# Vodafone’s Modernization Journey to AWS

Course-end Capstone Project

**Overview:**

In this project, you will design an application running three interlinked microservices, design the infrastructure in the cloud, and implement an AWS Architecture for Vodafone’s digital services. The architecture will support the company’s digital initiatives, such as its mobile and internet services, and provide a scalable, reliable, and secure platform for its customers. You will use AWS Services such as Amazon EC2, Amazon ECS, Amazon IAM, Amazon Elastic Load balancer, AWS systems manager, AWS Certificate Manager, AWS Amplify, AWS ECR, AWS CodeCommit, AWS CodeDeploy, AWS CodePipeline, AWS CloudWatch, AWS S3, AWS Cloudfront, AWS Fargate, AWS Copilot, AWS Inspector, AWS VPC, AWS FireLens, and Cloud9 as some of the services. (Not all services may be used necessarily. It's up to learners to choose a combination of services and design the solution.)

**Objectives:**

1. Design the AWS cloud architecture for Vodafone's digital services, considering scalability, reliability, security, and cost optimization factors
2. Develop the necessary infrastructure code using AWS CloudFormation templates to provision the required resources
3. Configure the necessary networking components like Amazon VPC and set up infrastructure
4. Configure the compute resources using Amazon EC2, AWS ECS, and AWS S3/Amplify to handle Vodafone's workloads and applications
5. Configure the storage resources using Amazon S3, Amazon EBS, and Amazon EFS to store and manage Vodafone's data
6. Configure the necessary security features, such as AWS IAM roles and policies, to ensure the cloud architecture's security and compliance
7. Configure and perform all tasks per WAF (Well architected framework) pillars of operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability
8. Configure the necessary monitoring and logging features, such as AWS CloudTrail and AWS Config, to track the cloud architecture's performance and troubleshoot issues
9. Host the application, storage, and file server to the AWS platform
10. In the future, Vodafone is also thinking of utilizing automated patching, state management, and most of the tasks automatically using some capabilities from the AWS platform via systems manager (If you are using IaaS services).
11. Vodafone should be able to monitor their infrastructure using robust technologies or services that are available natively.
12. Vodafone should be able to host the container-based applications to AWS using native AWS Services. They are thinking of leveraging the CI/CD pipeline services of AWS or any CI/CD service to provide an automated deployment of apps to AWS.
13. Test and deploy the cloud architecture using AWS services, such as AWS CodeBuild, AWS CodePipeline, AWS CodeDeploy, or any other services
14. Include various observability tools, whcih give visibility to the infrastructure and can provide monitoring and reporting capabilities of the infrastructure

**Conclusion:**

By completing this project, you will showcase your expertise in designing and implementing an AWS cloud architecture, tailored to the needs of a prominent telecommunications company like Vodafone. Through hands-on experience with diverse AWS services, you will acquire valuable skills in configuring these services to establish a robust, scalable, and secure platform for digital services. Your contributions will play a pivotal role in enhancing Vodafone's digital initiatives and facilitating improved services for its customers.

## User Identification

Account ID: **4926-8423-6516** - IAM user: **odl\_user\_1149497**

[gronzul1/capstone-1 (github.com)](https://github.com/gronzul1/capstone-1)

## AWS Services used

* S3 Bucket
* EC2
* VPC
* CloudFormation
* CloudWatch
* IAM
* Route53
* Cloudfront

Contents

[Vodafone’s Modernization Journey to AWS 1](#_Toc151133341)

[User Identification 2](#_Toc151133342)

[AWS Services used 2](#_Toc151133343)

[Project Diagram 4](#_Toc151133344)

[IAM User access KEY 5](#_Toc151133345)

[S3 Bucket 5](#_Toc151133346)

[Script Execution Results: 8](#_Toc151133347)

[Deploy Web Site 9](#_Toc151133348)

[Files copy results: 9](#_Toc151133349)

[Create VPC and EC2 resources 9](#_Toc151133350)

[Create CloudFormation Stack to deploy VPC and related resources: 10](#_Toc151133351)

[Create EC2 resources with Cloudformation 13](#_Toc151133352)

[Create stack with EC2.yaml file and configure required params: 13](#_Toc151133353)

[Update EC2.yaml script with valid AMI image: 15](#_Toc151133354)

[Connect to EC2 instance via RDP and decrypt Administrator password with Keypair: 16](#_Toc151133355)

[Install IIS Server with Management Tools via powershell: 17](#_Toc151133356)

[Configure Security Group to allow Http access to instance: 17](#_Toc151133357)

[Deploy microservices on EC2 instance 18](#_Toc151133358)

[Test website deploy on browser: 20](#_Toc151133359)

[Select Android branch 20](#_Toc151133360)

[Select Apple branch 21](#_Toc151133361)

[Website Heath check 21](#_Toc151133362)

[View EC2 statics with CloudWatch 23](#_Toc151133363)

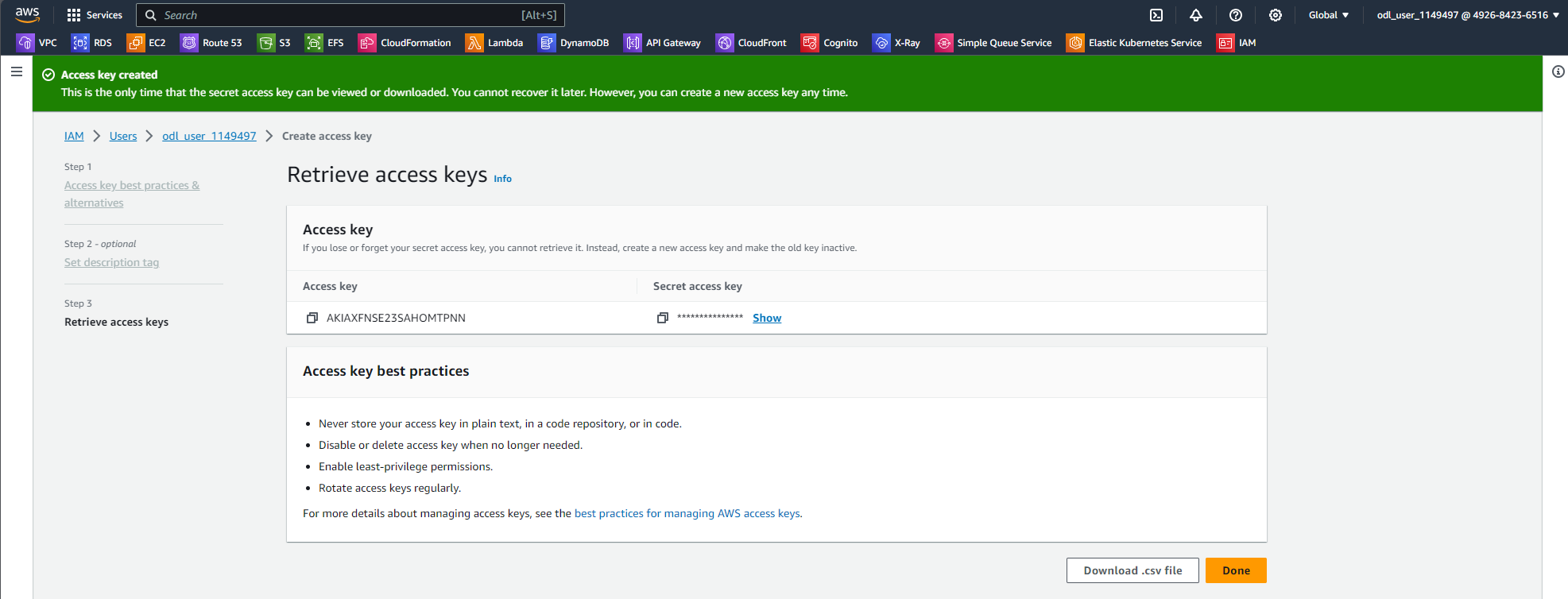
## Project Diagram

A screenshot of a computer

Description automatically generated

## IAM User access KEY

Create user access key for enable VSCode CLI: I created the access keys for the user odl\_user\_1149497 to be able to use the aws-sdk via CLI



Access key: **AKIAXFNSE23SAHOMTPNN**

Secret access key: **enYKxRFPWf2aI7McZzf0K5+zfuF+yUMFpLhaNboG**

## S3 Bucket

Create and configure S3 Bucket: the first step is to create an S3 bucket to contain the source code of the microservices

to achieve this I used the aws-sdk for JavaScript v3 **@aws-sdk/client-s3**

//#region fx S3

const s3 = new S3Client(config);

export const listS3 = async () => {

    console.log("listS3...")

    const input = {};

    const command = new ListBucketsCommand(input);

    return await s3.send(command) || "";

}

export const deleteS3 = async () => {

    console.log("deleteS3...")

    const input = {

        "Bucket": BUCKETNAME

    };

    const command = new DeleteBucketCommand(input);

    return await s3.send(command);

}

export const createS3 = async () => {

    console.log("createS3...")

    const input = {

        "Bucket": BUCKETNAME

    };

    const command = new CreateBucketCommand(input);

    return await s3.send(command);

}

export const allowPublic = async () => {

    console.log("allowPublic...")

    const input = { // DeletePublicAccessBlockRequest

        Bucket: BUCKETNAME

    };

    const command = new DeletePublicAccessBlockCommand(input);

    return await s3.send(command);

}

export const enableWebSite = async () => {

    console.log("enableWebSite...")

    const input = {

        "Bucket": BUCKETNAME,

        'WebsiteConfiguration':

        { // WebsiteConfiguration

            IndexDocument: { // IndexDocument

                "Suffix": "index.html"

            },

        }

    };

    const command = new PutBucketWebsiteCommand(input);

    return await s3.send(command);

}

export const assignS3Policy = async () => {

    console.log("assignS3Policy...")

    const input = {

        "Bucket": BUCKETNAME,

        "Policy": JSON.stringify(

            {

                "Id": "Policy1693310659772",

                "Version": "2012-10-17",

                "Statement": [

                    {

                        "Sid": "Stmt1693310658340",

                        "Action": [

                            "s3:GetObject",

                            // "s3:PutObject"

                        ],

                        "Effect": "Allow",

                        "Resource": "arn:aws:s3:::" + BUCKETNAME + "/\*",

                        "Principal": "\*"

                    }

                ]

            }

        )

    };

    const command = new PutBucketPolicyCommand(input);

    return await s3.send(command);

}

function creaS3() {

    createS3().then(data => {

        console.log(data.Location);

        allowPublic().then(data => {

            enableWebSite().then(data => {

                assignS3Policy().then(data => {

                })

            })

        })

    })

}

//#endregion

### Script Execution Results:



S3 Static website hosting Url: <http://gr-cap-1.s3-website-us-east-1.amazonaws.com>

## Deploy Web Site

ecs-images into S3 Bucket:I downloaded ecs-images into local machine from github repository :

$> git clone https://github.com/simplilearnaz305/ecs-images.git

and I copied the files and directories to S3Bucket via CLI:

$>   aws s3 cp . s3://gr-cap-1 –recursive

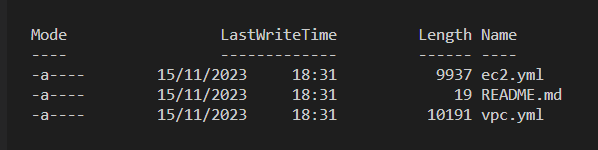
### Files copy results:



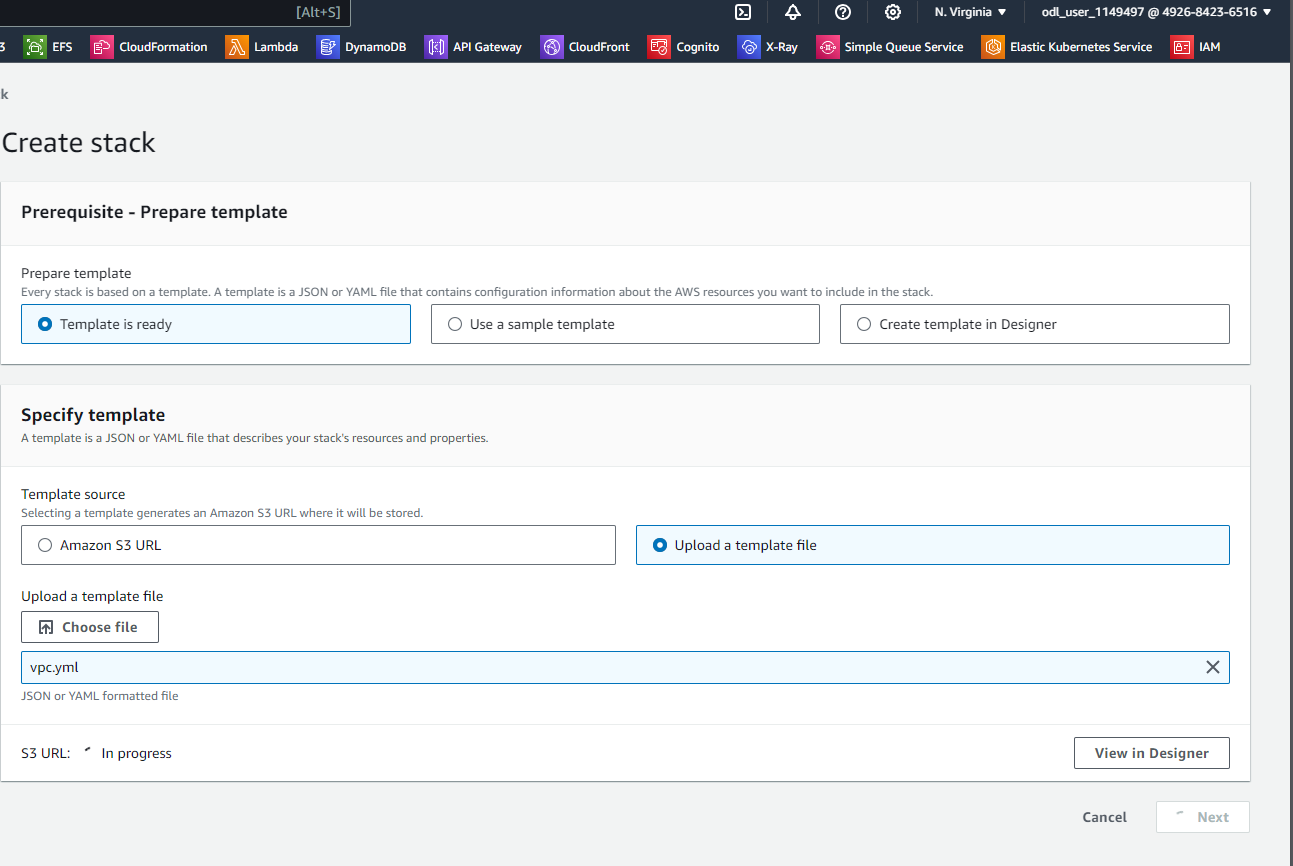
## Create VPC and EC2 resources

CloudFormation: the necessary infrastructure and resources were created using a CloudFormation stack available at the following url:

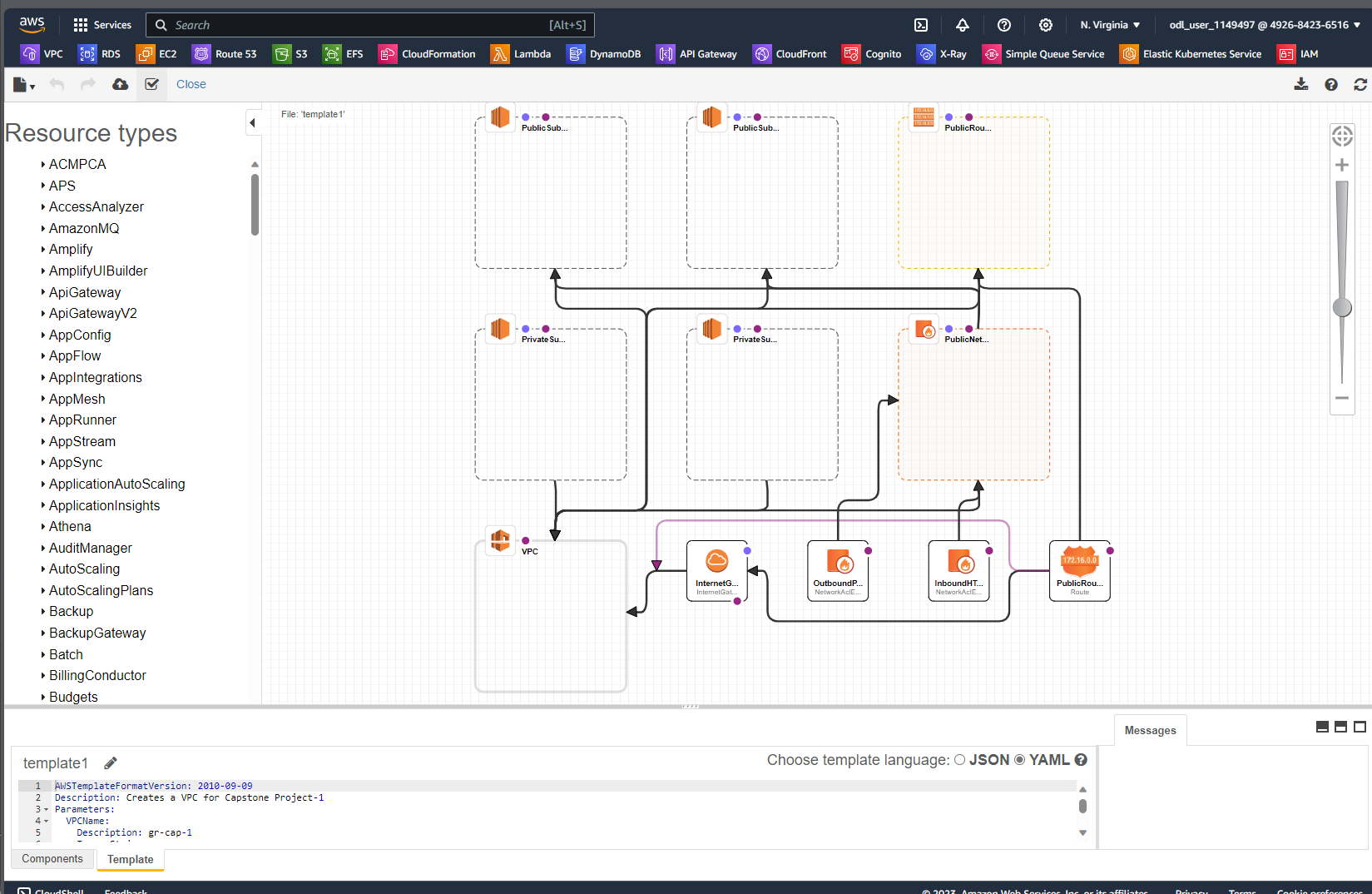
$> git clone https://github.com/simplilearnaz305/serverbasedms-vpc.git



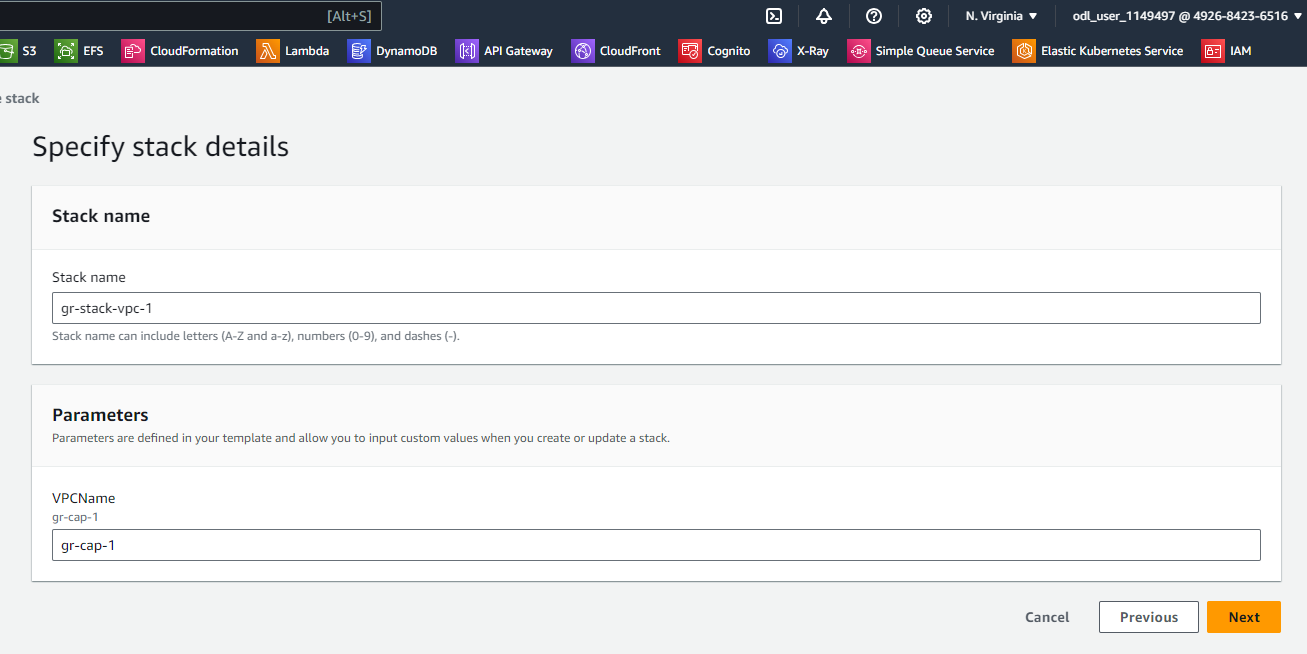
### Create CloudFormation Stack to deploy VPC and related resources:

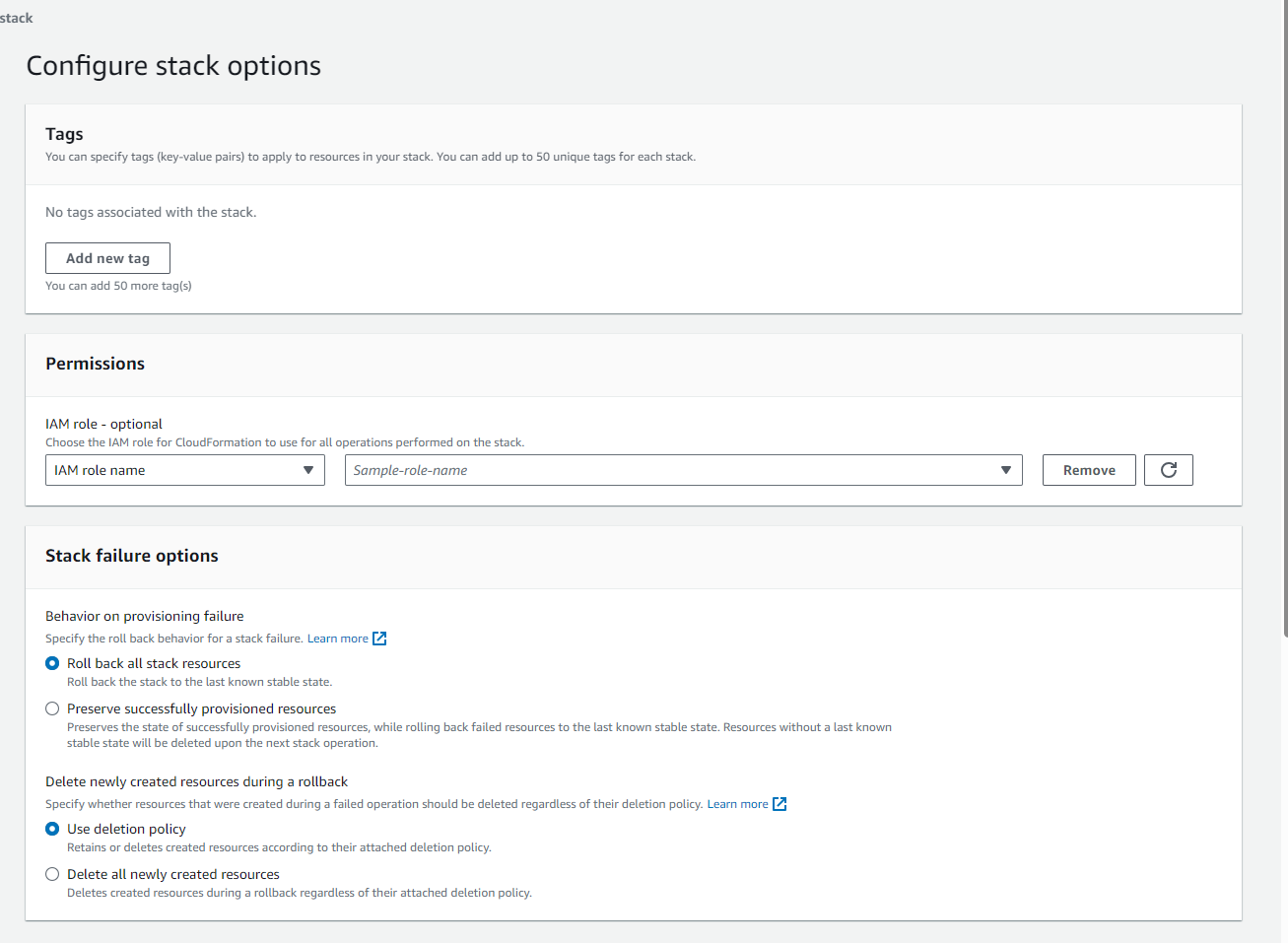


#### View file vpc.yaml in designer:

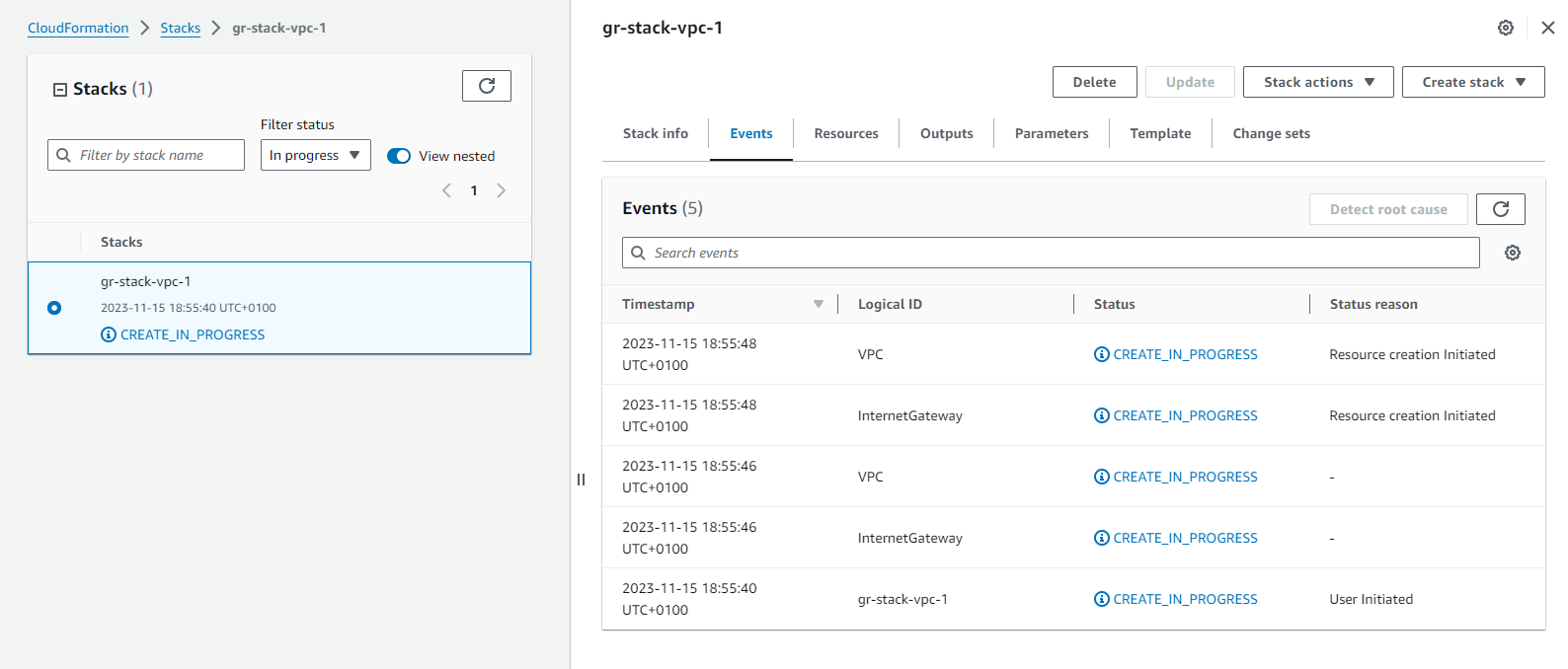


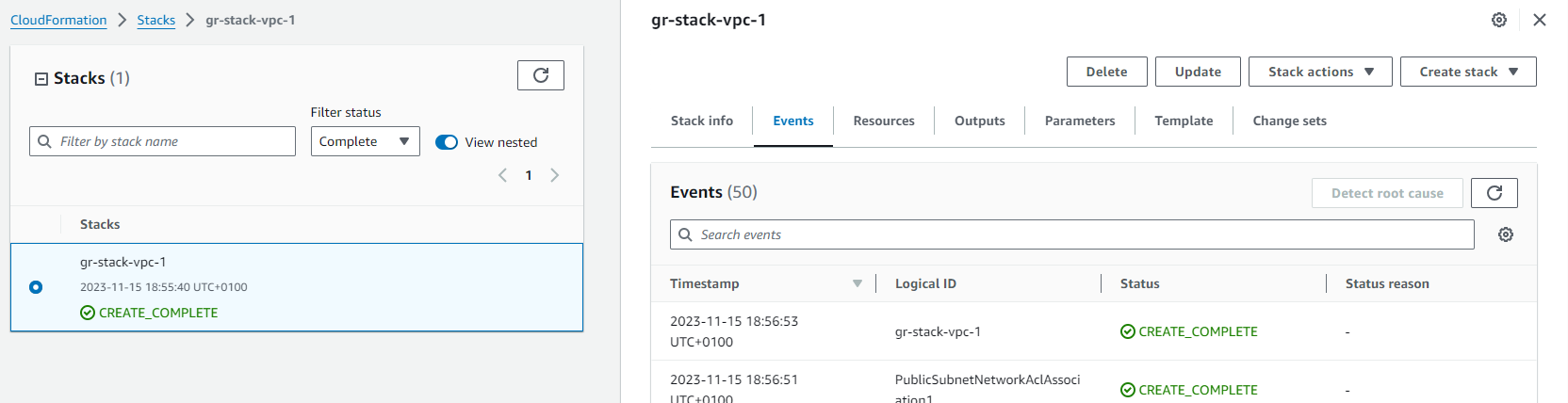
#### Create stack:



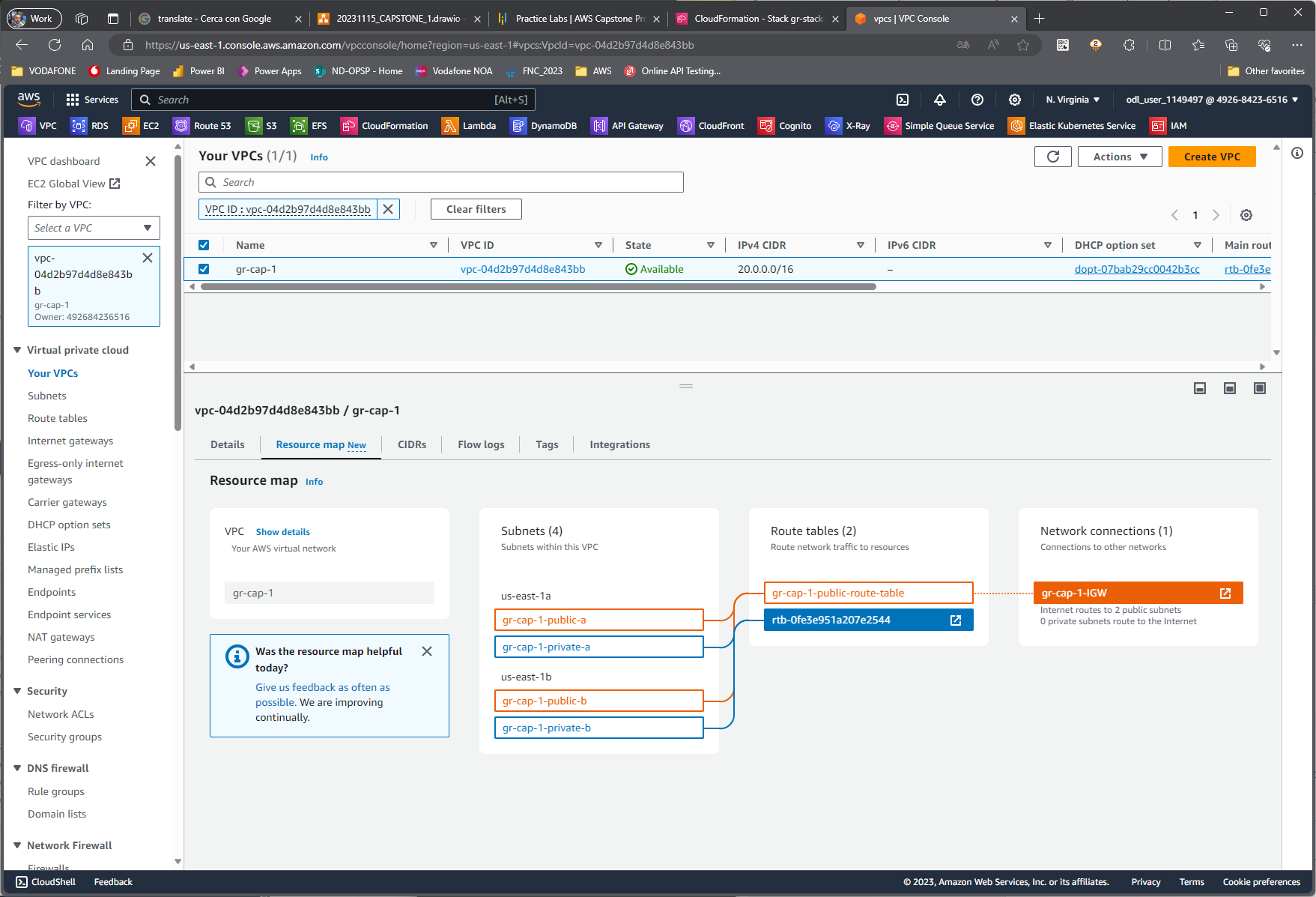


#### Stack creation results:

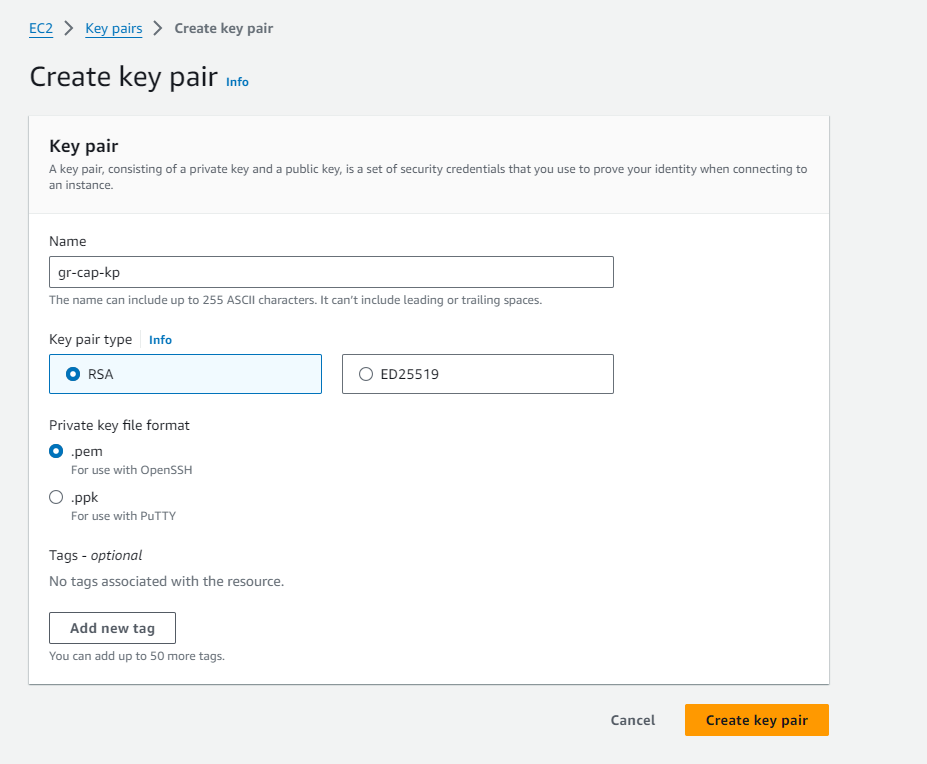




#### VPC creation resuts:



## Create EC2 resources with Cloudformation

Pre-req: create keypair with Aws EC2 console for RDP access:  


File Keypair generated:



### Create stack with EC2.yaml file and configure required params:

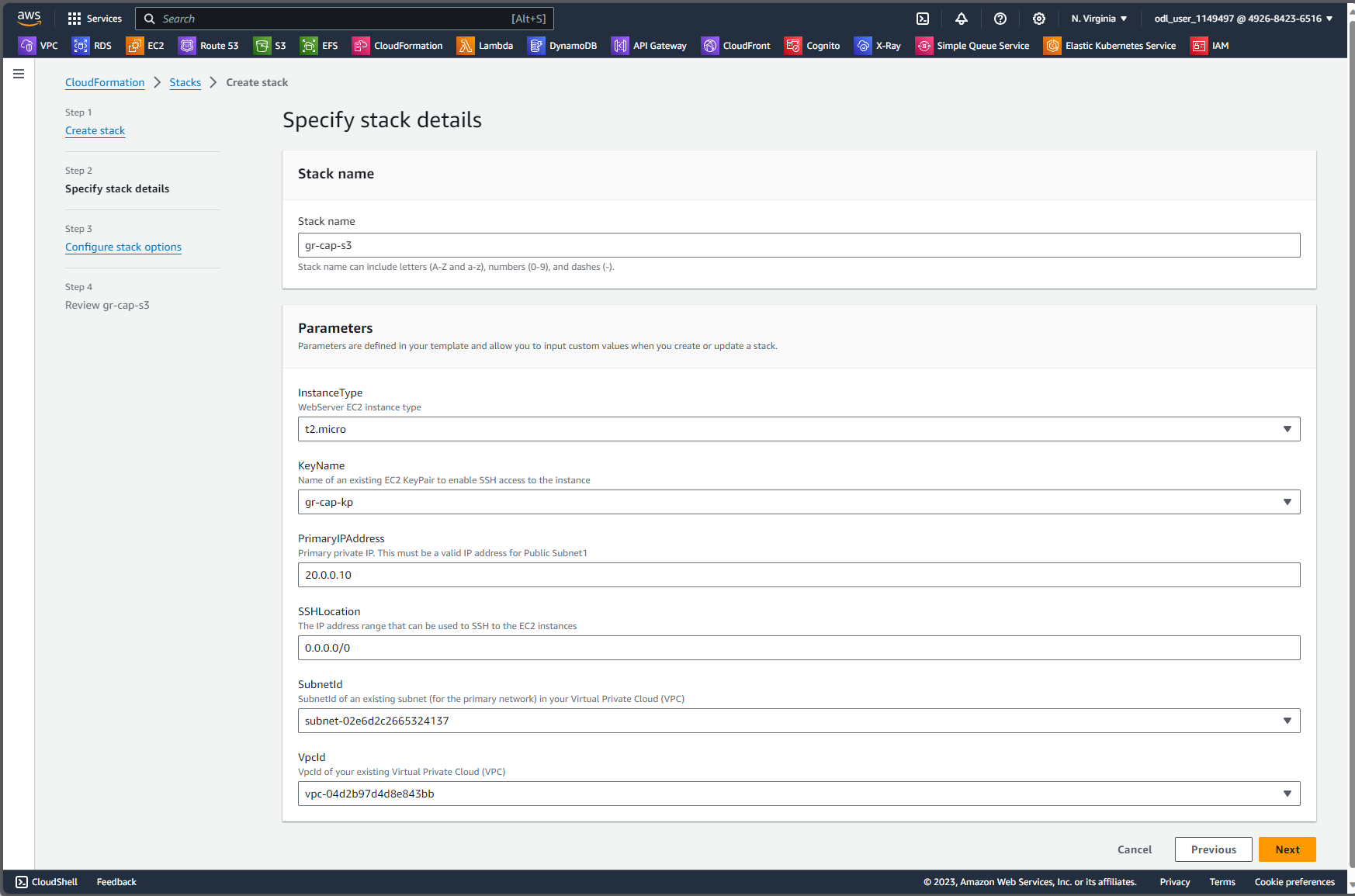
VpcId: **gr-cap-1**

SubnetId: **gr-cap-1-public-a**

PrimaryIPAddress: accordly with subnet 20.0.0.10

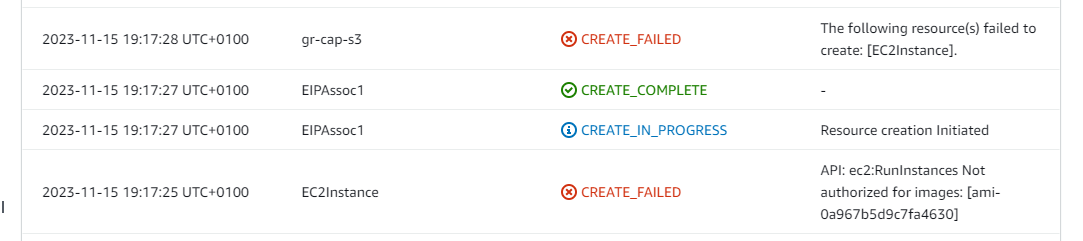
KeyName: **gr-cap-kp**

Stack failure options: **Preserve successfully provisioned resources**

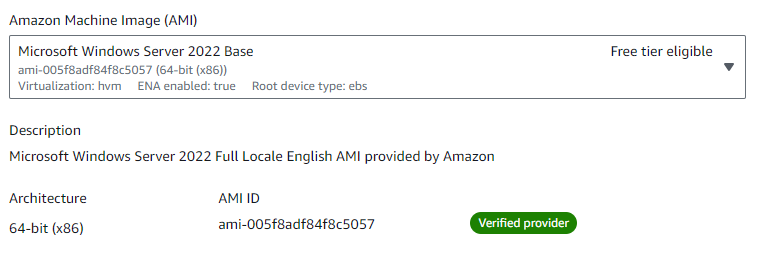


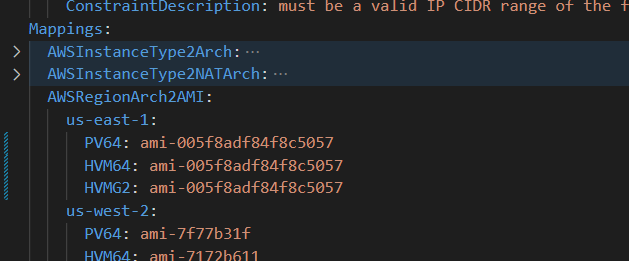


#### Stack creation failed with following error:

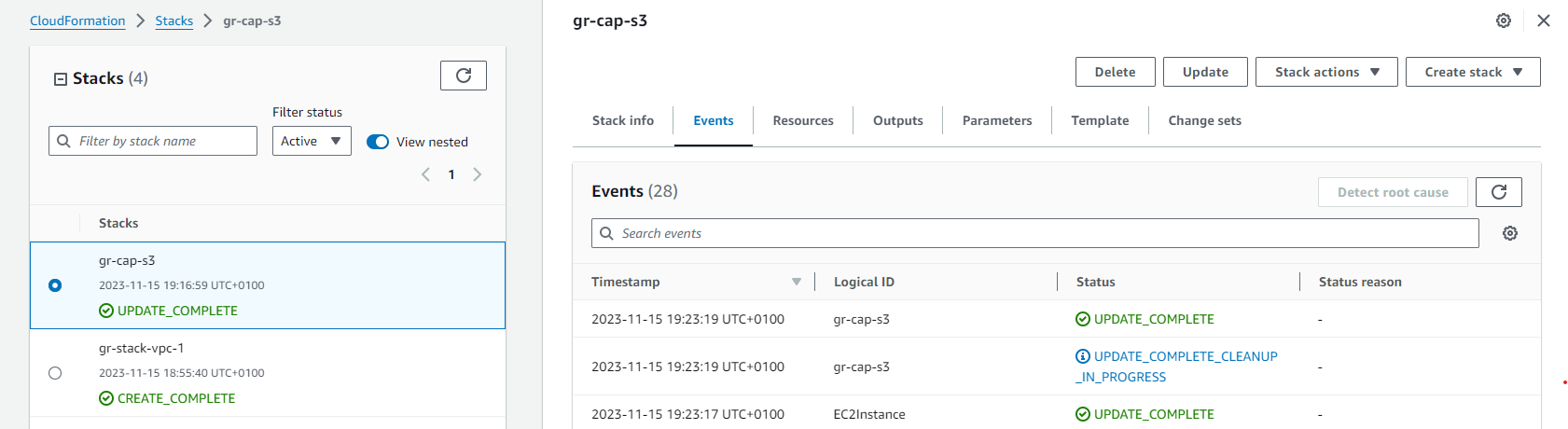


### Update EC2.yaml script with valid AMI image:



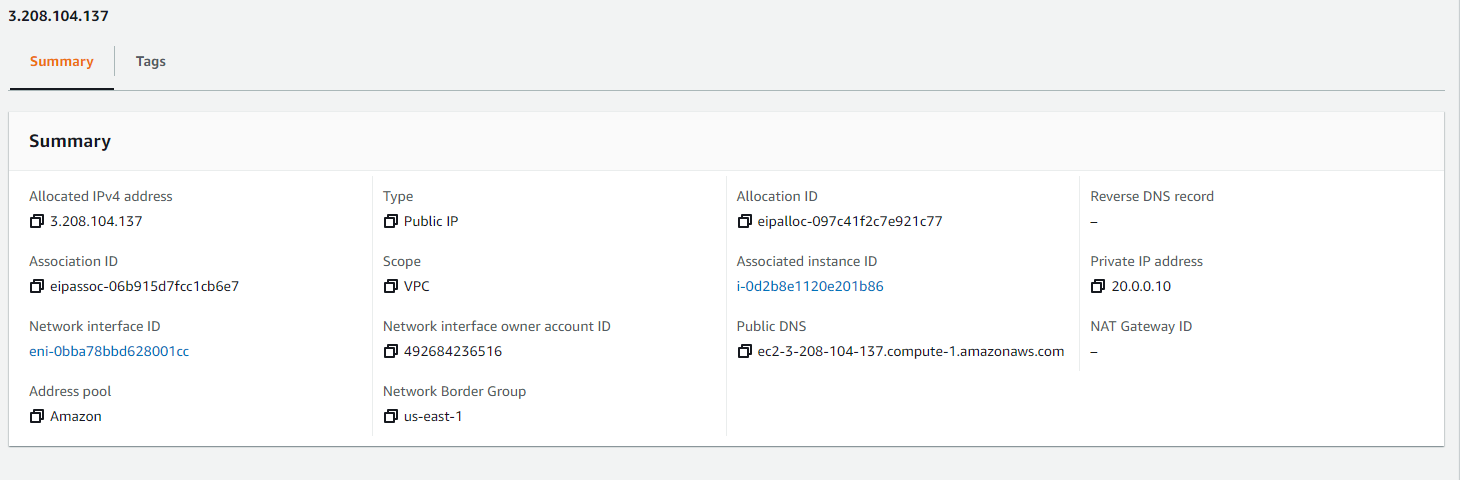


#### Update stack and run new values:

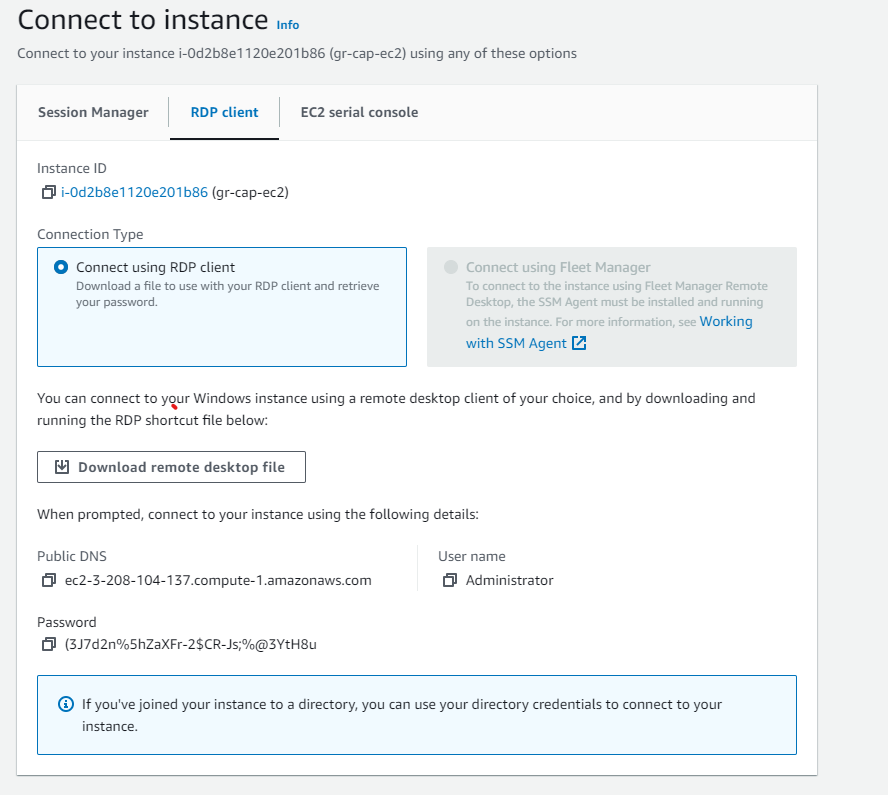


#### Verify EIP association for EC2:

3.208.104.137/ ec2-3-208-104-137.compute-1.amazonaws.com 🡺 20.0.0.10



### Connect to EC2 instance via RDP and decrypt Administrator password with Keypair:

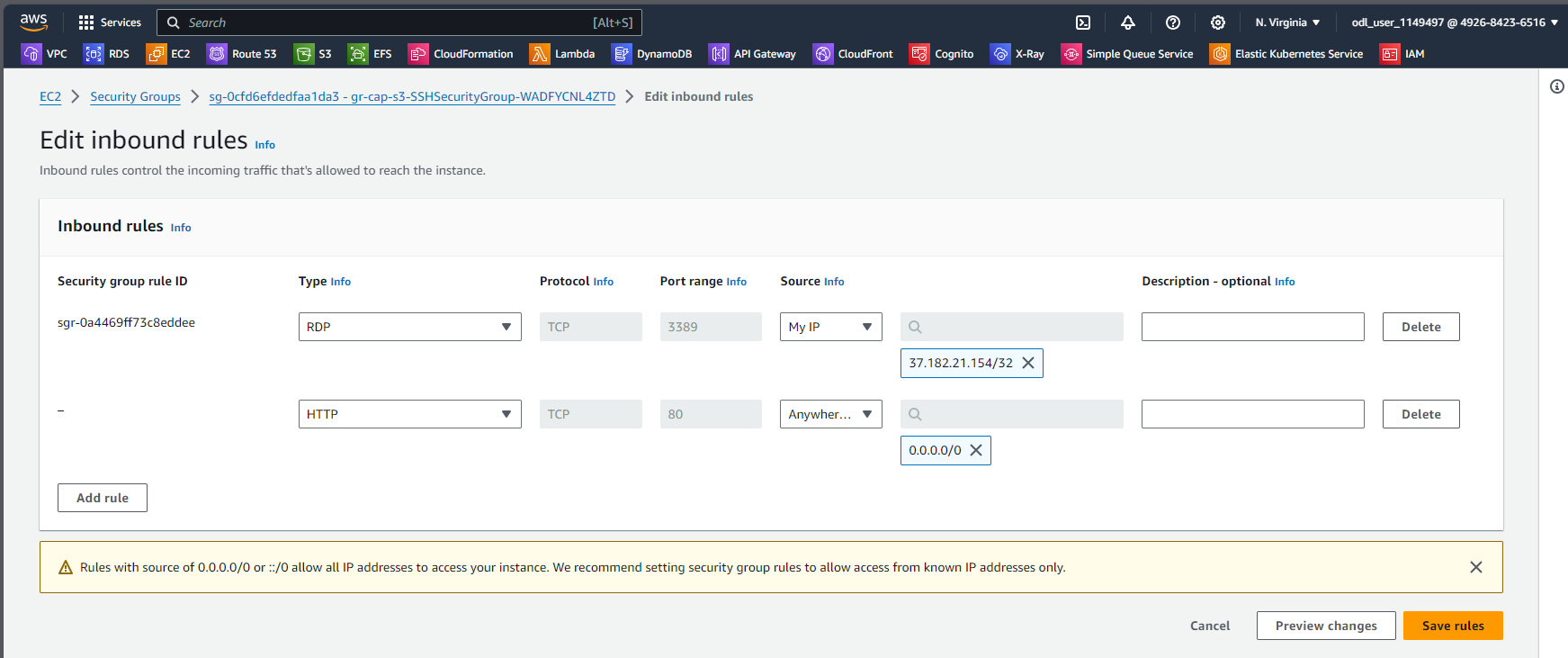


PWD: (3J7d2n%5hZaXFr-2$CR-Js;%@3YtH8u

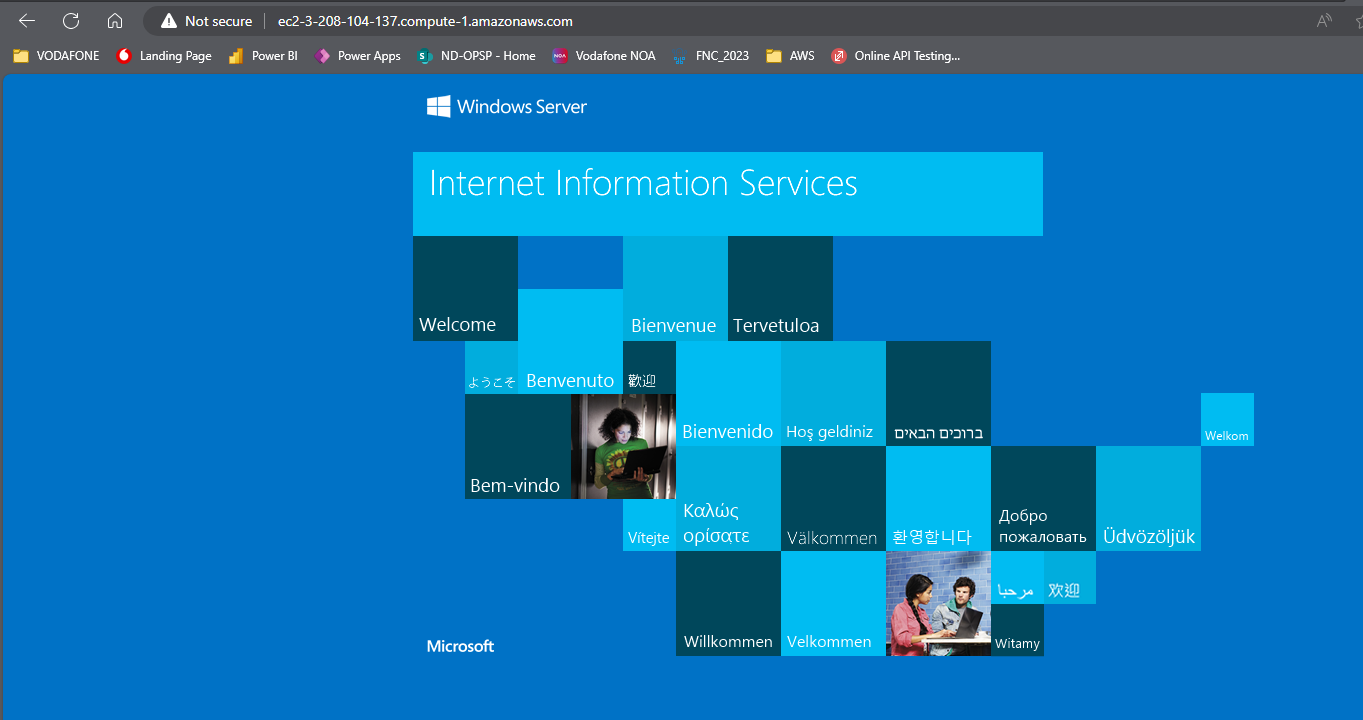
### Install IIS Server with Management Tools via powershell:



### Configure Security Group to allow Http access to instance:



And check rule: <http://ec2-3-208-104-137.compute-1.amazonaws.com/>

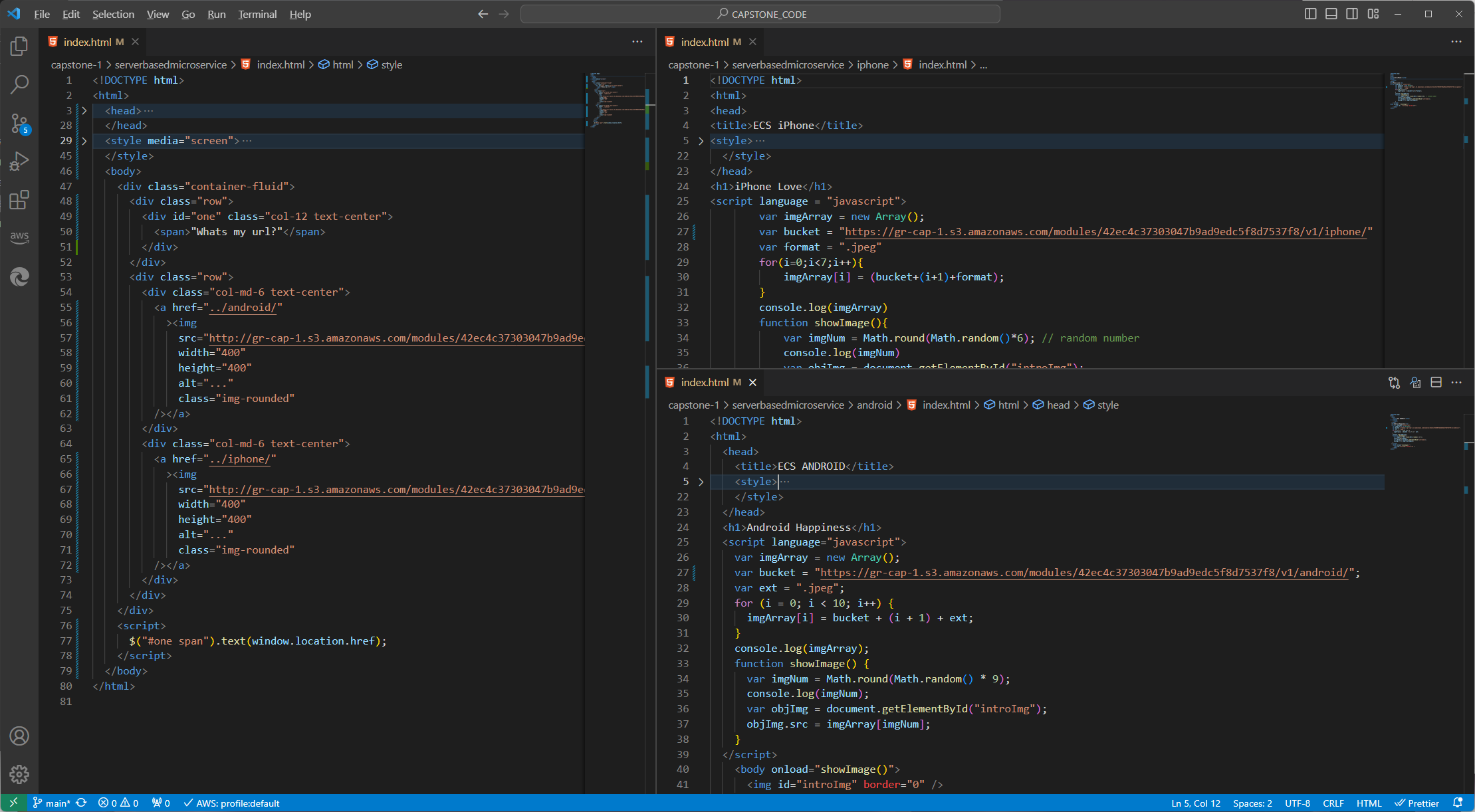


## Deploy microservices on EC2 instance

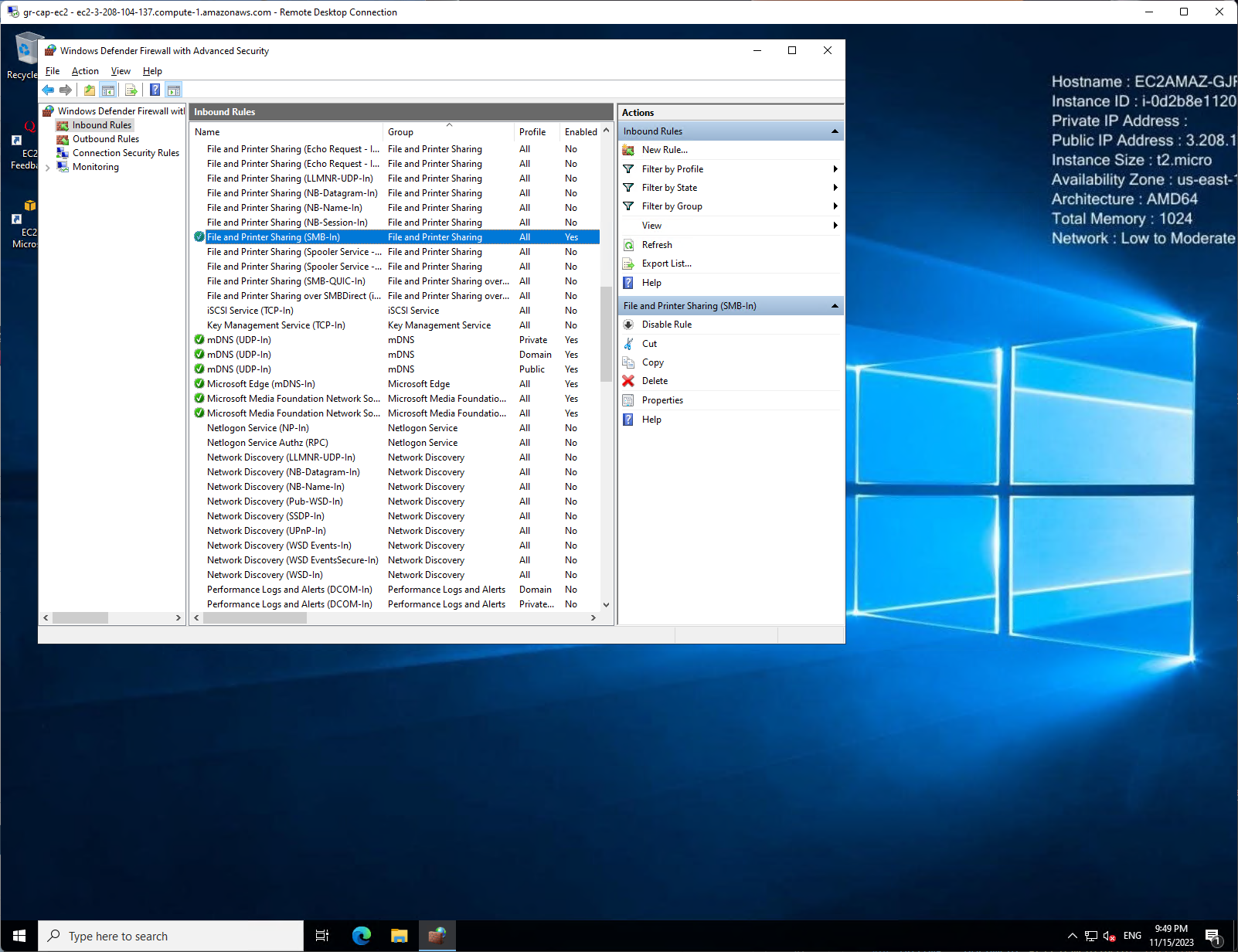
Download serverbasedmicroservice into local machine from github repository:

$> git clone https://github.com/simplilearnaz305/serverbasedmicroservice.git

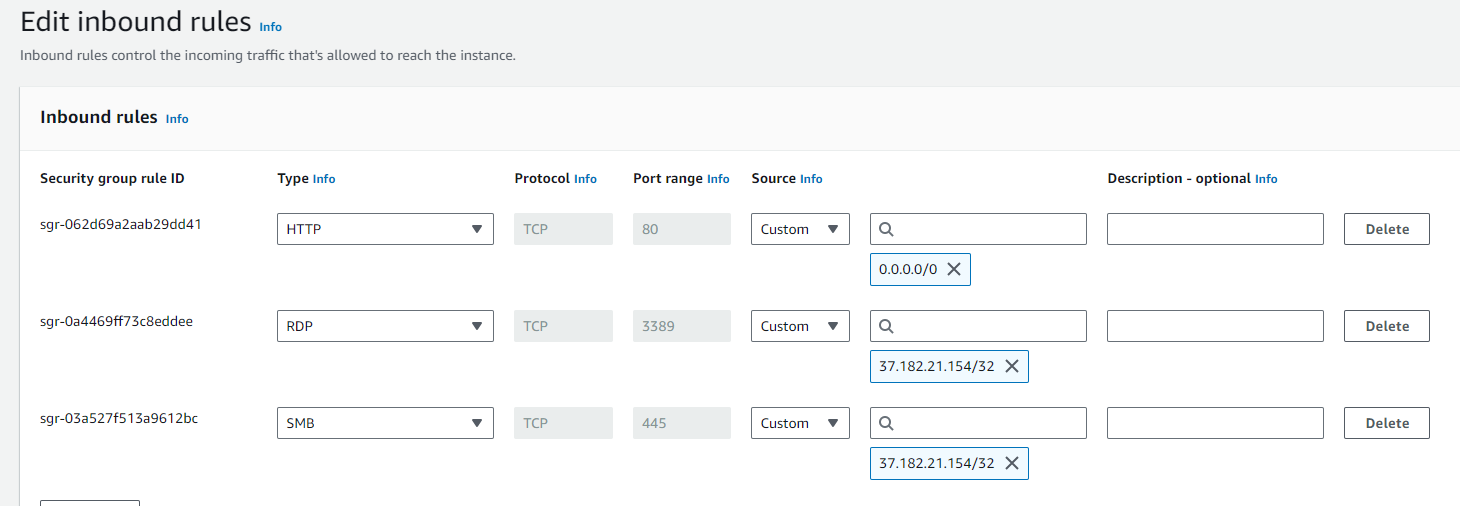
Edit and update each index.html file to point to the URL of static web site on s3 bucket:



#### Enable file&printer sharing (Samba) rule on Windows Firewall to enable copy file from local PC to Server:



#### Configure Security group to allow SMB access to instance from myIP:

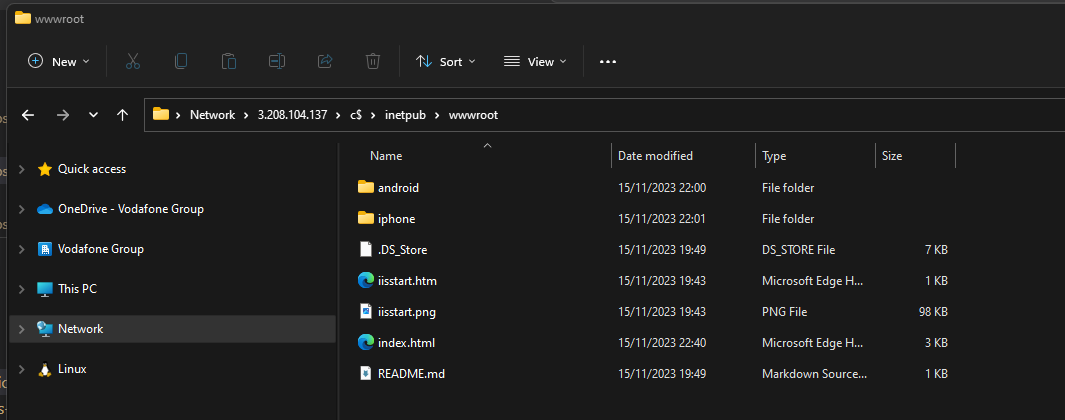


#### Access to administrative share with Administrator Credentials:

[\\3.208.104.137\c$](file:///\\3.208.104.137\c$)

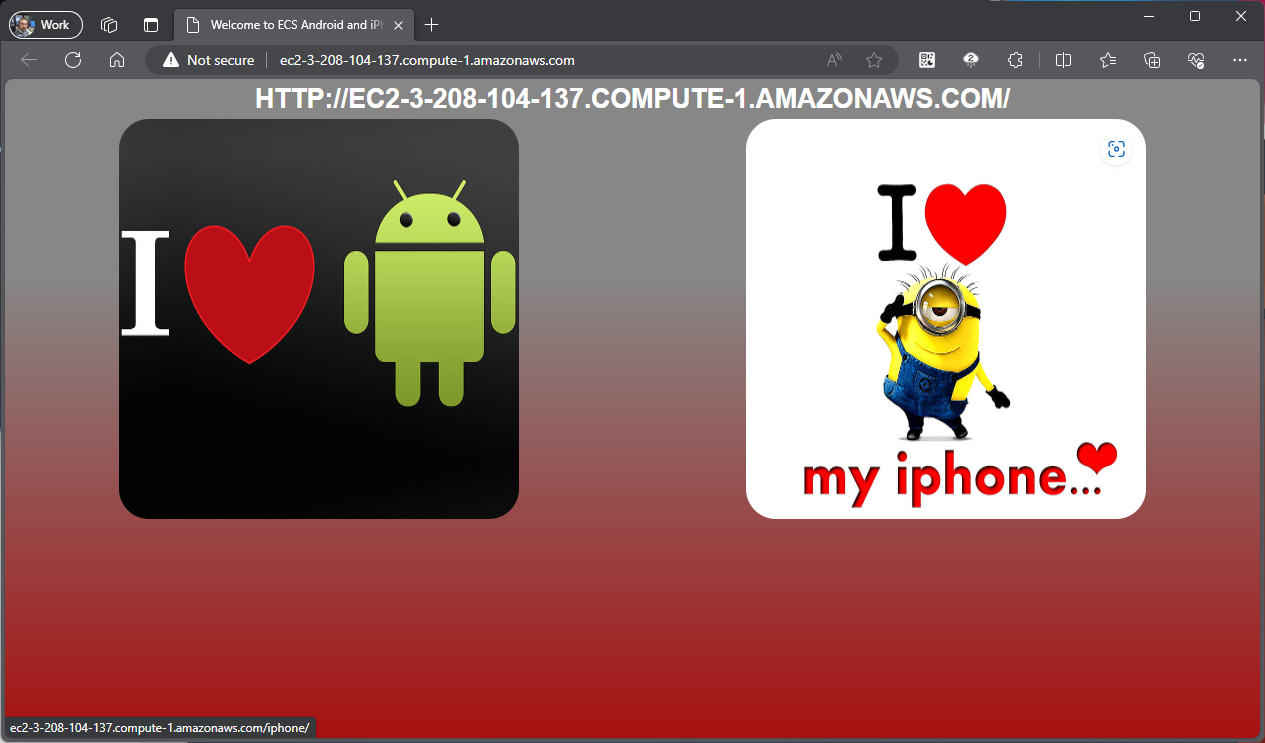
#### Copy modified serverbasedmicroservice files and directory

Copy modified serverbasedmicroservice files and directory to [\\3.208.104.137\c$\inetpub\wwwroot](file:///\\3.208.104.137\c$\inetpub\wwwroot)

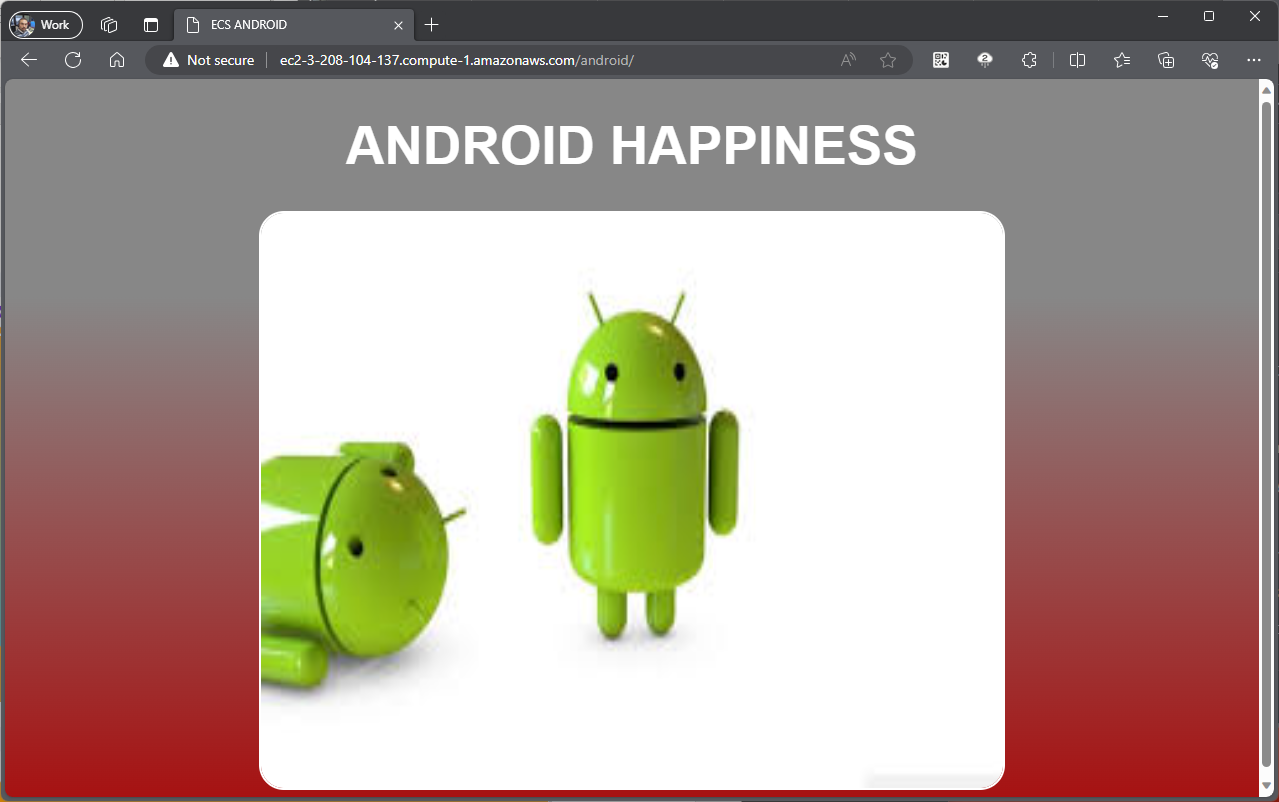


## Test website deploy on browser:

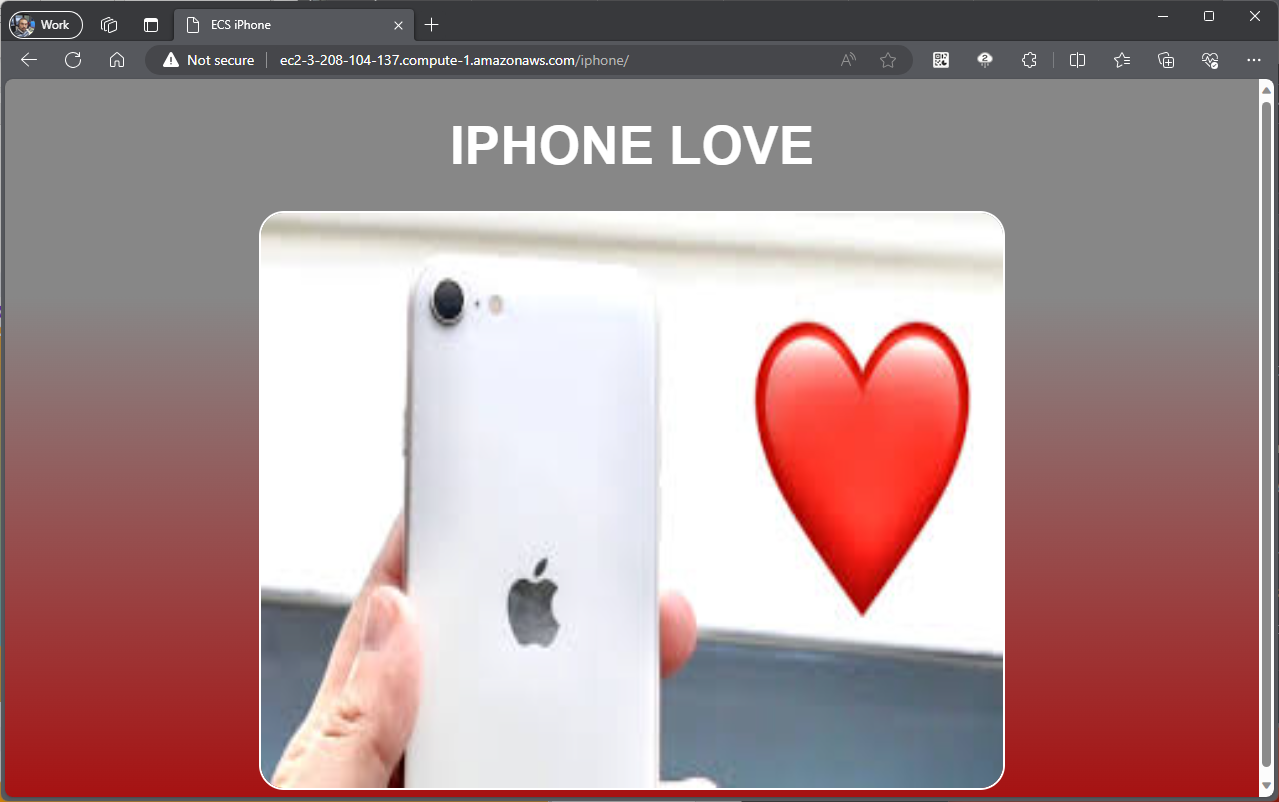
<http://ec2-3-208-104-137.compute-1.amazonaws.com>



### Select Android branch

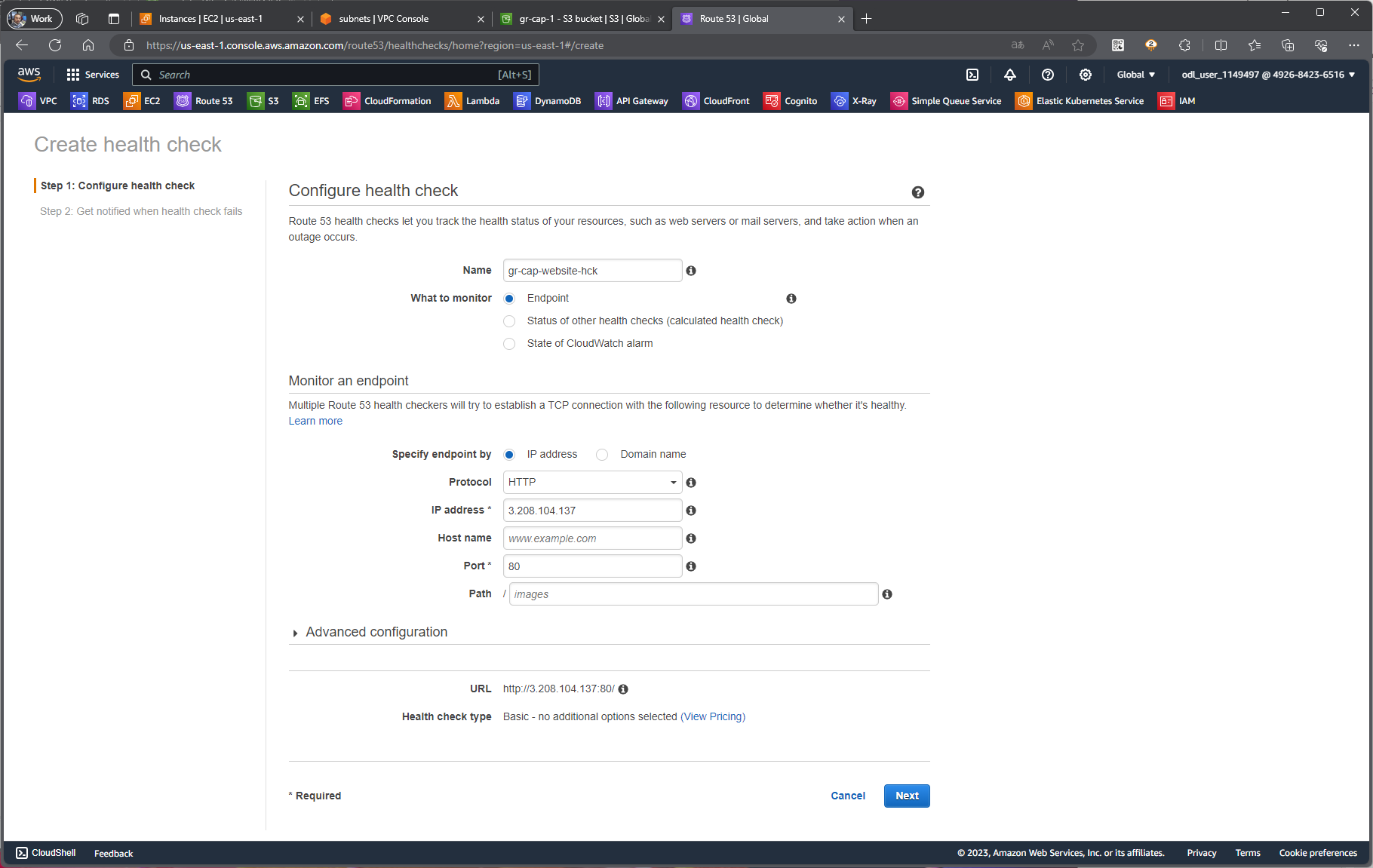


### Select Apple branch

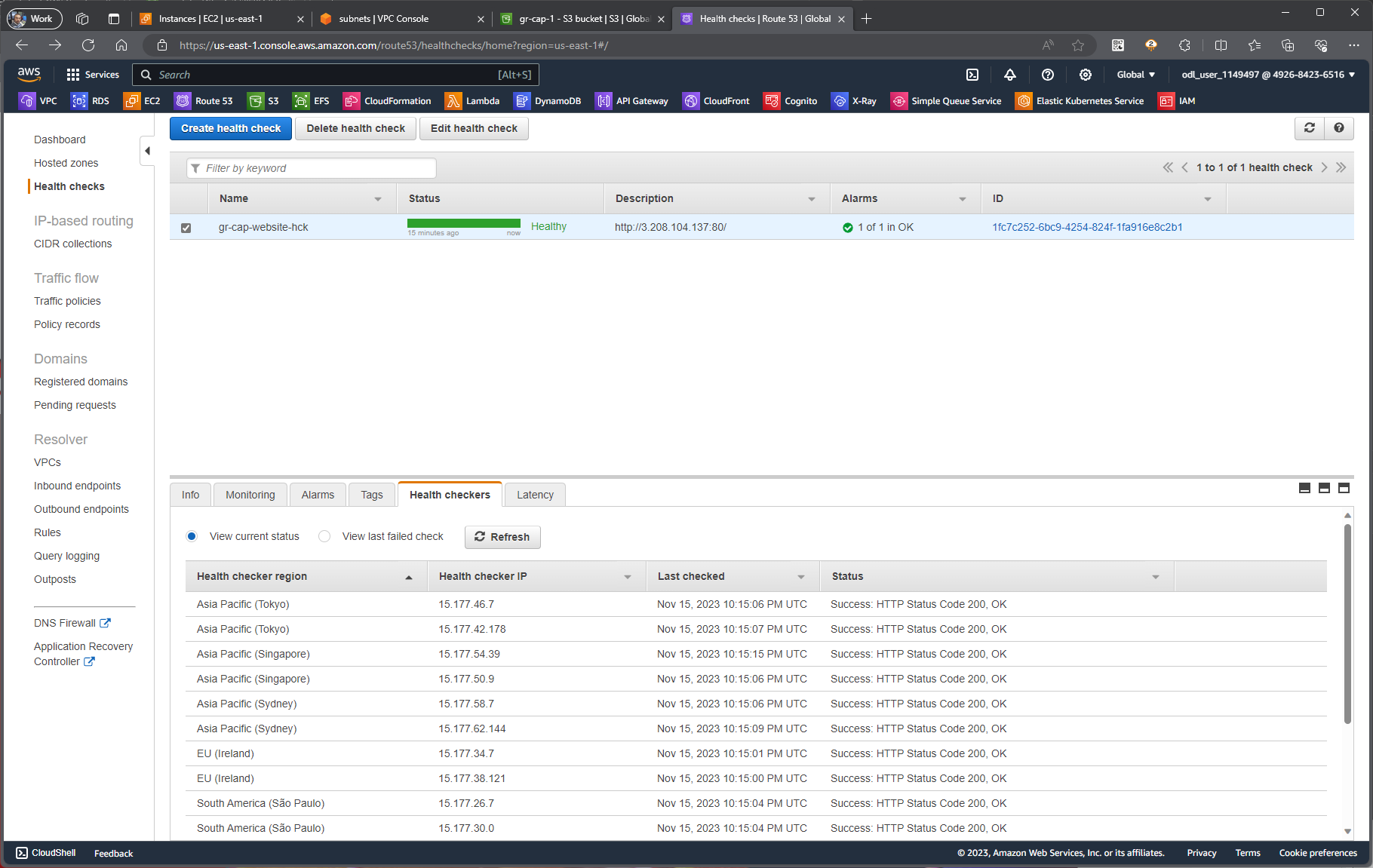


## Website Heath check

Define Heath check with Route53:







## View EC2 statics with CloudWatch

