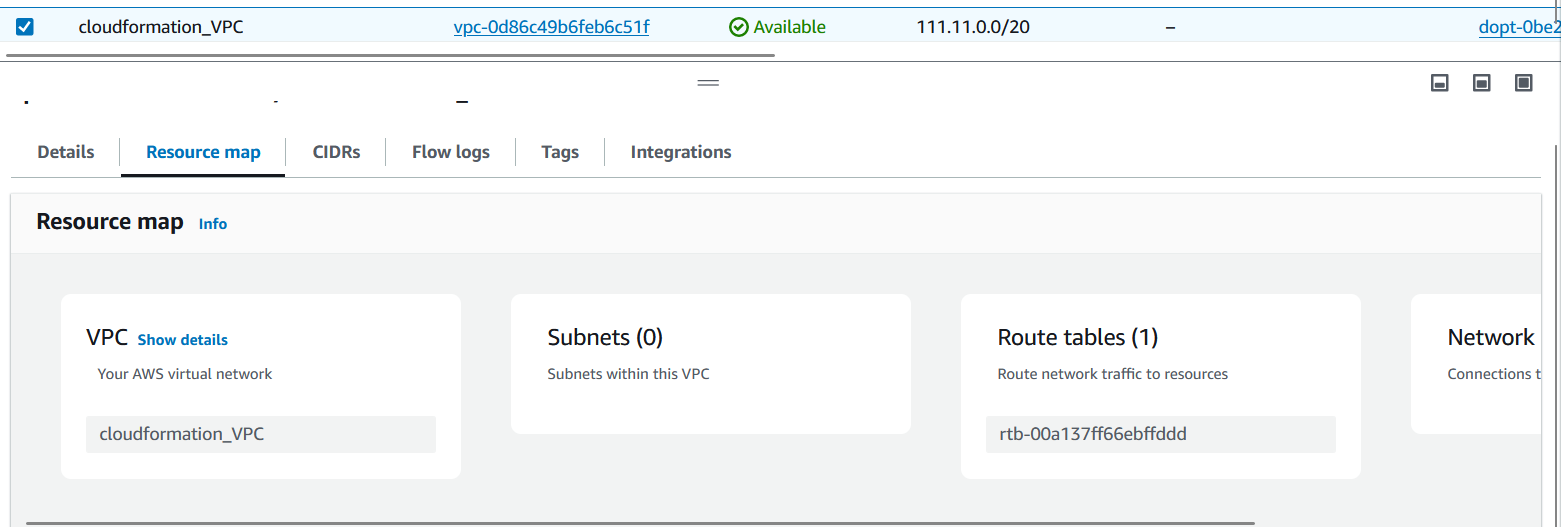




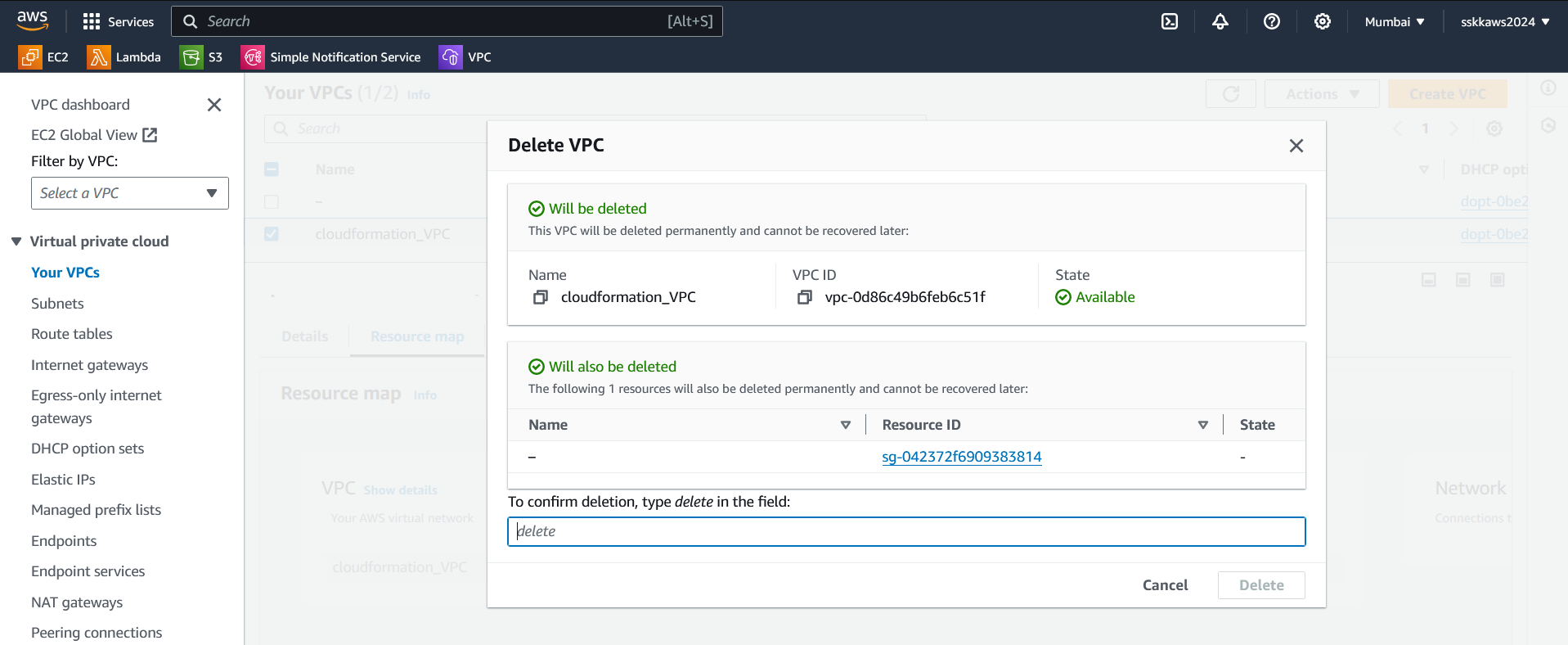
To clean up CloudFormation stacks, you can use the delete-stack command in the AWS CLI.



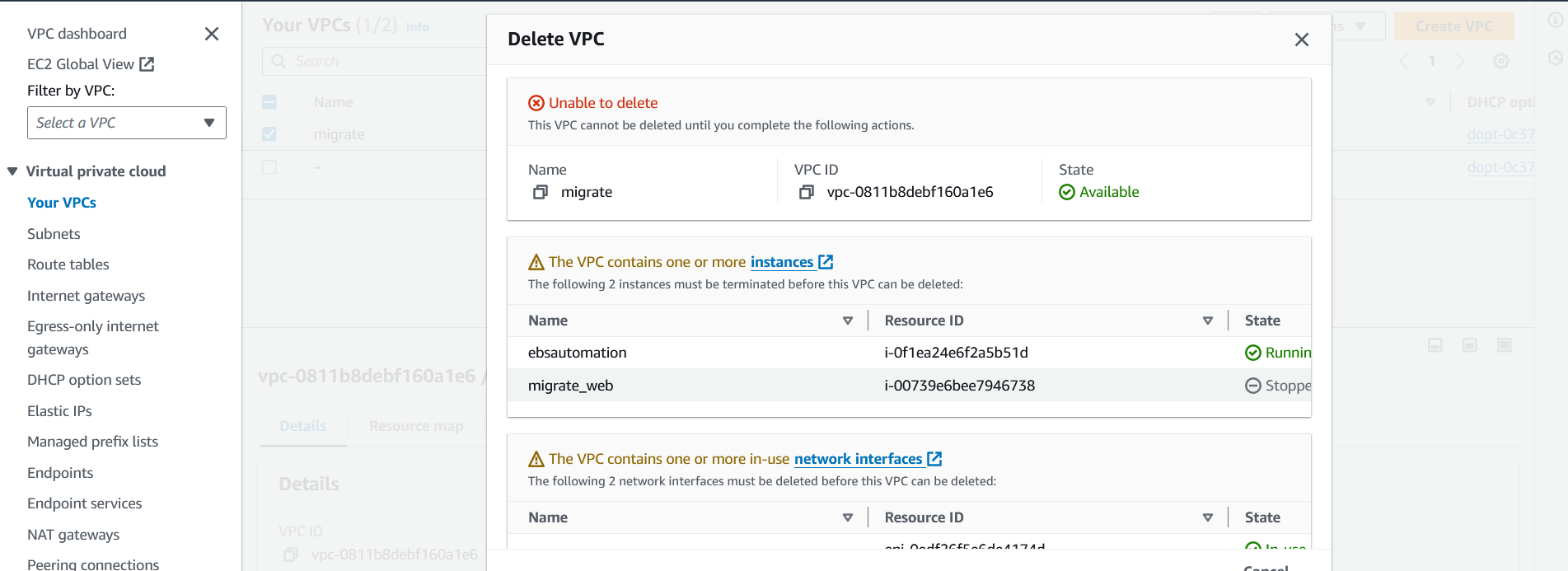




Since ec2 is a minor resource, no major cleanup is required. But the VPC should be cleaned up in a complex way, by deprovisioning the individual resources. The route tables are to disassociated and the gateways to be removed from the incoming and outgoing ports. So the user have to be careful while cleaning up. But since all the resources are provisioned by the stack, delete-stack command removes all individual components. It is easier to provision and deprovision if the user is well aware of the architecture.



We can take other VPCs as reference. Refer the VPC below, deleting the VPC gives warning about the resources operating within the VPC.



**Testing and Verification**

**Testing Steps:**

1. Created a CloudFormation stack using the yaml template.
2. Verified the stack creation status in the CloudFormation console.
3. Checked the provisioned EC2 instance in the EC2 service to ensure it's running.
4. Accessed the EC2 instance via SSH to verify Apache installation and service status.

**Verification Steps:**

* The CloudFormation stack creation completed successfully without any errors.
* The EC2 instance was provisioned with the specified configuration.
* Apache web server was successfully installed and running on the EC2 instance.
* Verified that the EC2 instance is accessible via SSH and HTTP.

**CloudFormation template defining a CloudWatch alarm for monitoring CPU utilization of an EC2 instance**

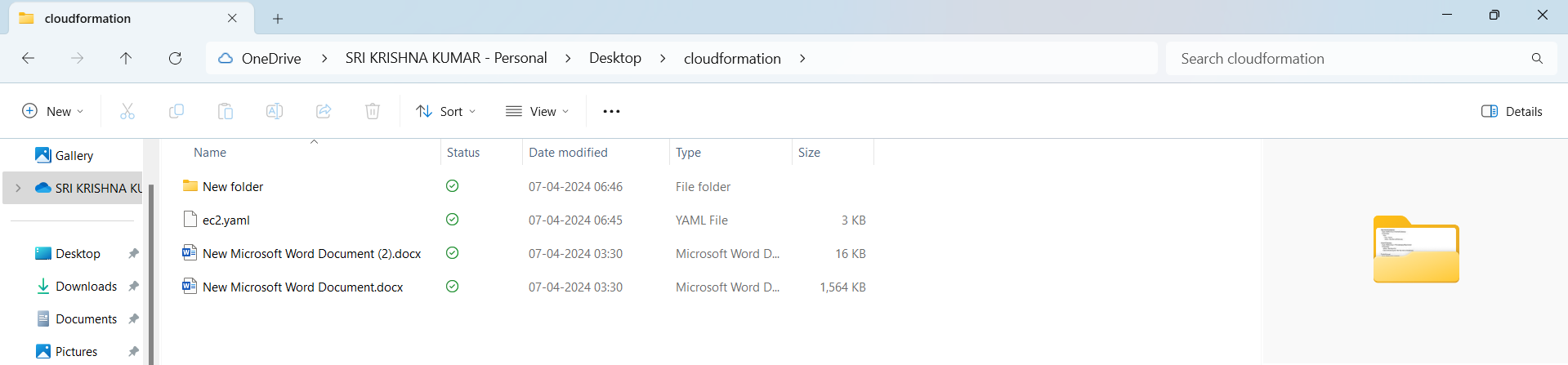
To create an EC2 instance with CloudWatch monitoring enabled using CloudFormation, we can modify the template to include an EC2 instance resource with CloudWatch monitoring enabled.

|  |
| --- |
| AWSTemplateFormatVersion: '2010-09-09'  Description: Creates an EC2 instance with CloudWatch monitoring enabled  Resources:  MyInstance:  Type: AWS::EC2::Instance  Properties:  InstanceType: t2.micro  ImageId: ami-0c55b159cbfafe1f0 # You can replace this with your desired AMI  Monitoring: true # Enable CloudWatch monitoring  Tags:  - Key: Name  Value: MyEC2Instance  Outputs:  InstanceId:  Description: ID of the EC2 instance  Value: !Ref MyInstance  Export:  Name: MyEC2InstanceId |

Now, we shall launch an entire environment for hosting a server. We will configure a VPC, with subnets-security group-internet gateway, an ec2 instance with apache2 installed and configure a cloudwatch alarm to monitor the cpu utilization in a single template. We shall enable the creation of resources using aws console.

**Save the Template**

|  |
| --- |
| **AWSTemplateFormatVersion: '2010-09-09'**  **Description: Creates a VPC with subnets, security group, internet gateway, an EC2 instance with Apache2 installed, and a CloudWatch alarm for CPU utilization**  **Resources:**  **MyVPC:**  **Type: AWS::EC2::VPC**  **Properties:**  **CidrBlock: '123.12.0.0/20'**  **Tags:**  **- Key: Name**  **Value: MyVPC**  **MyInternetGateway:**  **Type: AWS::EC2::InternetGateway**  **Properties:**  **Tags:**  **- Key: Name**  **Value: MyInternetGateway**  **AttachGateway:**  **Type: AWS::EC2::VPCGatewayAttachment**  **Properties:**  **VpcId: !Ref MyVPC**  **InternetGatewayId: !Ref MyInternetGateway**  **PublicSubnet:**  **Type: AWS::EC2::Subnet**  **Properties:**  **VpcId: !Ref MyVPC**  **CidrBlock: '123.12.0.0/24'**  **AvailabilityZone: !Select [0, !GetAZs '']**  **Tags:**  **- Key: Name**  **Value: PublicSubnet**  **MySecurityGroup:**  **Type: AWS::EC2::SecurityGroup**  **Properties:**  **GroupDescription: Enable SSH and HTTP traffic**  **VpcId: !Ref MyVPC**  **SecurityGroupIngress:**  **- IpProtocol: tcp**  **FromPort: '22'**  **ToPort: '22'**  **CidrIp: '0.0.0.0/0'**  **- IpProtocol: tcp**  **FromPort: '80'**  **ToPort: '80'**  **CidrIp: '0.0.0.0/0'**  **Tags:**  **- Key: Name**  **Value: MySecurityGroup**  **MyEC2Instance:**  **Type: AWS::EC2::Instance**  **Properties:**  **InstanceType: t2.micro**  **ImageId: ami-007020fd9c84e18c7**  **SubnetId: !Ref PublicSubnet**  **SecurityGroupIds:**  **- !Ref MySecurityGroup**  **AssociatePublicIpAddress: true**  **UserData:**  **Fn::Base64: |**  **#!/bin/bash**  **sudo apt update -y**  **sudo apt install -y apache2**  **sudo systemctl start apache2**  **sudo systemctl enable apache2**  **Tags:**  **- Key: Name**  **Value: MyEC2Instance**  **CPUUtilizationAlarm:**  **Type: AWS::CloudWatch::Alarm**  **Properties:**  **AlarmDescription: Alarm for CPU utilization exceeding 50%**  **Namespace: AWS/EC2**  **MetricName: CPUUtilization**  **Dimensions:**  **- Name: InstanceId**  **Value: !Ref MyEC2Instance**  **Statistic: Average**  **Period: 300**  **EvaluationPeriods: 1**  **Threshold: 70**  **ComparisonOperator: GreaterThanThreshold**  **AlarmActions:**  **- !Sub "arn:aws:sns:${AWS::Region}:${AWS::AccountId}:MyEmailNotificationTopic"**  **MyEmailNotificationTopic:**  **Type: AWS::SNS::Topic**  **Properties:**  **DisplayName: MyEmailNotificationTopic**  **MyEmailSubscription:**  **Type: AWS::SNS::Subscription**  **Properties:**  **Endpoint: myemail@example.com**  **Protocol: email**  **TopicArn: !Ref MyEmailNotificationTopic**  **Outputs:**  **InstanceId:**  **Description: ID of the EC2 instance**  **Value: !Ref MyEC2Instance**  **Export:**  **Name: MyEC2InstanceId** |

****

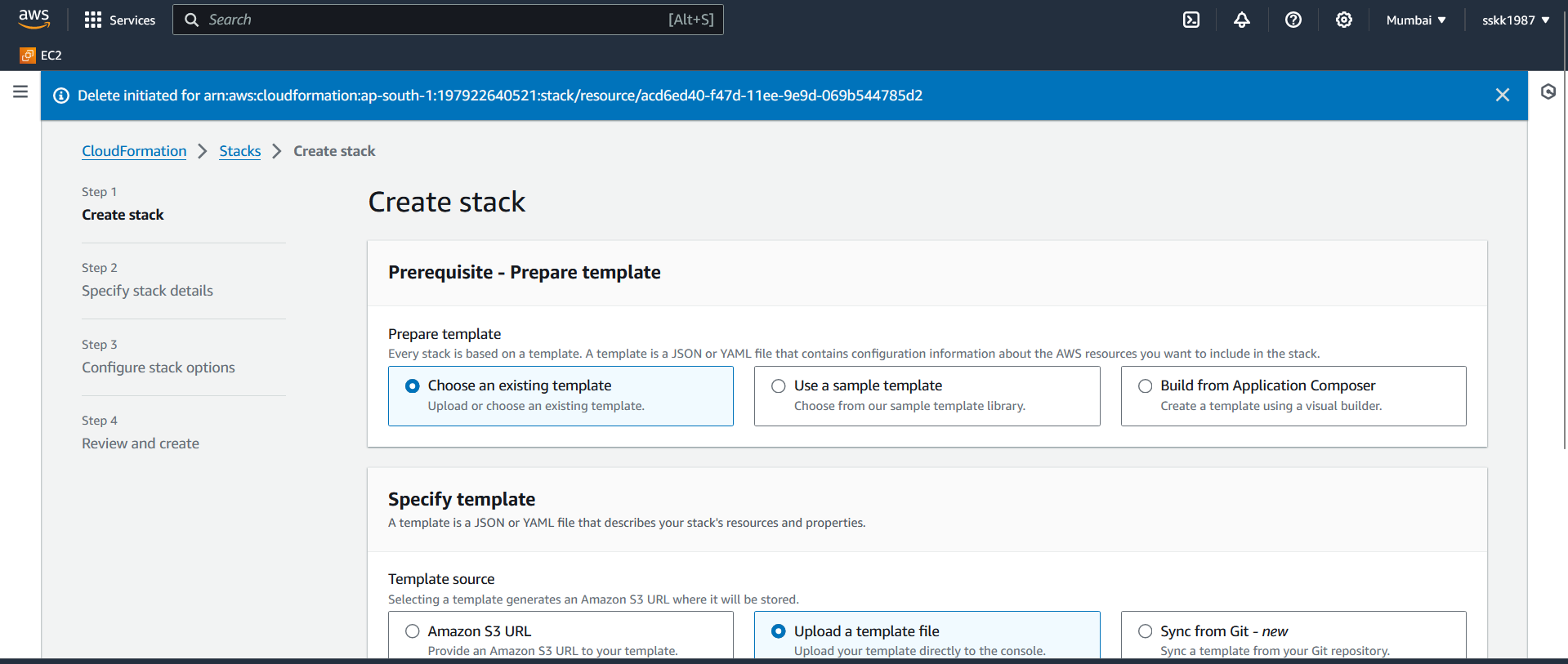
**The template for VPC, ec2 and cloudwatch is stored as an yaml file in local disk.**

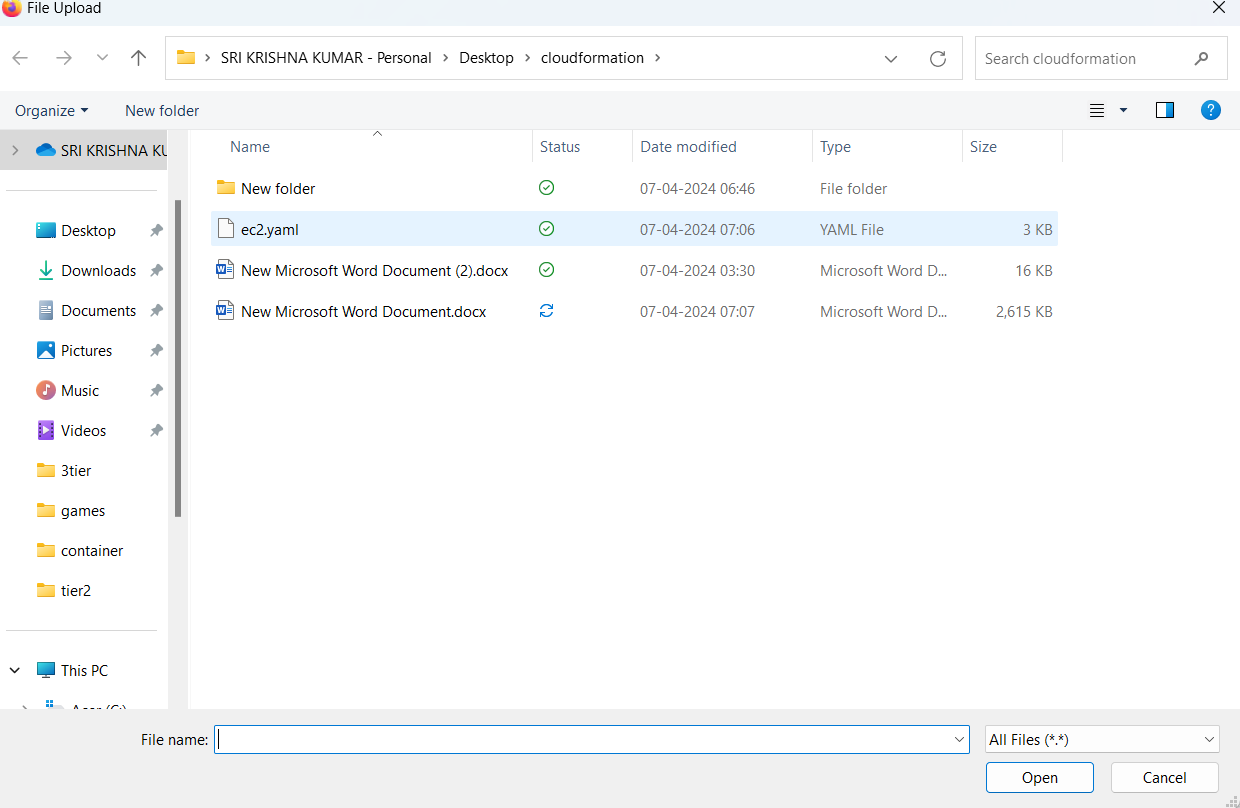
**Deploy the Template**

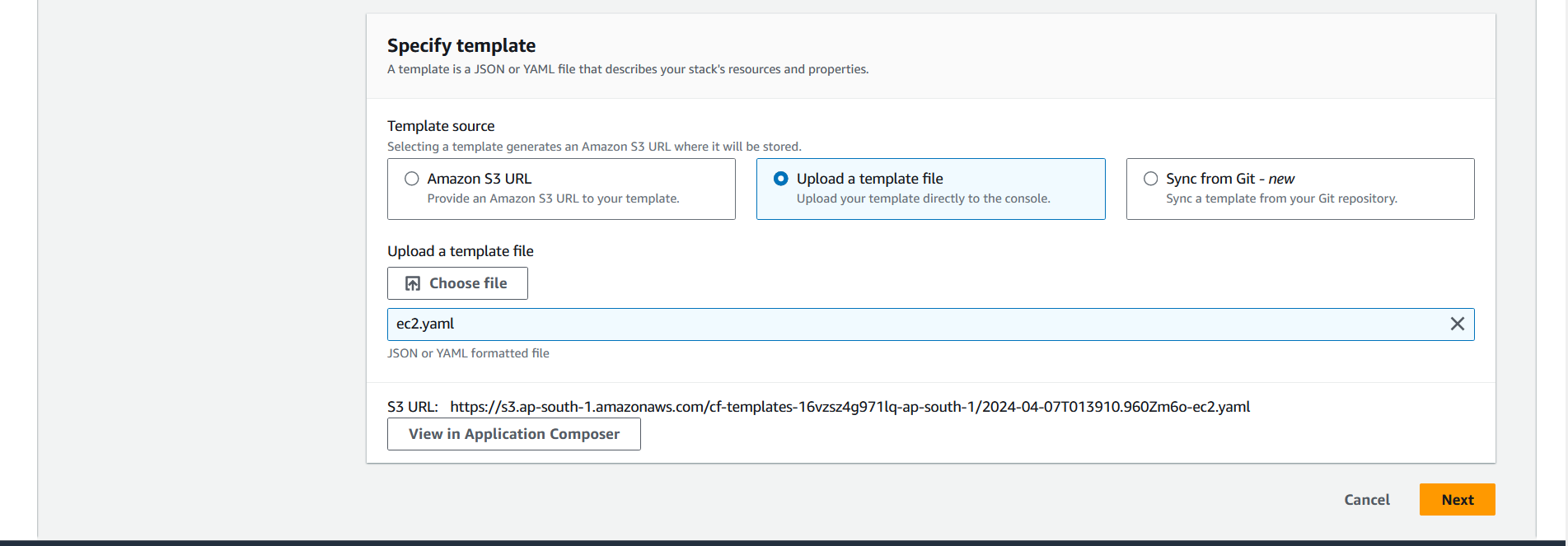
 Navigate to the CloudFormation service.

 Click on "Create stack" and select "With new resources (standard)".

 Choose the option to upload a template file, and select the YAML file you saved.

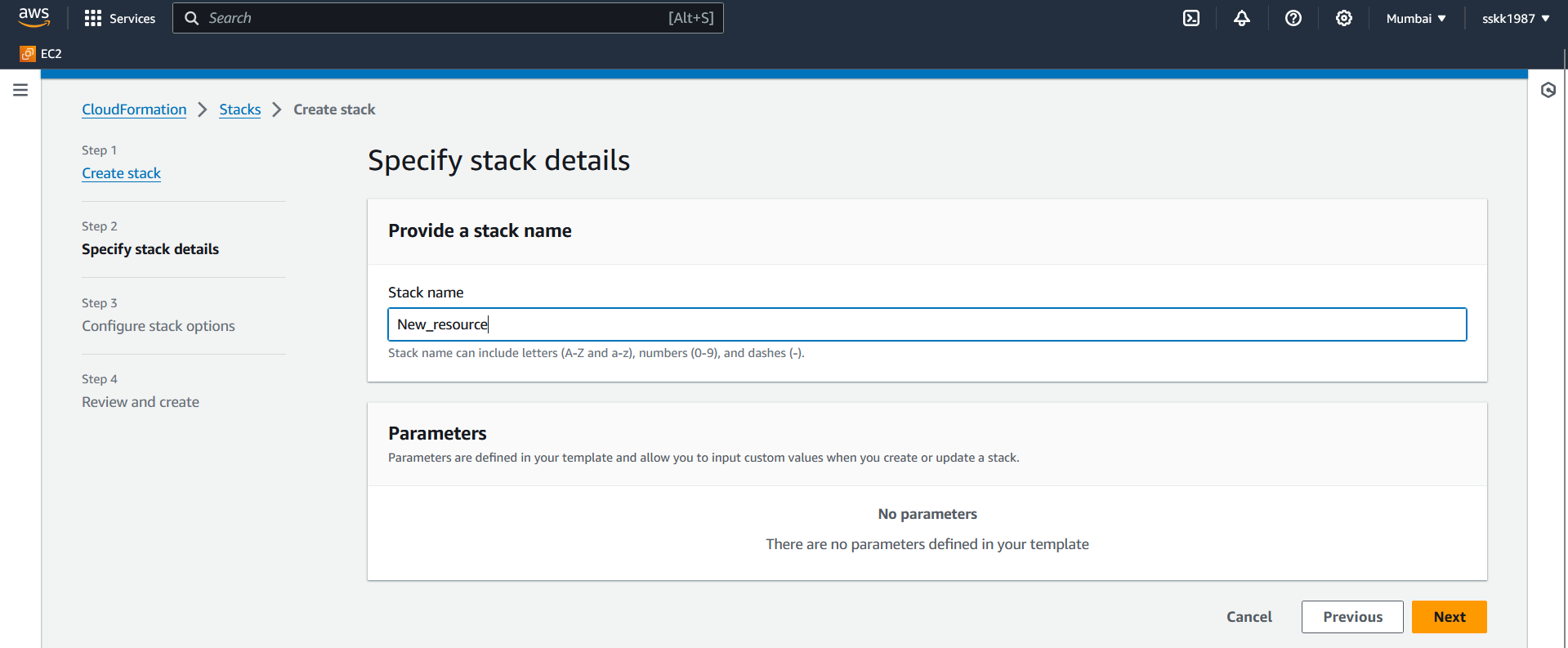






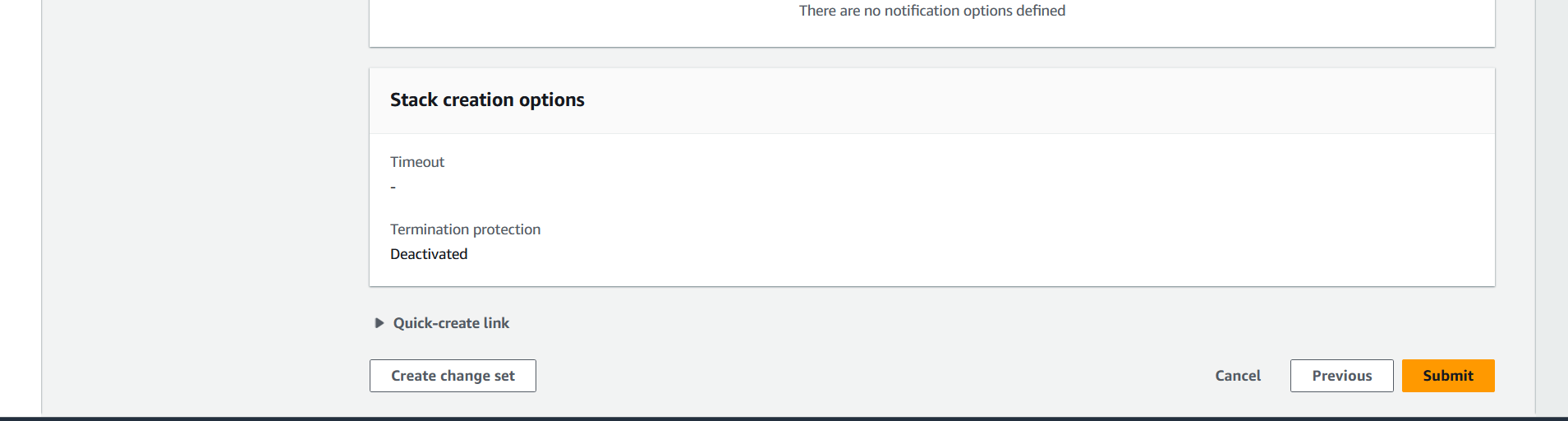
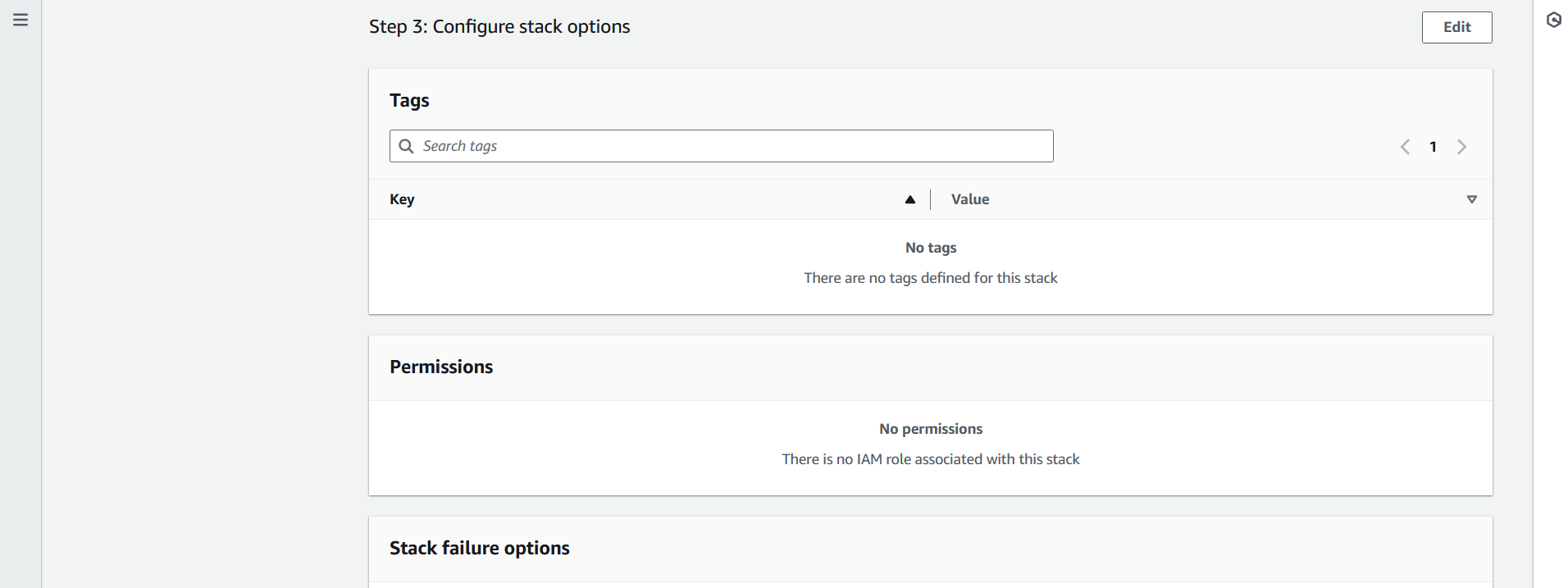
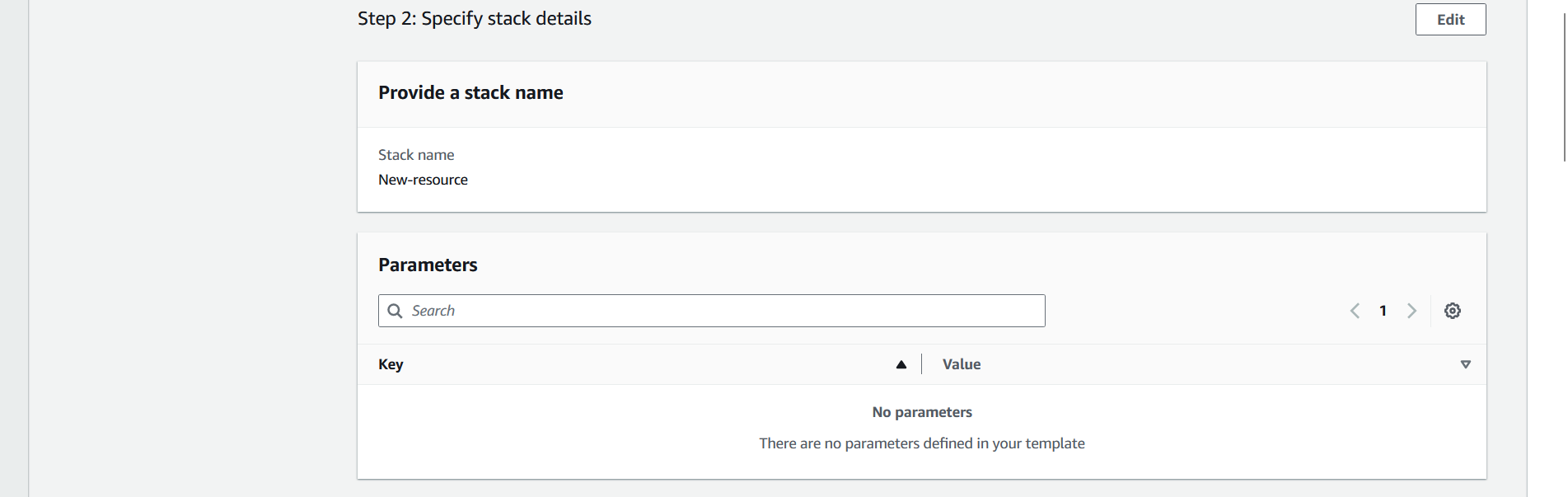
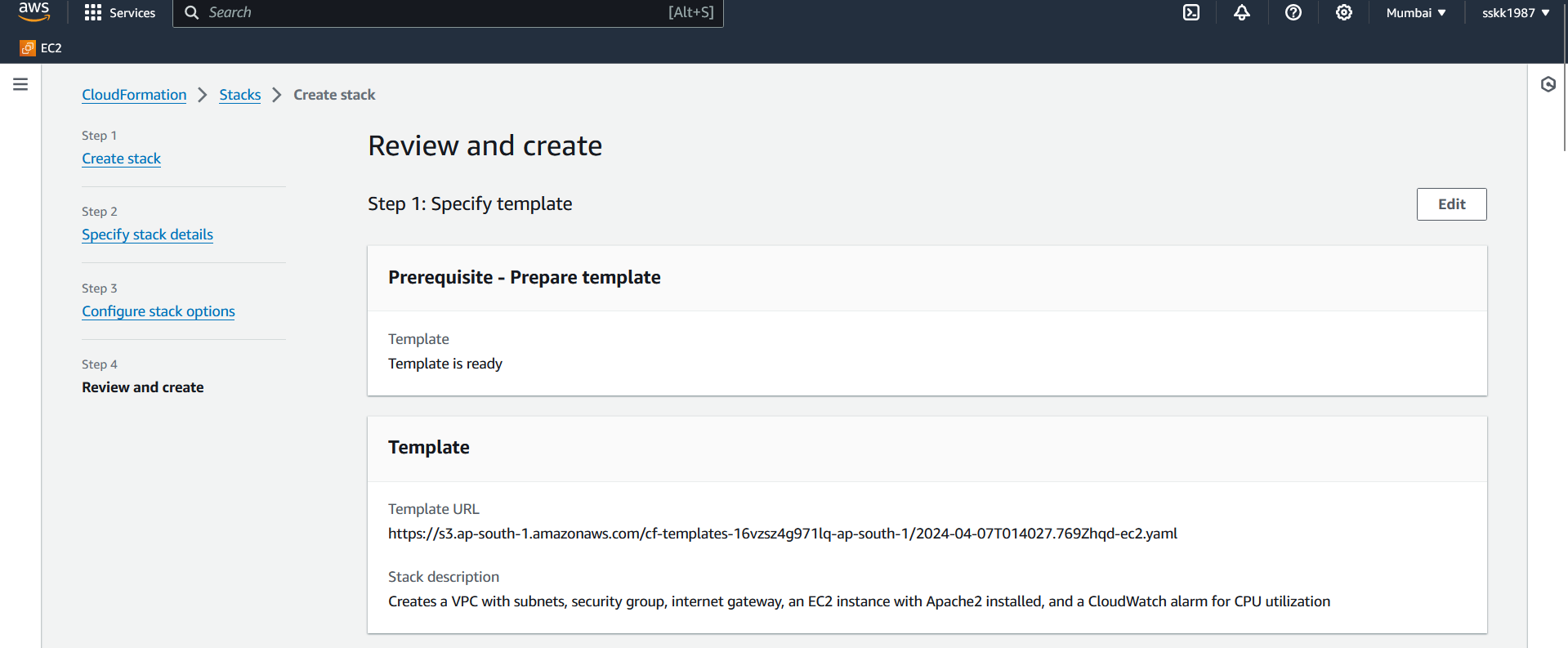
**Configure Stack Options**

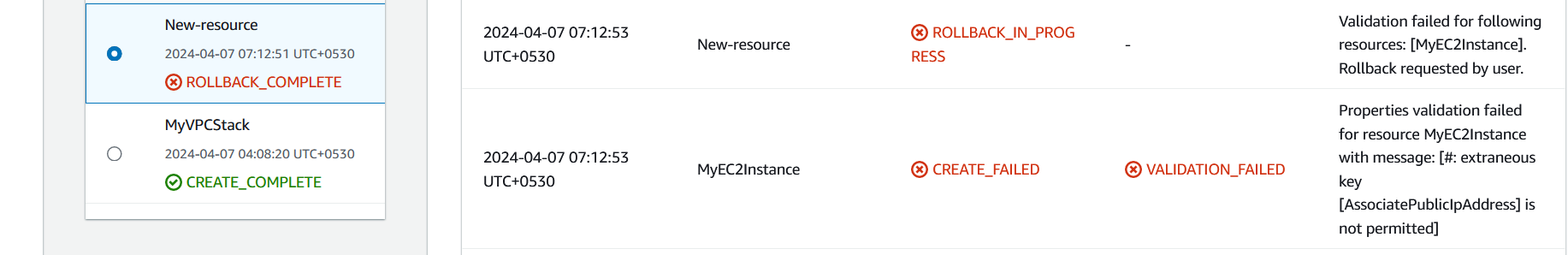
* Enter a stack name.



**Review**

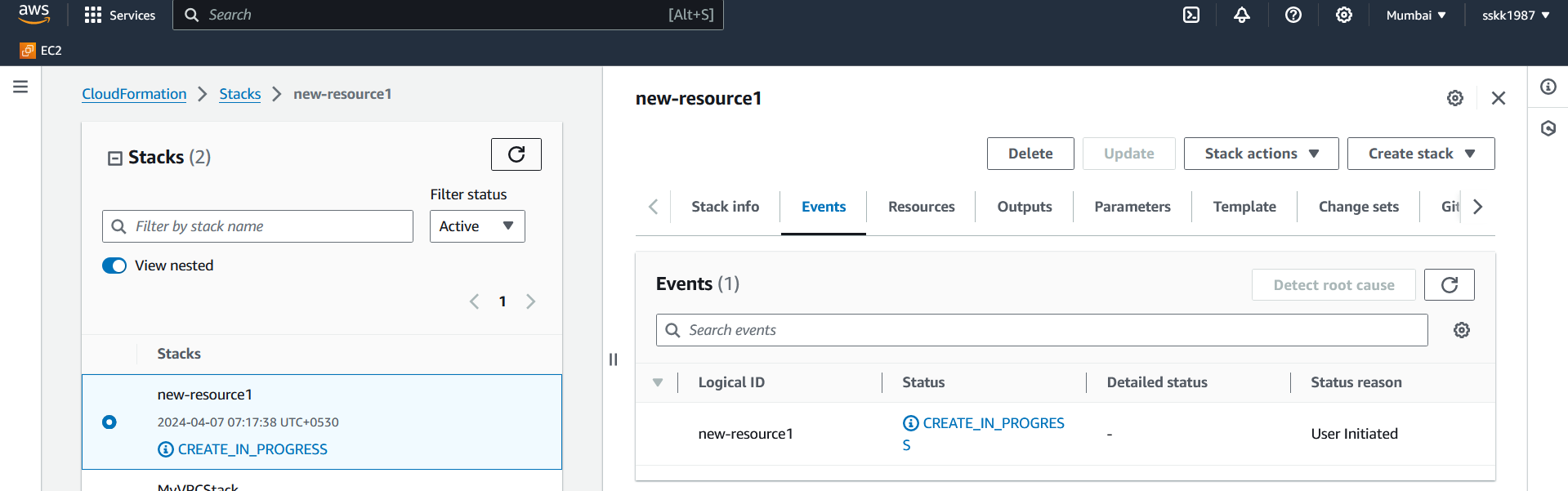
* Review the configuration.
* Click "Submit".

****

****

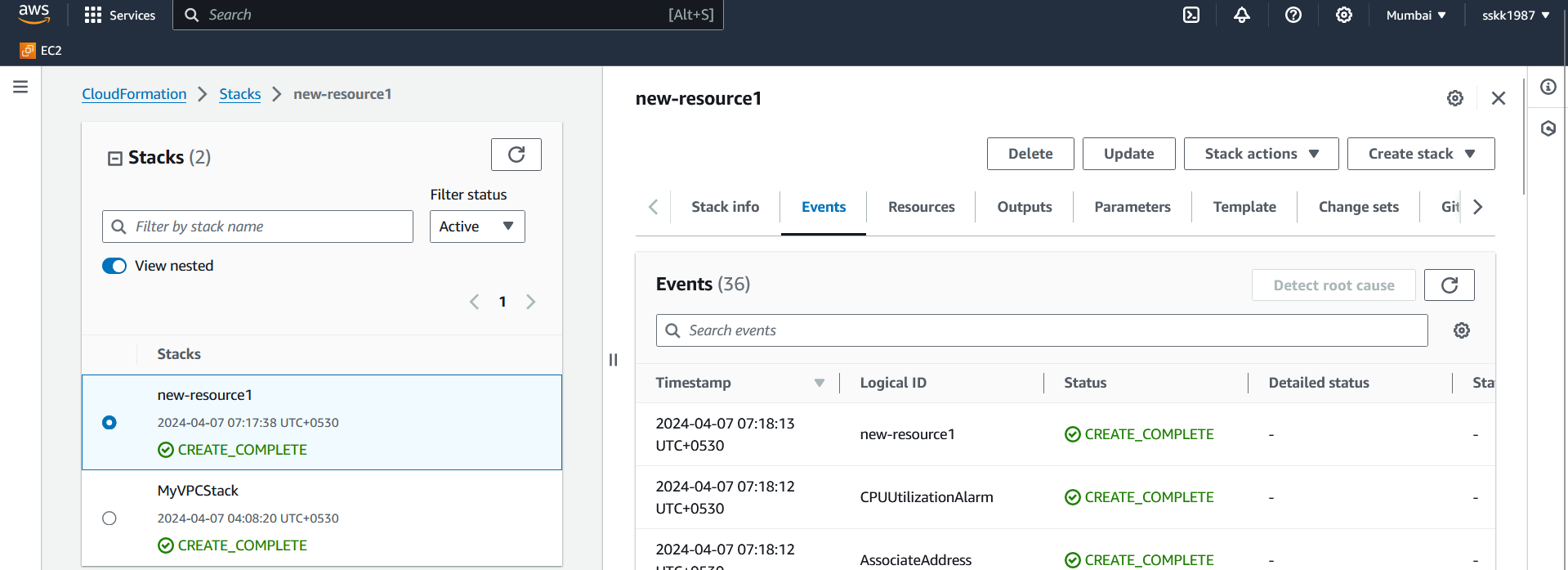
* Code is modified to allocate elastic ip, since associate public address is an option that can be used in cloudformation template using CLI

|  |
| --- |
| MyEIP:  Type: AWS::EC2::EIP  Properties:  Domain: vpc  AssociateAddress:  Type: AWS::EC2::EIPAssociation  Properties:  InstanceId: !Ref MyEC2Instance  AllocationId: !GetAtt MyEIP.AllocationId |

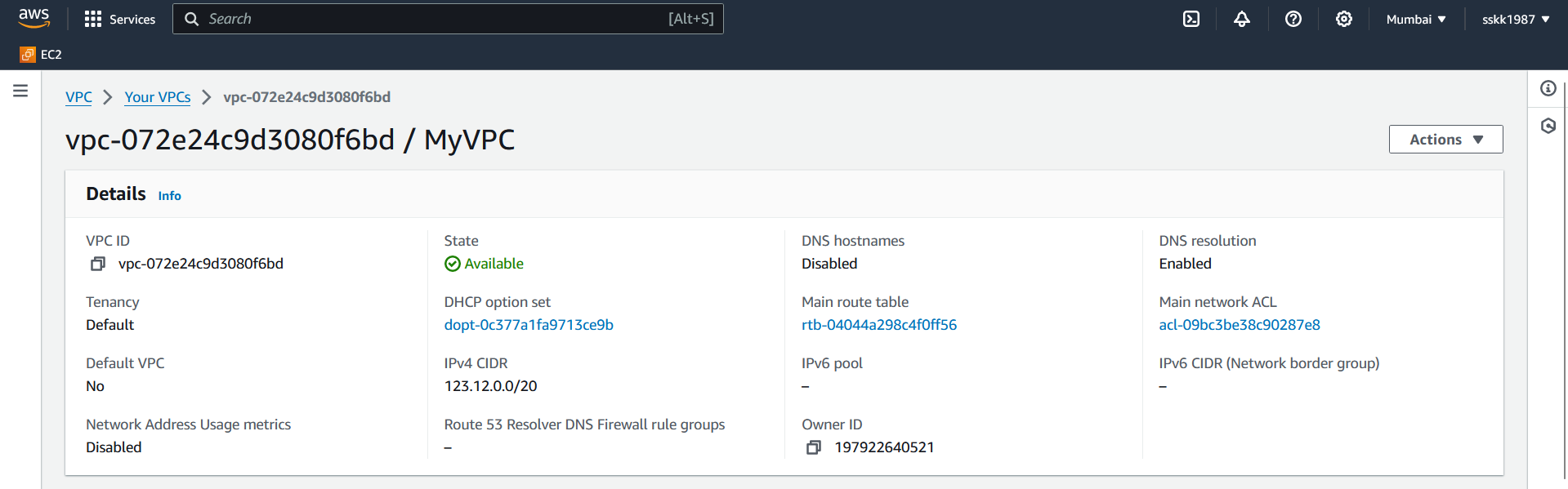


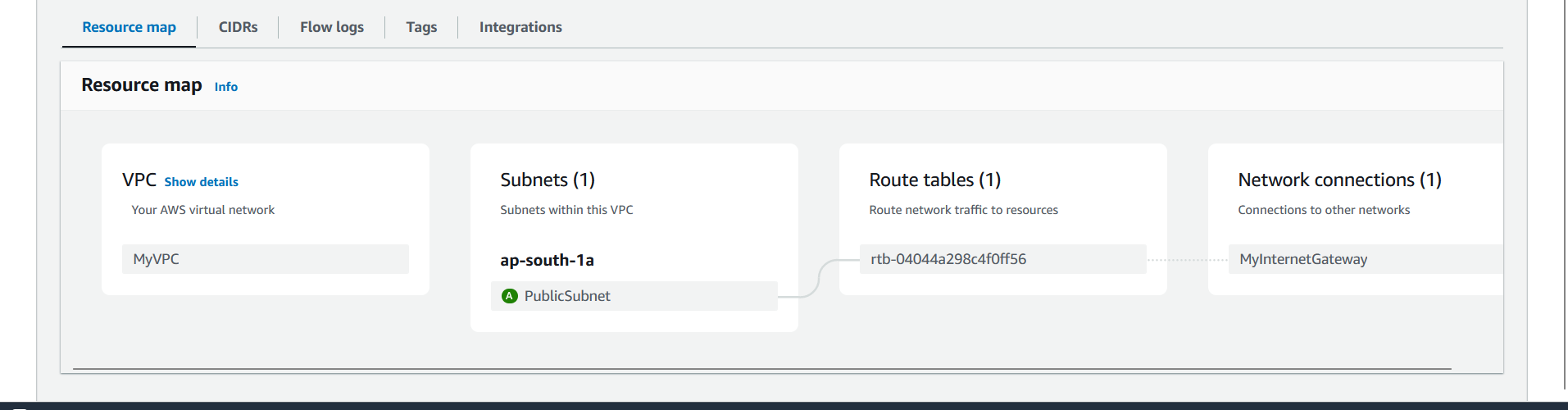
**Wait for Deployment**

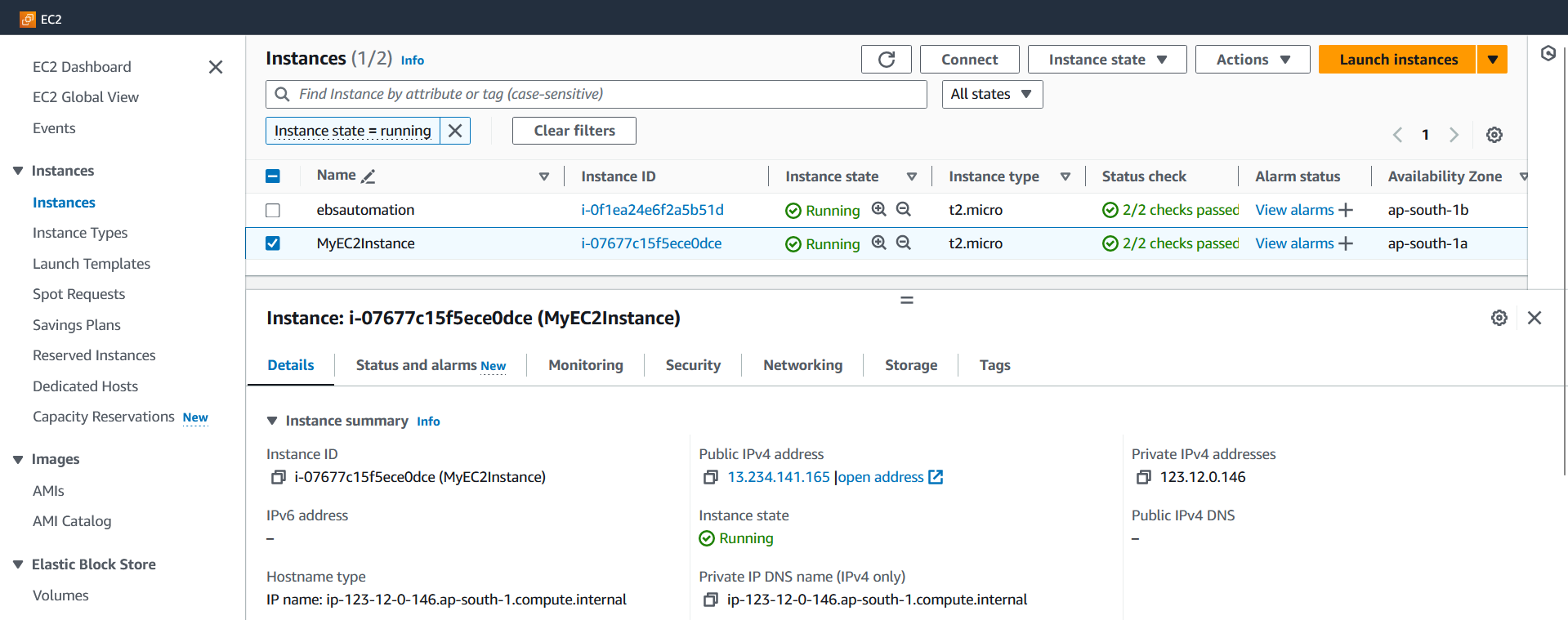
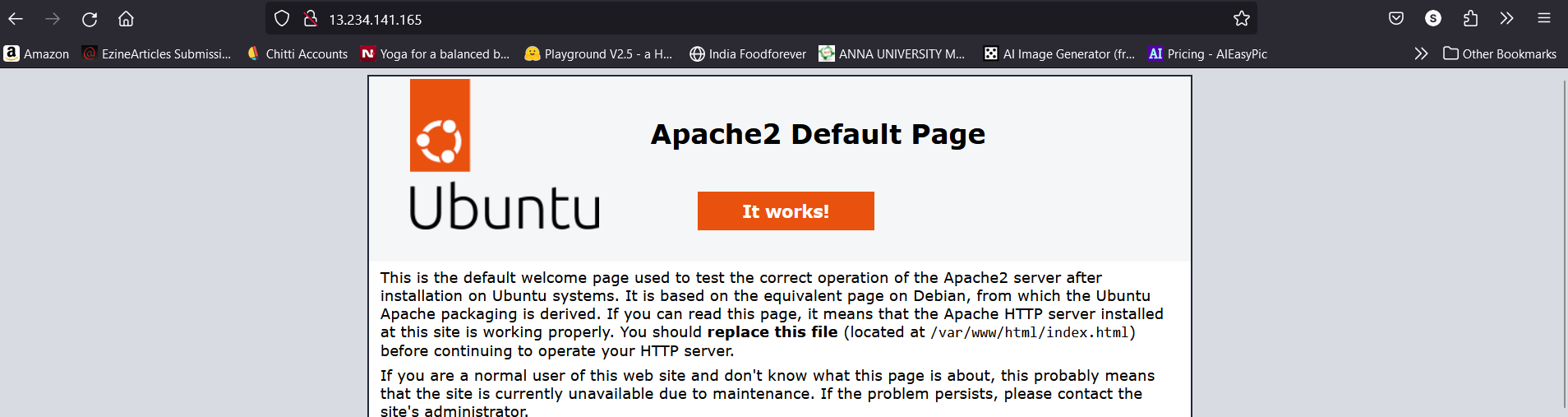
* Wait for the stack creation process to complete. This may take a few minutes.



**Verify resources**

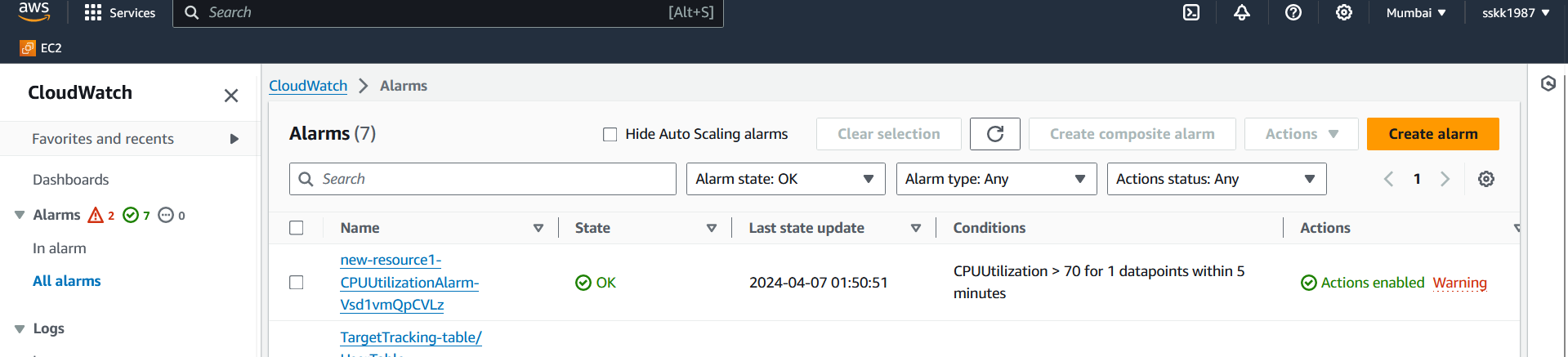




**** ****

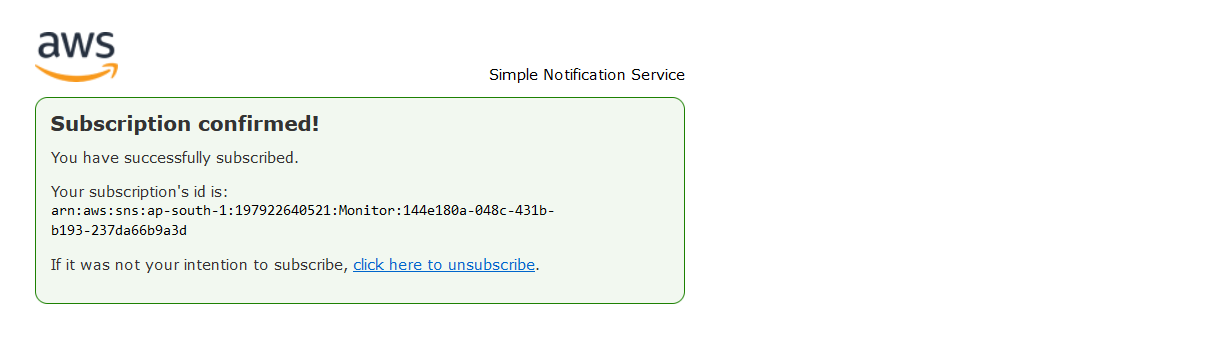
**Check CloudWatch Alarm**

 Verify that the CloudWatch alarm for CPU utilization exceeding 70% is created.

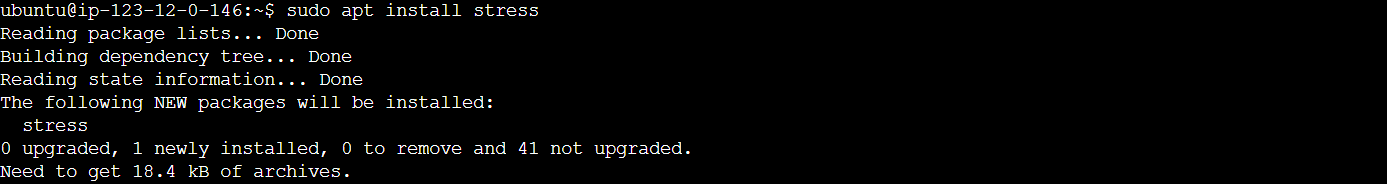


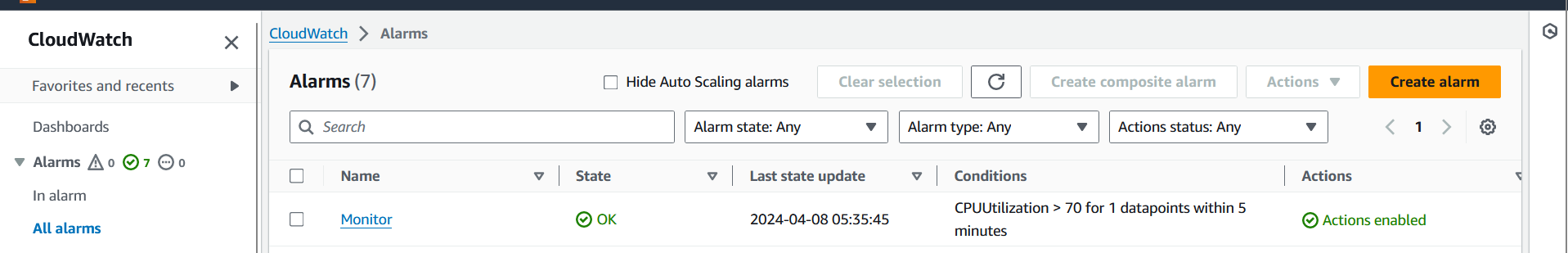
Once the alarm is created, we can configure SNS to send email regarding the cloudwatch alarms,

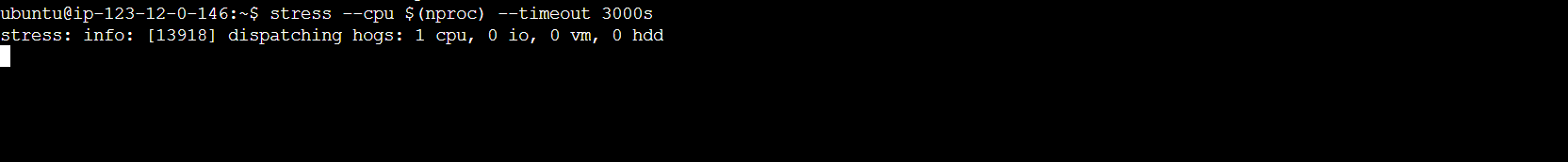




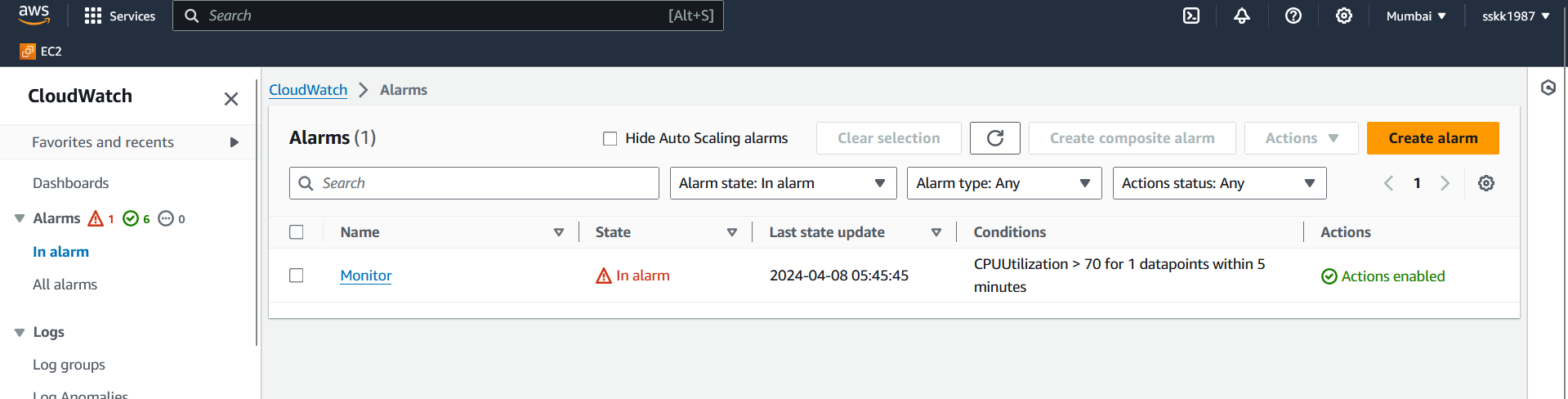
* We can simulate high CPU usage on the EC2 instance to trigger the alarm and verify its functionality.



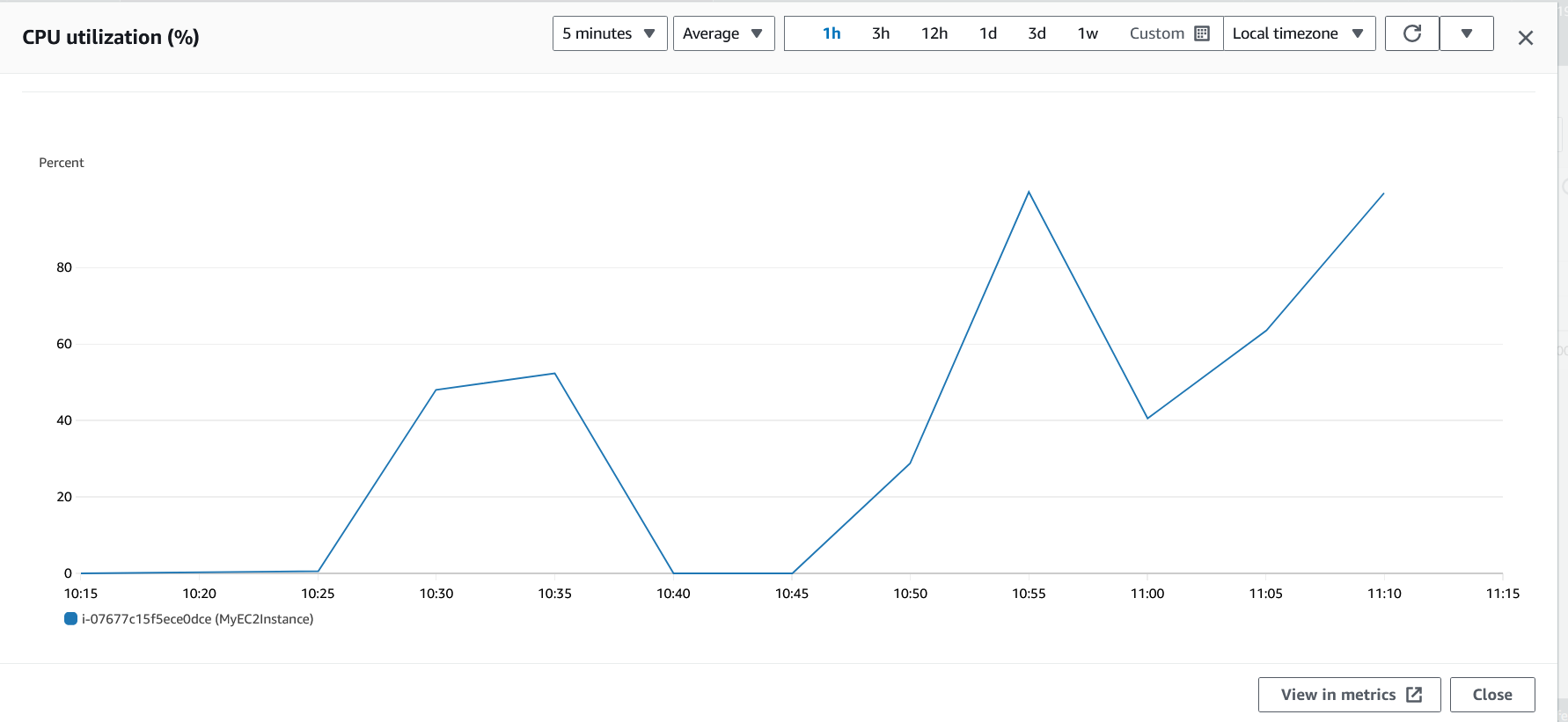


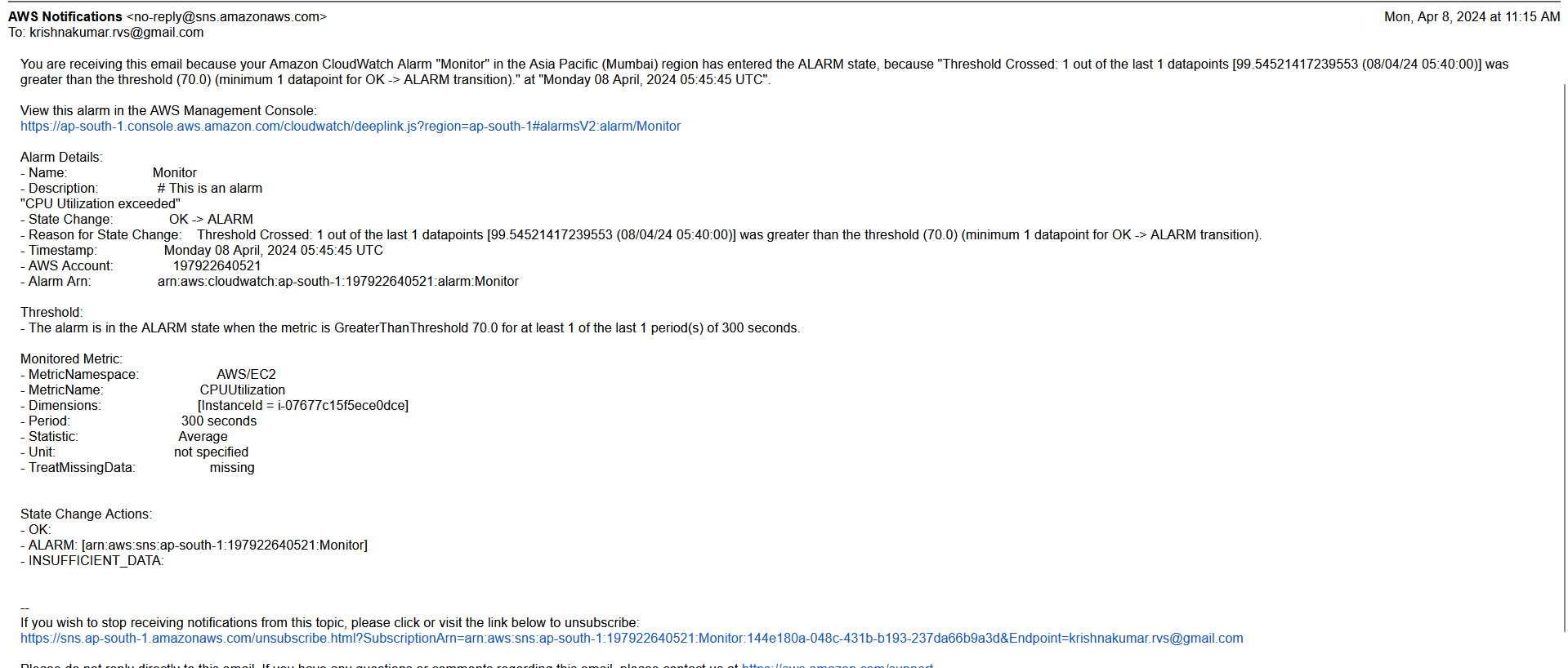


* Initially when the alarm is below threshold, there are no changes. Hence ‘In alarm” tab remains empty.



* When the threshold is breached, the Monitor alarm is moved to “In alarm” tab and corresponding actions are automatically taken. Here we have configured for SNS to trigger aa automated email to a specific id.





After the project is over or migrated, we can cleanup the resources. Here we choose to delete the stack manually. Same operations as the delete-stack command, once the command is initiated all the associated resources are deleted automatically.

