Case Study 7: Kubernetes Cluster Management with Terraform

- **Concepts Used: Kubernetes, Terraform, AWS Cloud9.**
- **Problem Statement: "Use Terraform to provision a Kubernetes cluster on AWS. Then, use AWS Cloud9 IDE to deploy a sample application on the cluster using kubectl."**
- **Tasks:**
    - **Write a Terraform script to create a Kubernetes cluster on AWS.**
    - **Use AWS Cloud9 to configure kubectl for the newly created cluster.**
    - **Deploy a simple application (e.g., a Python Flask app) on the Kubernetes cluster and verify its deployment.**

Introduction:

Overview:

The following case study uses a number of services that would help the user to deploy the application onto Kubernetes using AWS Services such as EKS (Elastic Kubernetes Service), VPC (Virtual Private Cloud), etc. For creating this, we will be taking the help of Terraform (Infrastructure as Code). This is used to define the infrastructure of the required cluster and which in turn can be used by many users. This cluster will be then used to add and deploy a flask application.

Key Features:

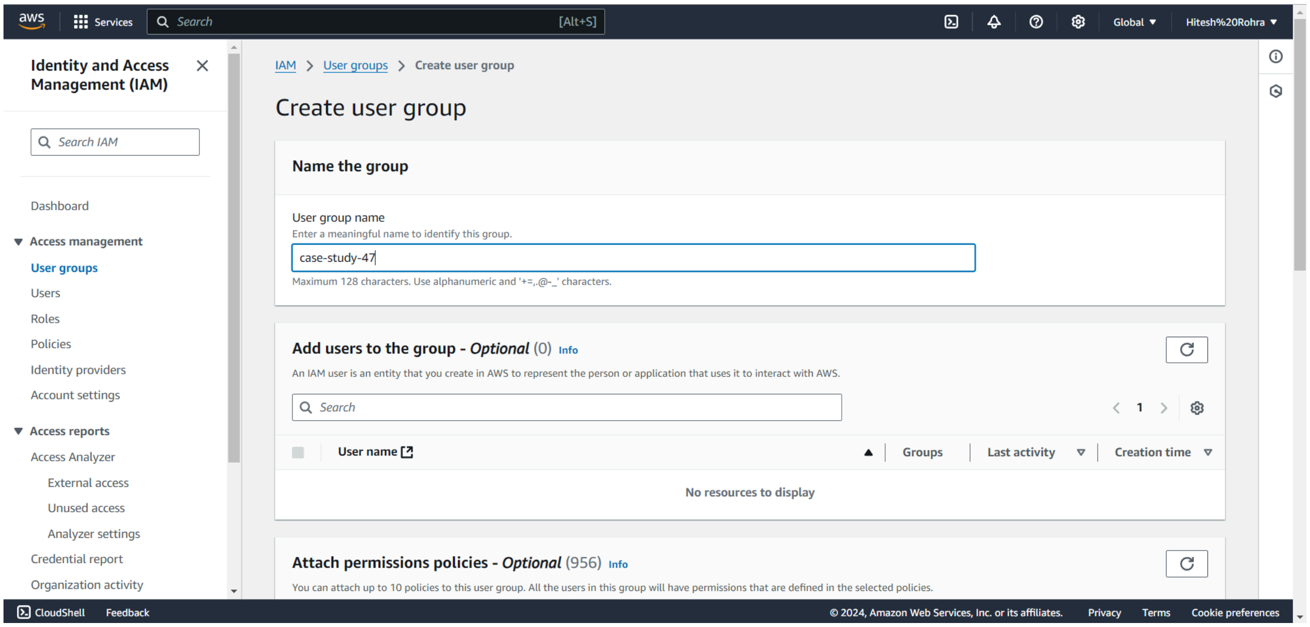
1) Use of Terraform to provide Infrastructure of EKS to be formed on AWS.
2) Using Docker to containerize the application.
3) Use of kubectl to manage cluster from terminal.
4) LoadBalancing to expose the application from container to local system.

Application:

The same case study could be used to deploy a larger scale flask project that could be used in various web applications. Not only that, other than flask, the projects working on different tech stacks could also be used in place of flask. This would give the main advantage that kubernetes allows to maintain Scalability, Reliability and Efficieny.

Step by Step Explanation:

Step 1:
Create a IAM user group and add the following inbuilt policies and a few inline policies.



Create user group

Name the group

User group name
Enter a meaningful name to identify this group.
case-study-47
Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

Add users to the group - *Optional* (0) Info
An IAM user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS.

User name	Groups	Last activity	Creation time
No resources to display			

Attach permissions policies - *Optional* (956) Info
You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

AWS Managed Policies:

1) AmazonEKSClusterPolicy
2) AmazonEKSWorkerNodePolicy
3) AmazonVPCFullAccess
4) IAMFullAccess

Customer Managed Policies:

1) DescribeInstance JSON:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": [
                                "ec2:DescribeInstanceCreditSpecifications",
                                "ec2:DescribeInstances",
                                "ec2:DescribeInstanceAttribute"
                        ],
                        "Resource": "*"
                }
        ]
}
```

2) EKSNodeGroup

JSON:

```
{                                                   "Version": "2012-10-17",
                                                    "Statement": [
                                                            {
                        "Effect": "Allow",
                        "Action": [
                                "eks:CreateNodegroup",
                                "eks:DescribeNodegroup",
                                "eks:DeleteNodegroup",
                                "eks:CreateCluster",
                                "eks:DescribeCluster",
                                "eks:DeleteCluster",
                                "iam:PassRole"
                        ],
                        "Resource": "*"
                }
        ]
}
```

3) EKSFullAccess JSON

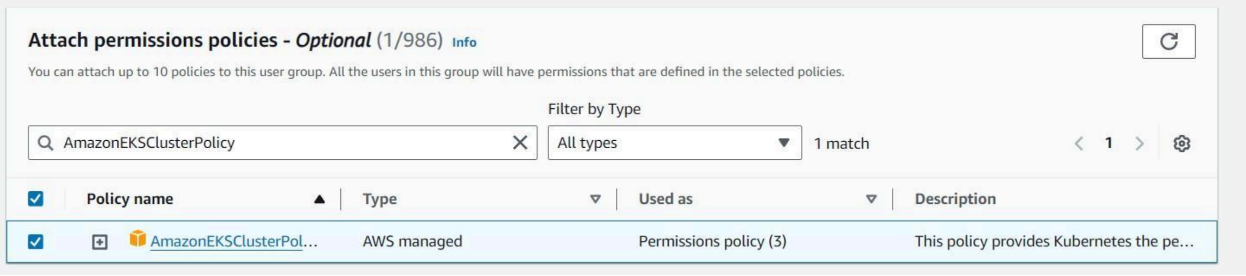
```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": [
                                "eks:CreateCluster",
                                "eks:DescribeCluster",
                                "eks:ListClusters",
                                "eks:DeleteCluster",
                                "ec2:DescribeSubnets",
                                "ec2:DescribeVpcs",
                                "iam:CreateServiceLinkedRole",
                                "iam:PassRole"
                        ],
                        "Resource": "*"
                }
        ]
}
```

4) InstancePolicy JSON:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": [
                                "ec2:RunInstances",
                                "ec2:DescribeInstances",
                                "ec2:TerminateInstances",
                                "ec2:CreateTags",
                                "ec2:DescribeKeyPairs"
                        ],
                        "Resource": "*"
                }
        ]
}
```

To add the AWS Managed Policies:

1) On the create User Group menu, give a name to your user group, then scroll down to Attach Permission Policies. Search for the listed policies and click on the checkbox next to the policy.



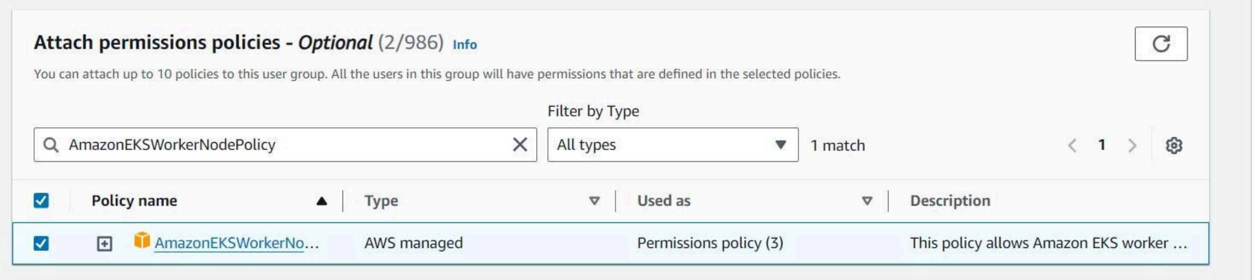
Attach permissions policies - *Optional* (1/986) Info

You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

Filter by Type

AmazonEKSClusterPolicy | All types | 1 match

☑	Policy name	Type	Used as	Description
☑	AmazonEKSClusterPol...	AWS managed	Permissions policy (3)	This policy provides Kubernetes the pe...



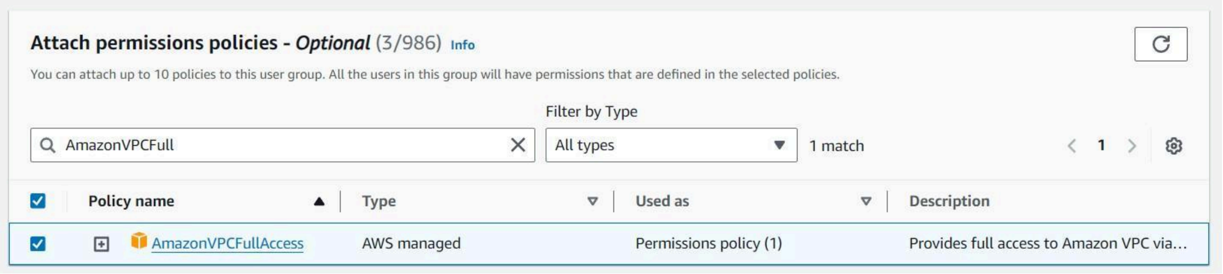
Attach permissions policies - *Optional* (2/986) Info

You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

Filter by Type

AmazonEKSWorkerNodePolicy | All types | 1 match

☑	Policy name	Type	Used as	Description
☑	AmazonEKSWorkerNo...	AWS managed	Permissions policy (3)	This policy allows Amazon EKS worker ...



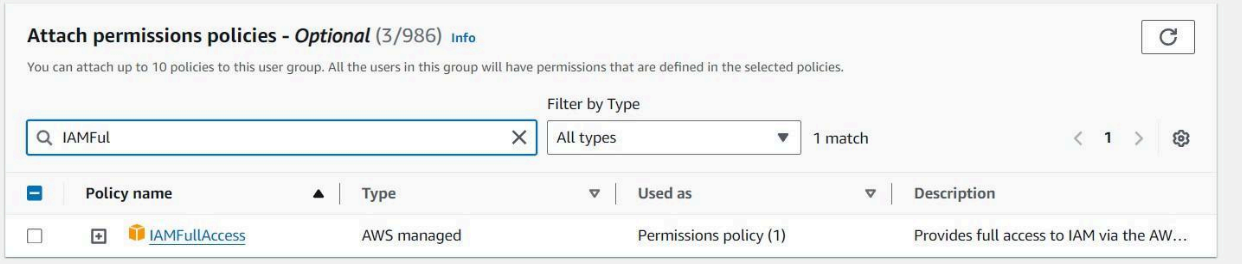
Attach permissions policies - *Optional* (3/986) Info

You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

Filter by Type

AmazonVPCFull | All types | 1 match

☑	Policy name	Type	Used as	Description
☑	AmazonVPCFullAccess	AWS managed	Permissions policy (1)	Provides full access to Amazon VPC via...



Attach permissions policies - *Optional* (3/986) Info

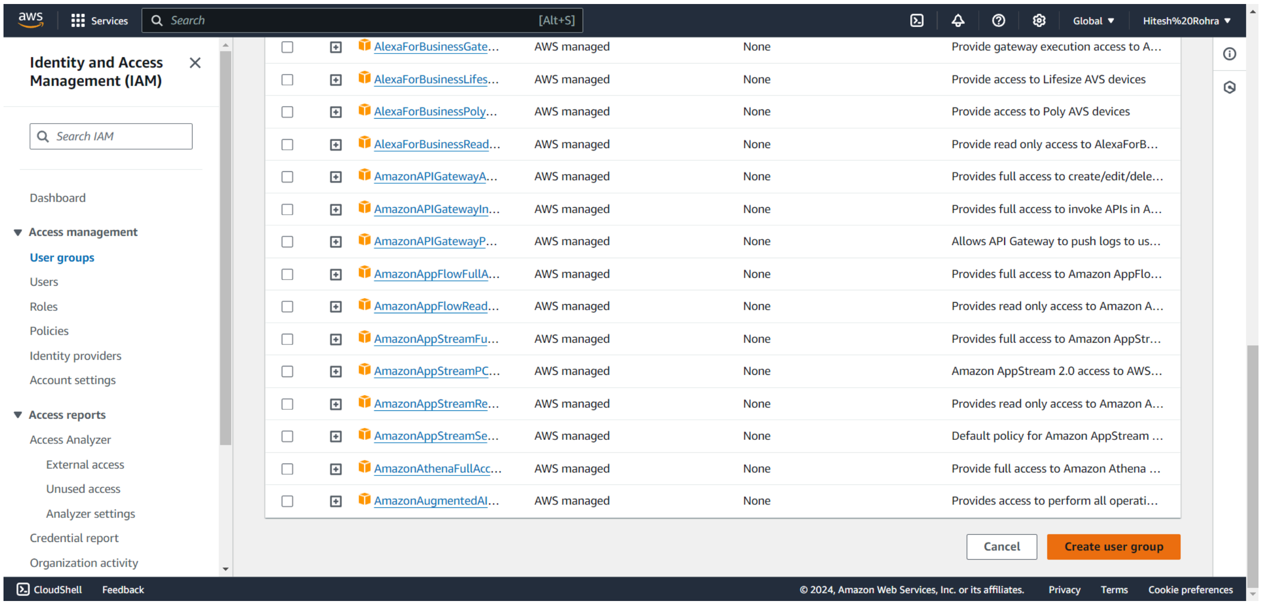
You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.

Filter by Type: All types

Search: IAMFul — 1 match

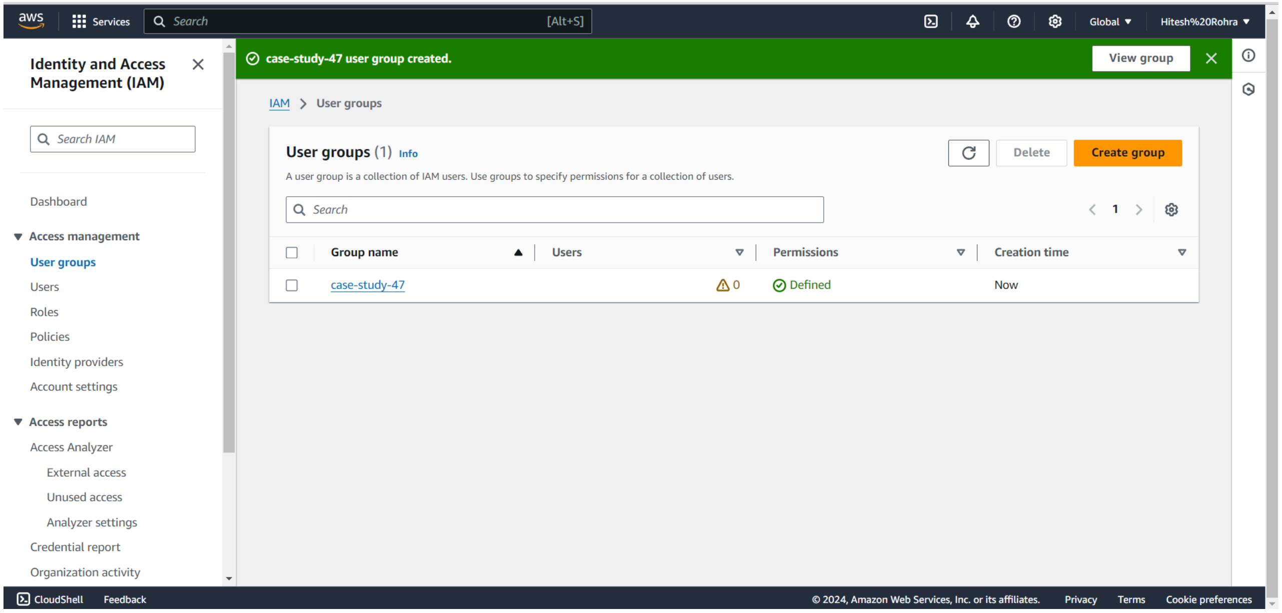
	Policy name	Type	Used as	Description
□	IAMFullAccess	AWS managed	Permissions policy (1)	Provides full access to IAM via the AW…

2) Now click on Create Group.



	Policy name	Type	Used as	Description
□	AlexaForBusinessGate…	AWS managed	None	Provide gateway execution access to A…
□	AlexaForBusinessLifes…	AWS managed	None	Provide access to Lifesize AVS devices
□	AlexaForBusinessPoly…	AWS managed	None	Provide access to Poly AVS devices
□	AlexaForBusinessRead…	AWS managed	None	Provide read only access to AlexaForB…
□	AmazonAPIGatewayA…	AWS managed	None	Provides full access to create/edit/dele…
□	AmazonAPIGatewayIn…	AWS managed	None	Provides full access to invoke APIs in A…
□	AmazonAPIGatewayP…	AWS managed	None	Allows API Gateway to push logs to us…
□	AmazonAppFlowFullA…	AWS managed	None	Provides full access to Amazon AppFlo…
□	AmazonAppFlowRead…	AWS managed	None	Provides read only access to Amazon A…
□	AmazonAppStreamFu…	AWS managed	None	Provides full access to Amazon AppStr…
□	AmazonAppStreamPC…	AWS managed	None	Amazon AppStream 2.0 access to AWS…
□	AmazonAppStreamRe…	AWS managed	None	Provides read only access to Amazon A…
□	AmazonAppStreamSe…	AWS managed	None	Default policy for Amazon AppStream …
□	AmazonAthenaFullAcc…	AWS managed	None	Provide full access to Amazon Athena …
□	AmazonAugmentedAI…	AWS managed	None	Provides access to perform all operati…

3) Here, you can see your user group created.



case-study-47 user group created.

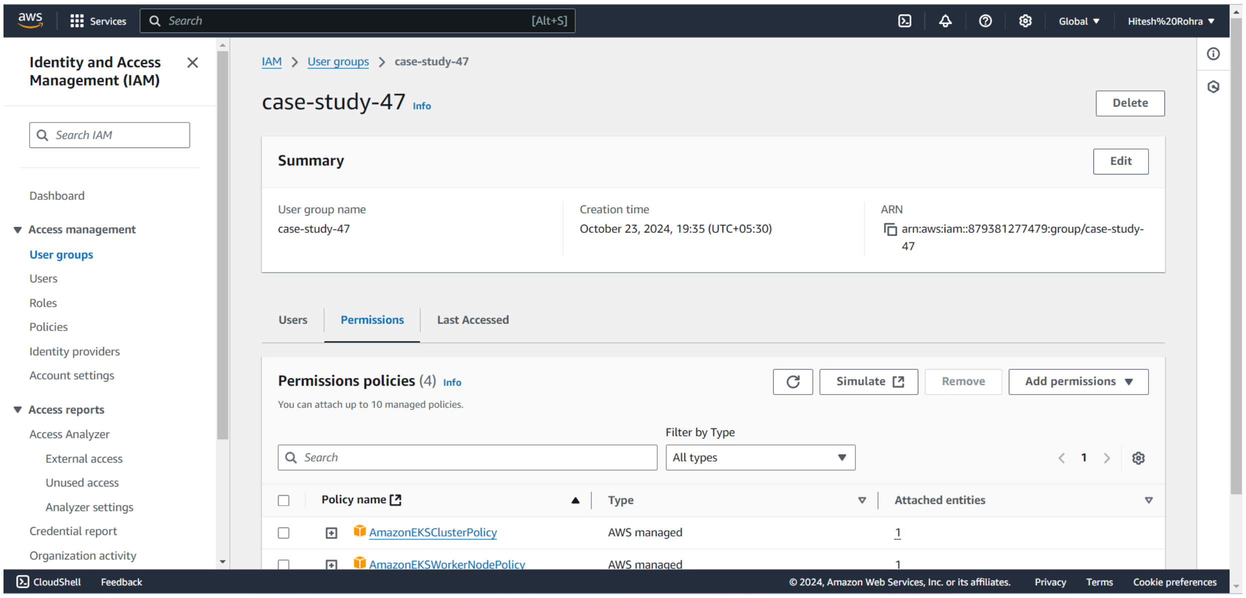
User groups (1) Info

A user group is a collection of IAM users. Use groups to specify permissions for a collection of users.

	Group name	Users	Permissions	Creation time
□	case-study-47	⚠ 0	✓ Defined	Now

To add the Customer Managed Policies, follow the following steps:

1) Click on the name of the group. Then, navgate to the Permissions tab.



IAM > User groups > case-study-47

case-study-47 Info

Summary

User group name: case-study-47

Creation time: October 23, 2024, 19:35 (UTC+05:30)

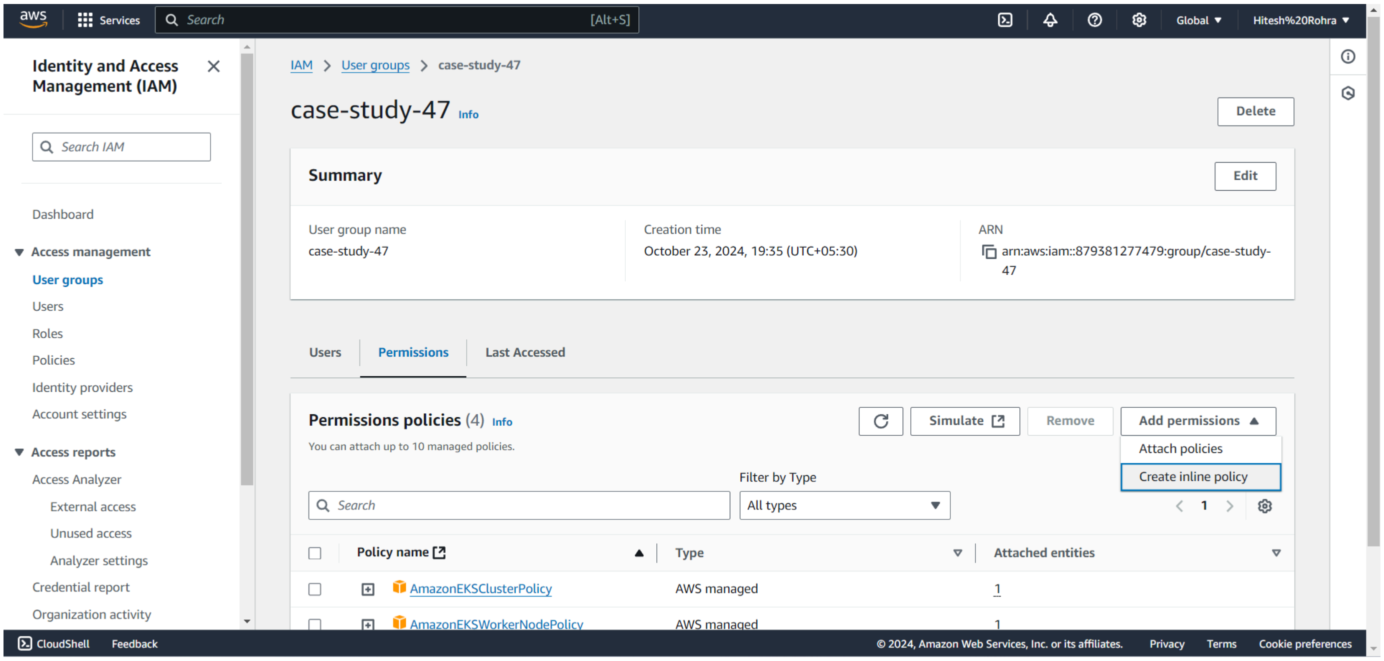
ARN: arn:aws:iam::879381277479:group/case-study-47

Permissions policies (4) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonEKSClusterPolicy	AWS managed	1
AmazonEKSWorkerNodePolicy	AWS managed	1

2) Here, click on the Add Permissions dropdown and click on Create Inline Policy



IAM > User groups > case-study-47

case-study-47 Info

Summary

User group name: case-study-47

Creation time: October 23, 2024, 19:35 (UTC+05:30)

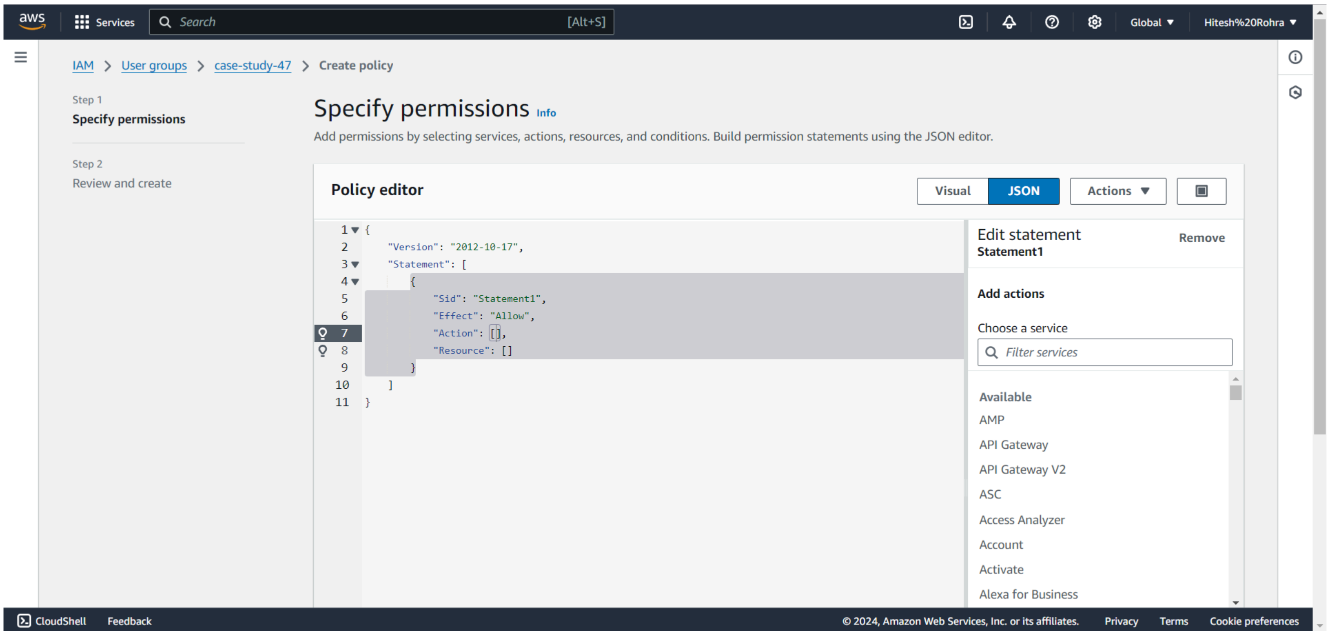
ARN: arn:aws:iam::879381277479:group/case-study-47

Permissions policies (4) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonEKSClusterPolicy	AWS managed	1
AmazonEKSWorkerNodePolicy	AWS managed	1

3) Switch to the JSON editor and replace the old code with the codes provided above.



Specify permissions

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [],
            "Resource": []
        }
    ]
}
```

Edit statement
Statement1

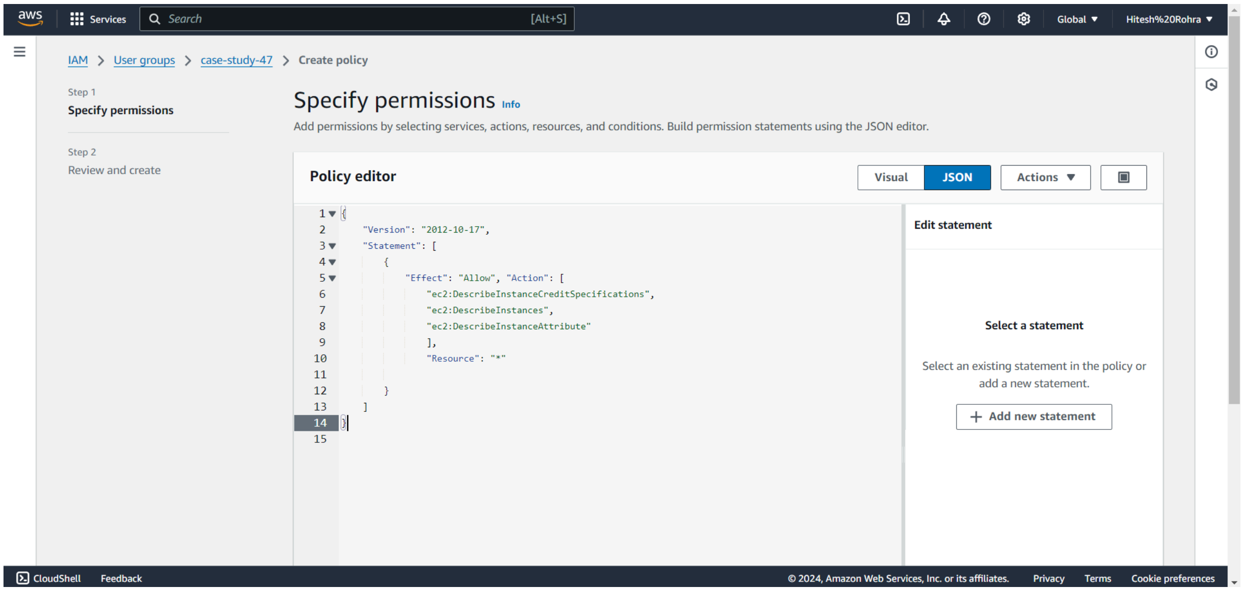
Add actions

Choose a service

Filter services

© 2024, Amazon Web Services, Inc. or its affiliates.

4) After pasting our code, click on Next.



Specify permissions

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

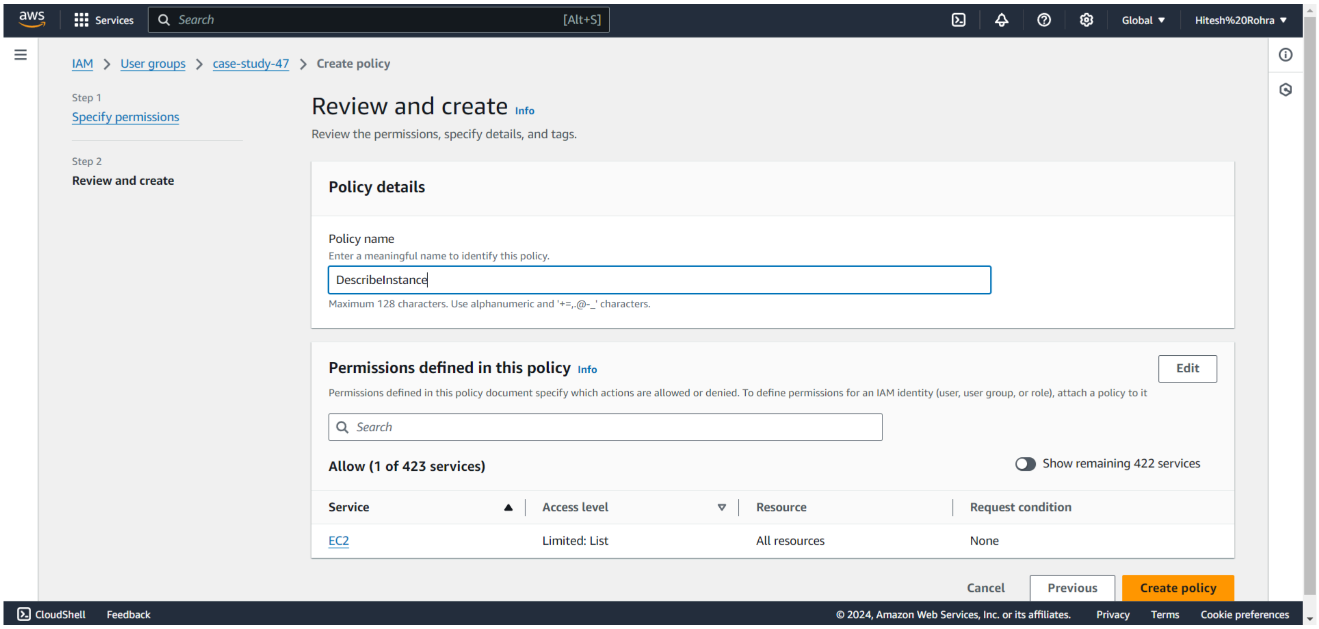
Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow", "Action": [
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceAttribute"
                ],
                "Resource": "*"

        }
    ]
}

```

Select a statement

Select an existing statement in the policy or add a new statement.

5) Give a name to your policy and click on Create Policy.



Review and create

Review the permissions, specify details, and tags.

Policy details

Policy name

Enter a meaningful name to identify this policy.

DescribeInstance

Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

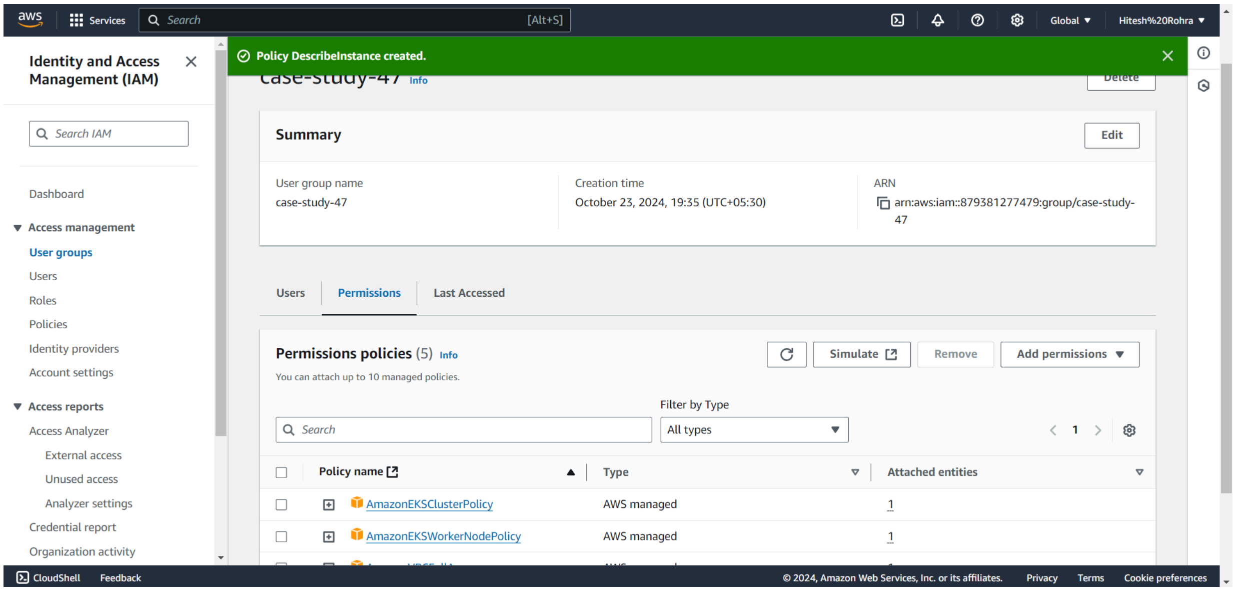
Permissions defined in this policy

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (1 of 423 services)

Service ▲	Access level ▽	Resource	Request condition
EC2	Limited: List	All resources	None

6) This adds the DescribeInstance policy as a permission. Repeat the same steps for the remaining 3 Customer Managed Policies.



Policy DescribeInstance created.

case-study-47 Info

Summary

User group name: case-study-47

Creation time: October 23, 2024, 19:35 (UTC+05:30)

ARN: arn:aws:iam::879381277479:group/case-study-47

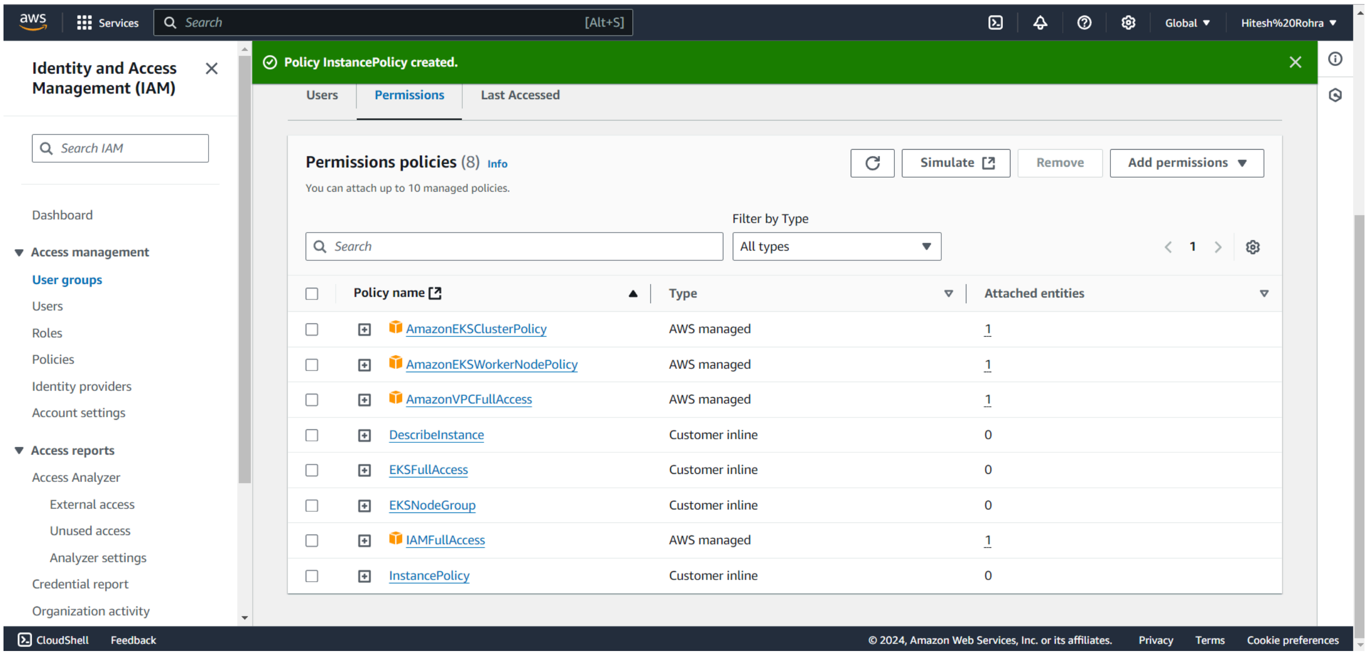
Permissions policies (5) Info

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonEKSClusterPolicy	AWS managed	1
AmazonEKSWorkerNodePolicy	AWS managed	1

All the policies are added to the user group.



Policy InstancePolicy created.

Permissions policies (8) Info

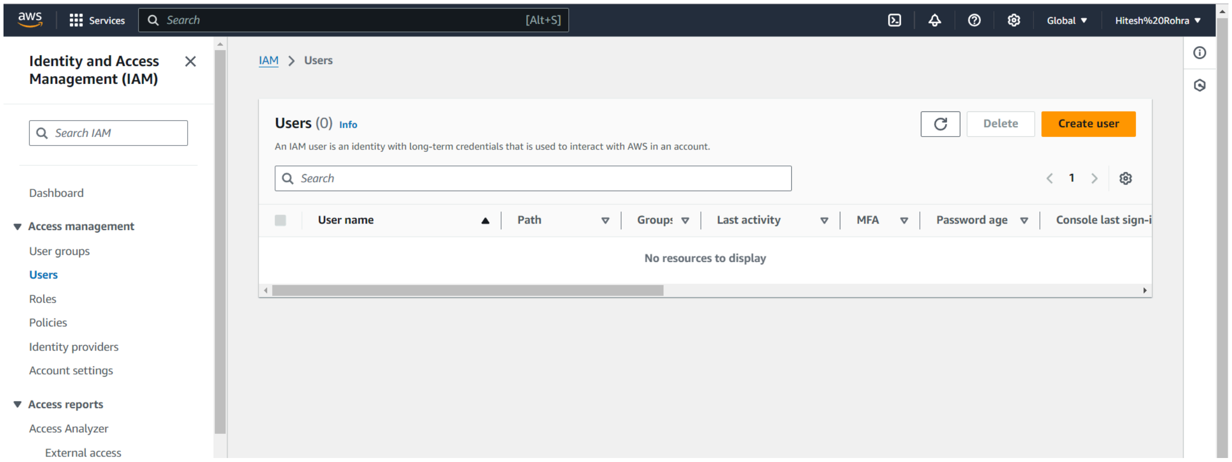
You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AmazonEKSClusterPolicy	AWS managed	1
AmazonEKSWorkerNodePolicy	AWS managed	1
AmazonVPCFullAccess	AWS managed	1
DescribeInstance	Customer inline	0
EKSFullAccess	Customer inline	0
EKSNodeGroup	Customer inline	0
IAMFullAccess	AWS managed	1
InstancePolicy	Customer inline	0

Step 2:
Add a user to this user group

1. Go to users form the left navigation pane.

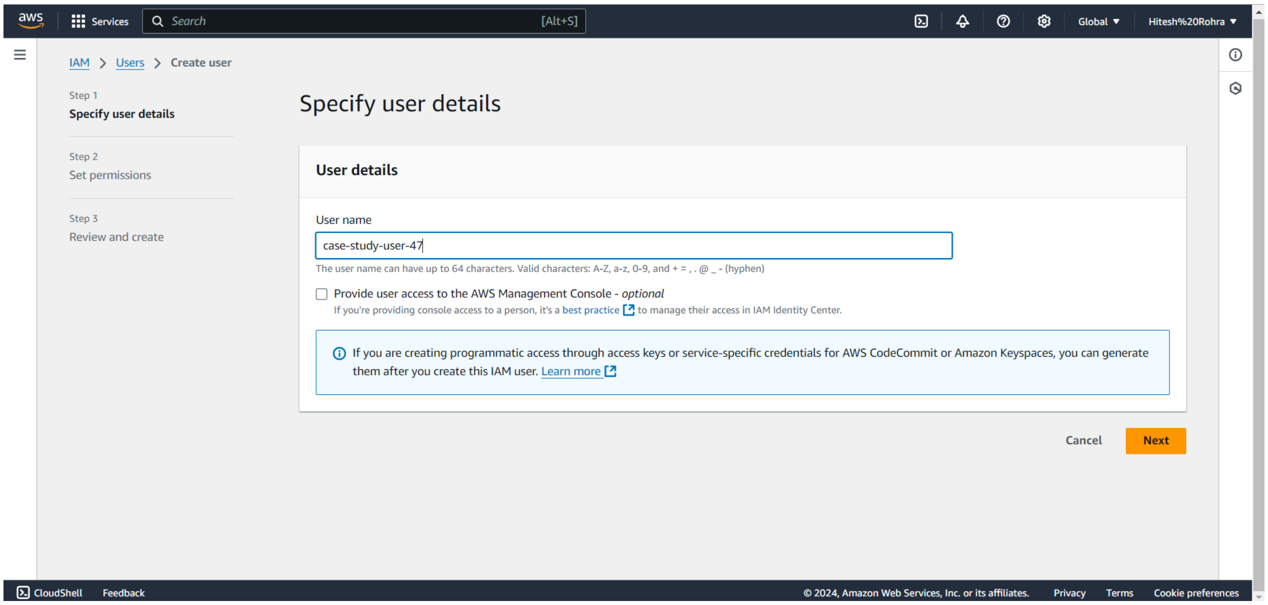


Users (0) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

No resources to display

2. Click on Create User. Give a name to your user and click Next.



Specify user details

User details

User name

case-study-user-47

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

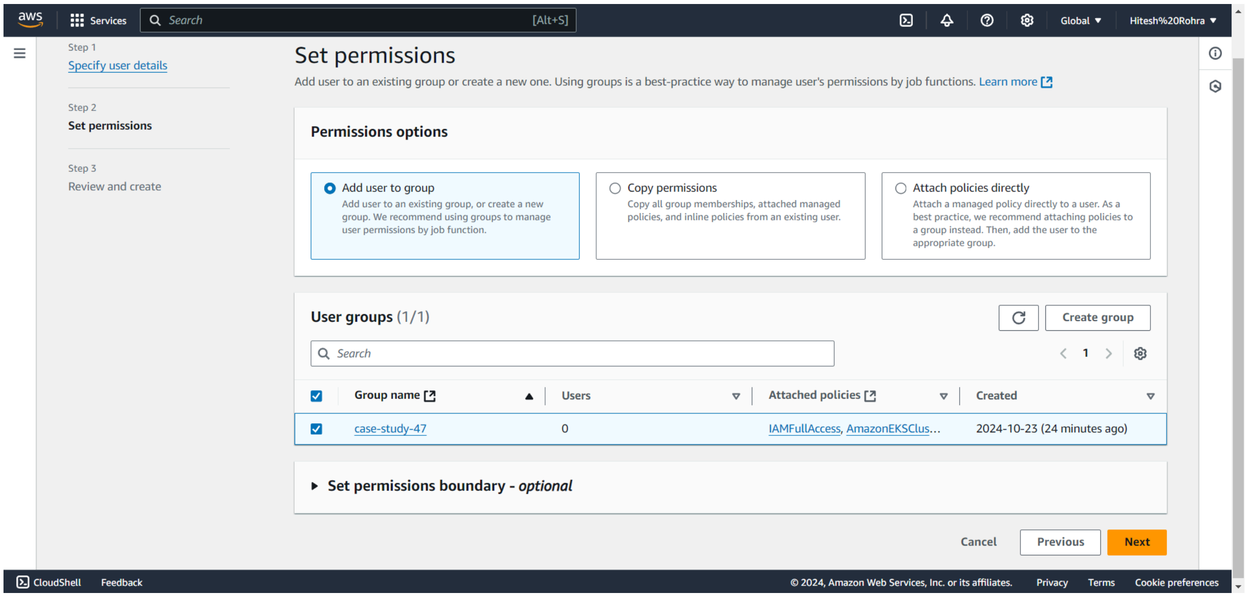
□ Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

© 2024, Amazon Web Services, Inc. or its affiliates.

3. To set the permissions, keep the option on Add User to Group and select the group we just created. Then click on Next.



Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

● Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

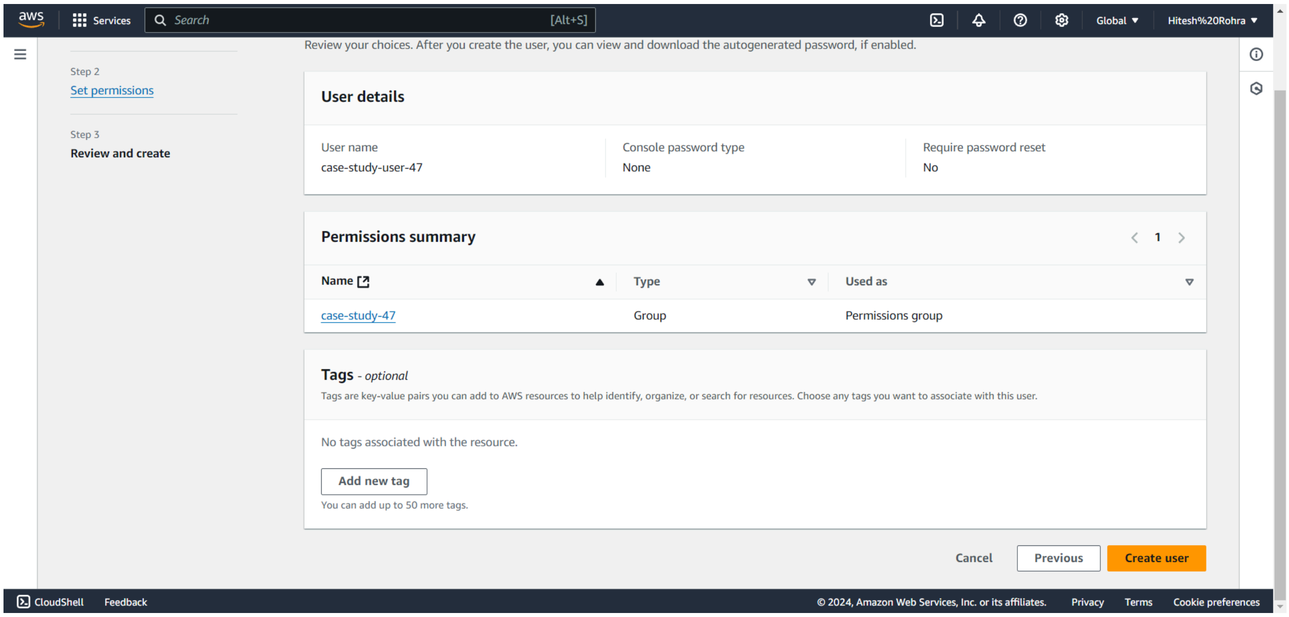
○ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

User groups (1/1)

☑	Group name	Users	Attached policies	Created
☑	case-study-47	0	IAMFullAccess, AmazonEKSClus...	2024-10-23 (24 minutes ago)

▸ Set permissions boundary - *optional*

4. Click on Create User.



Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

Step 2
Set permissions
Step 3
Review and create

User details

User name	Console password type	Require password reset
case-study-user-47	None	No

Permissions summary

Name	Type	Used as
case-study-47	Group	Permissions group

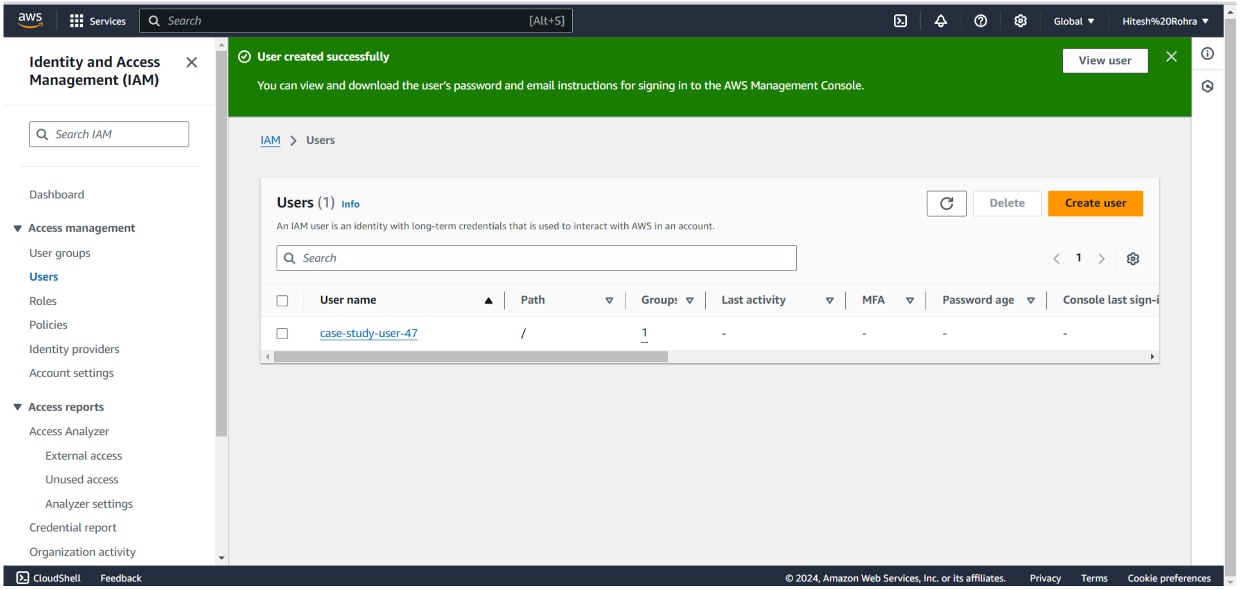
Tags - *optional*

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

You can add up to 50 more tags.

5. The user has been created successfully.



User created successfully

You can view and download the user's password and email instructions for signing in to the AWS Management Console.

Users (1)

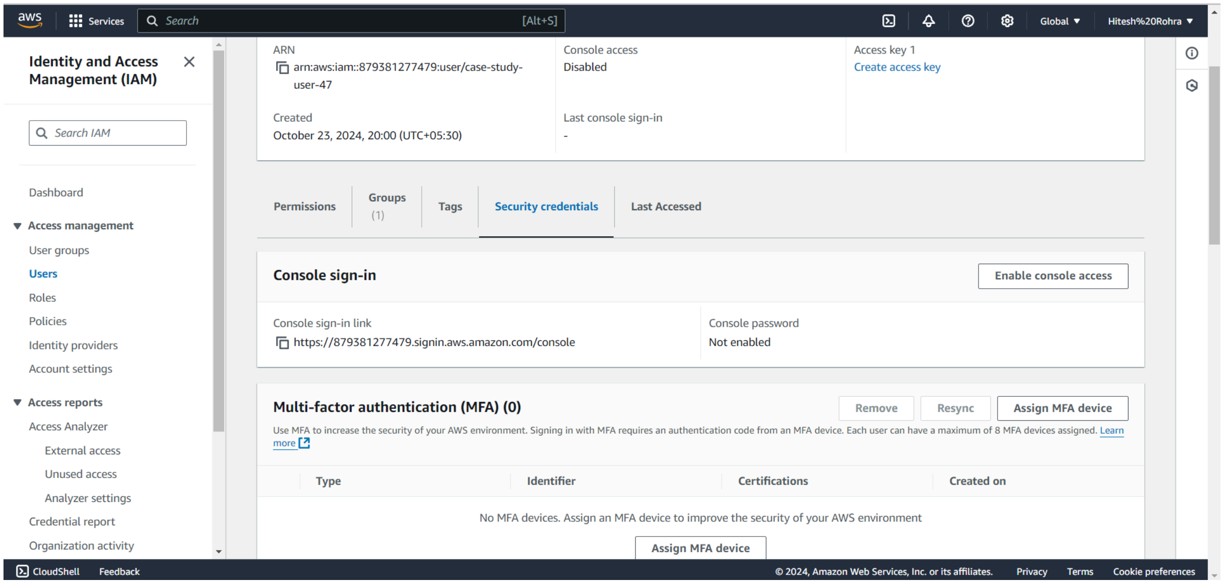
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

	User name	Path	Groups	Last activity	MFA	Password age	Console last sign-i
□	case-study-user-47	/	1	-	-	-	-

© 2024, Amazon Web Services, Inc. or its affiliates.

Step 3:
Generate access key (Normal and Secret) for the user.

1) Click on the name of the user just created. Here, go to Security Credentials.



ARN
arn:aws:iam::879381277479:user/case-study-user-47

Console access
Disabled

Access key 1
Create access key

Created
October 23, 2024, 20:00 (UTC+05:30)

Last console sign-in
-

Console sign-in

Console sign-in link
https://879381277479.signin.aws.amazon.com/console

Console password
Not enabled

Multi-factor authentication (MFA) (0)

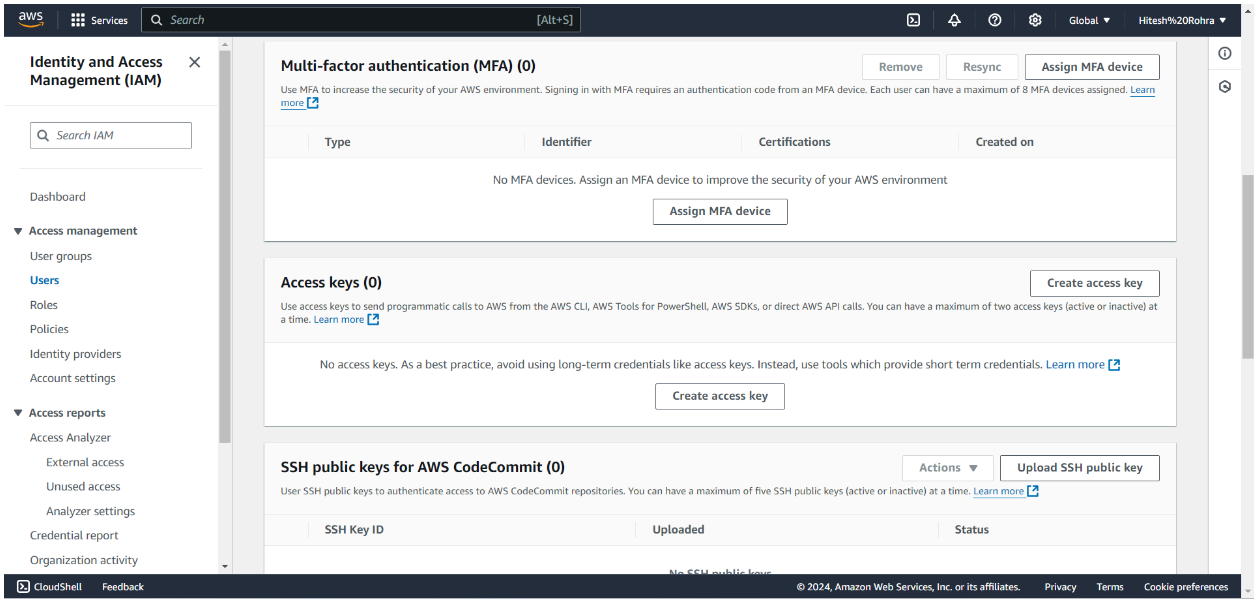
Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Type	Identifier	Certifications	Created on

No MFA devices. Assign an MFA device to improve the security of your AWS environment

© 2024, Amazon Web Services, Inc. or its affiliates.

2) Scroll down to the Access Keys and click on select Access Key.



Multi-factor authentication (MFA) (0)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Type	Identifier	Certifications	Created on

No MFA devices. Assign an MFA device to improve the security of your AWS environment

Assign MFA device

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. Learn more

Create access key

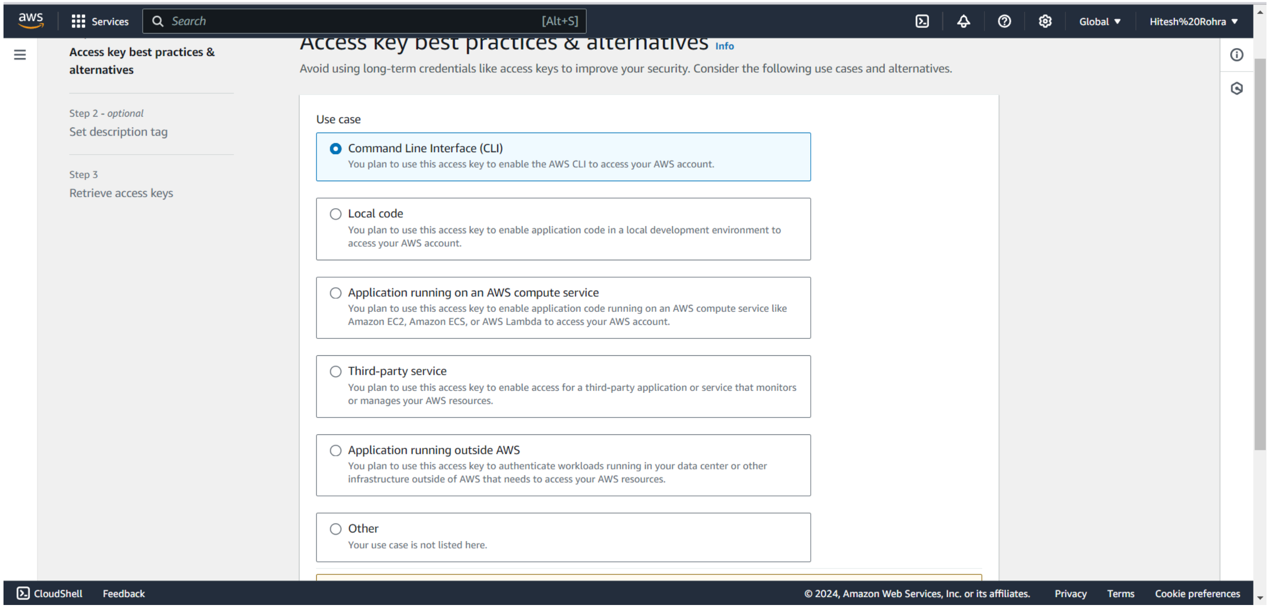
SSH public keys for AWS CodeCommit (0)

User SSH public keys to authenticate access to AWS CodeCommit repositories. You can have a maximum of five SSH public keys (active or inactive) at a time. Learn more

SSH Key ID	Uploaded	Status

No SSH public keys

3) Select Command Line Interface (CLI) as use case, check the confirmation and click on next.



Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

● Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

○ Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

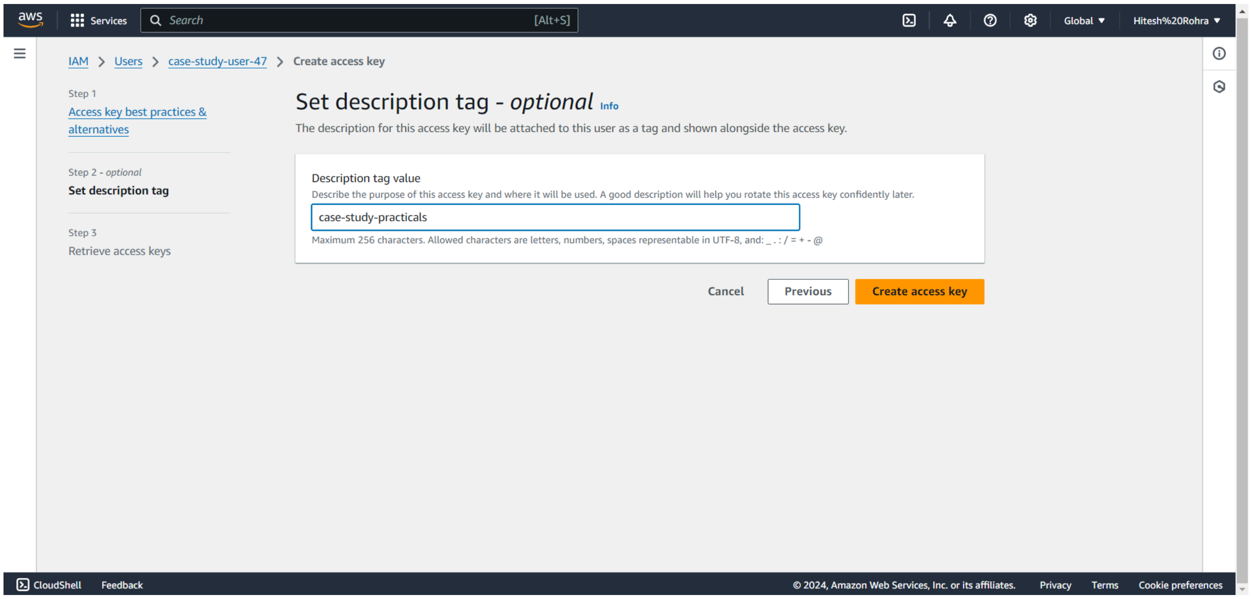
○ Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

○ Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

○ Application running outside AWS
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

○ Other
Your use case is not listed here.

4) Give a description to your access key.



Set description tag - *optional* Info

The description for this access key will be attached to this user as a tag and shown alongside the access key.

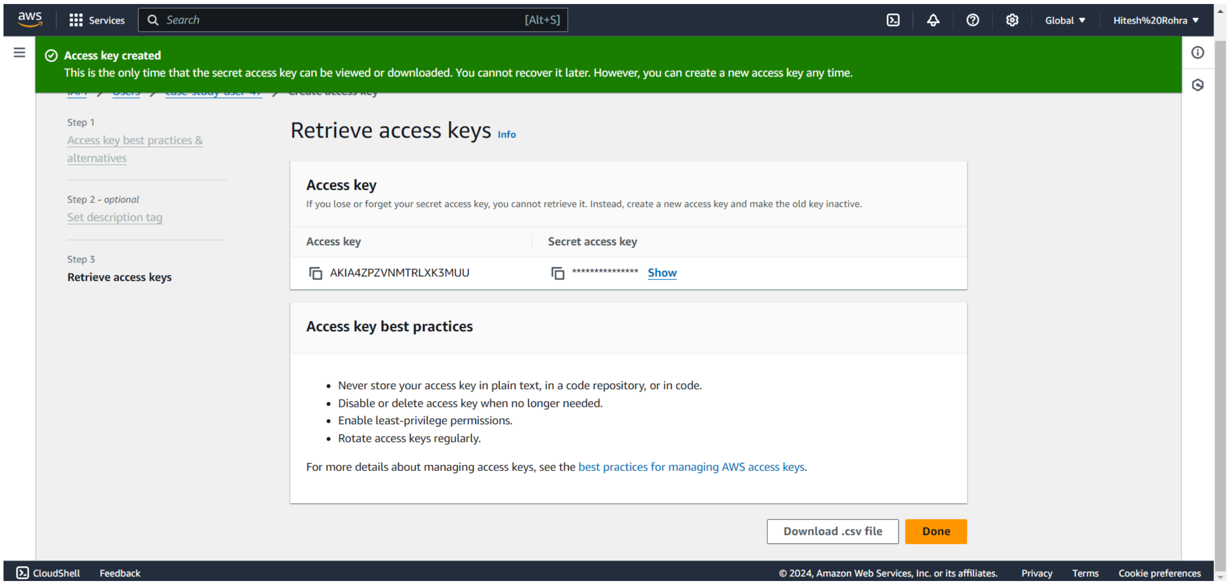
Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

case-study-practicals

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @

5) This has created you access key pair. Now, download it as a .csv file to use it later. Then click on done.



Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIA4ZPZVNMTRLXK3MUU	*************** Show

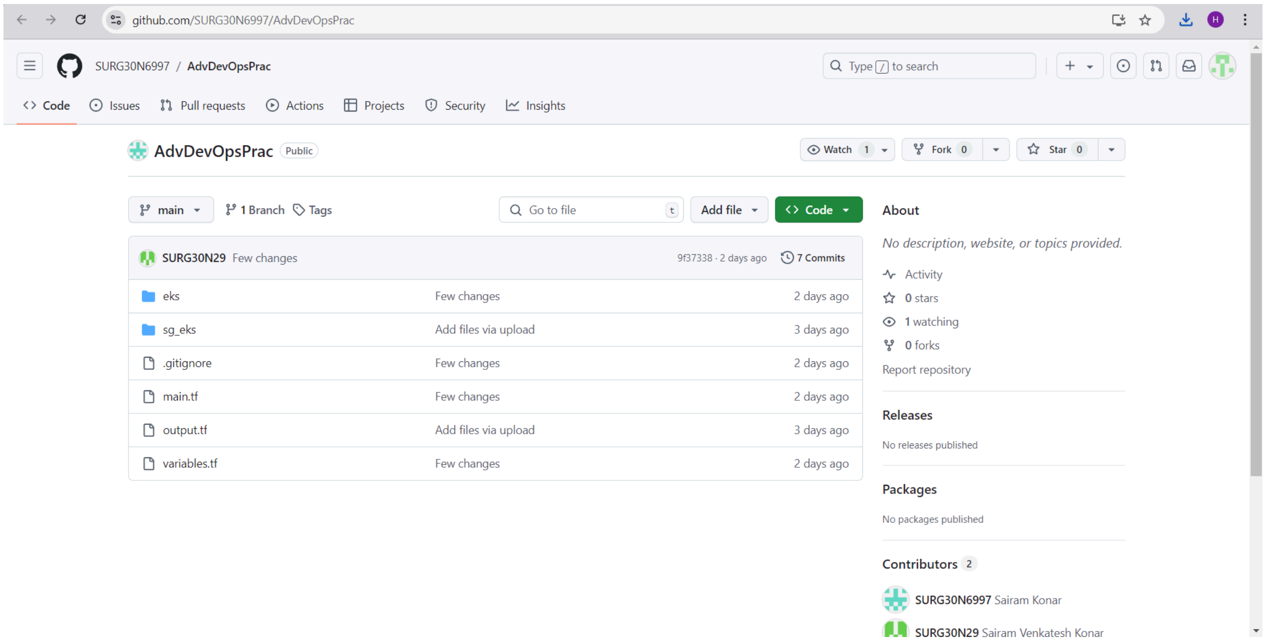
Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the best practices for managing AWS access keys.

Step 4:
Set up a terraform script for creating a Kubernetes cluster on AWS (Elastic Kubernetes Cluster)

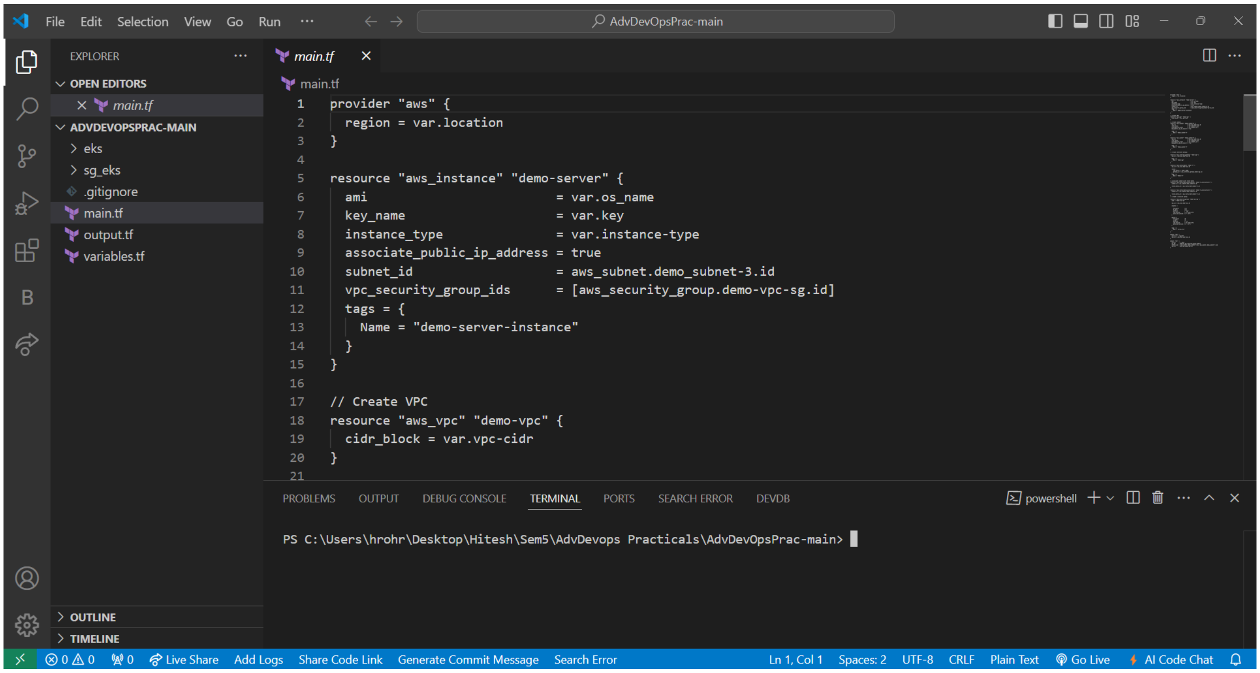
1) Visit the following github link. Here, you will find the required files for the scripts.



SURG30N29 Few changes 9f37338 · 2 days ago 7 Commits

eks	Few changes	2 days ago
sg_eks	Add files via upload	3 days ago
.gitignore	Few changes	2 days ago
main.tf	Few changes	2 days ago
output.tf	Add files via upload	3 days ago
variables.tf	Few changes	2 days ago

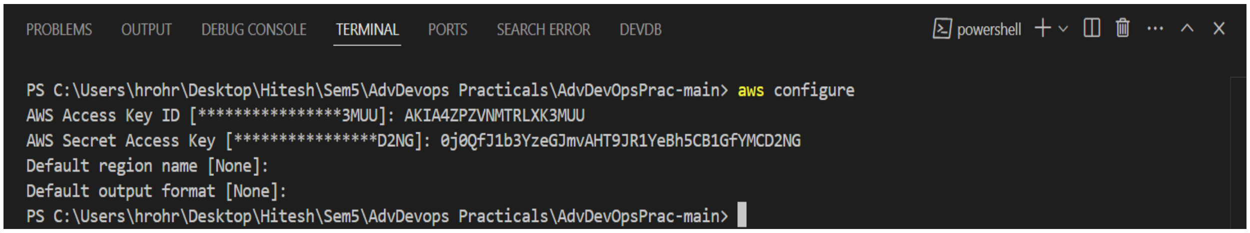
2) Clone the github.



```
provider "aws" {
  region = var.location
}

resource "aws_instance" "demo-server" {
  ami                         = var.os_name
  key_name                    = var.key
  instance_type               = var.instance-type
  associate_public_ip_address = true
  subnet_id                   = aws_subnet.demo_subnet-3.id
  vpc_security_group_ids      = [aws_security_group.demo-vpc-sg.id]
  tags = {
    Name = "demo-server-instance"
  }
}

// Create VPC
resource "aws_vpc" "demo-vpc" {
  cidr_block = var.vpc-cidr
}

```

```
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main>
```

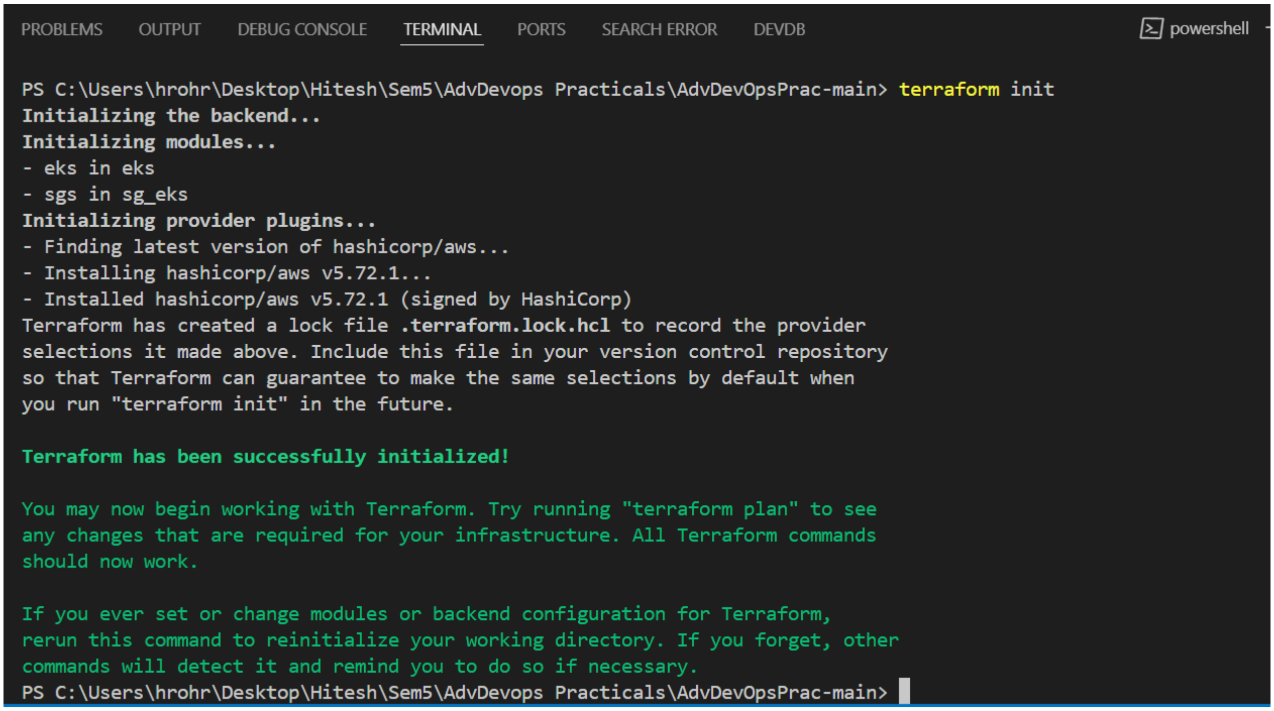
3) First, we will setup our IAM user on this script. For that, run the command
   aws configure



```
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main> aws configure
AWS Access Key ID [***************3MUU]: AKIA4ZPZVNMTRLXK3MUU
AWS Secret Access Key [***************D2NG]: 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
Default region name [None]:
Default output format [None]:
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main>
```

Here, paste the Access Key ID and Secret Access Key ID from the .csv file.
If for region name and output format, it shows default, give input as us-east-1 (or your region) and output as json.

4) Now the user is configured. We need to run the terraform scripts. For this, first run
   terraform init

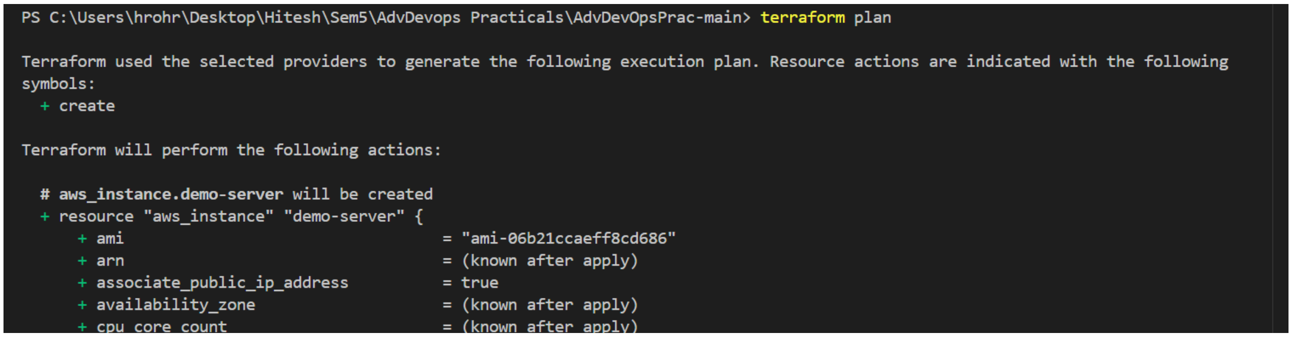


```
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main> terraform init
Initializing the backend...
Initializing modules...
- eks in eks
- sgs in sg_eks
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main>
```

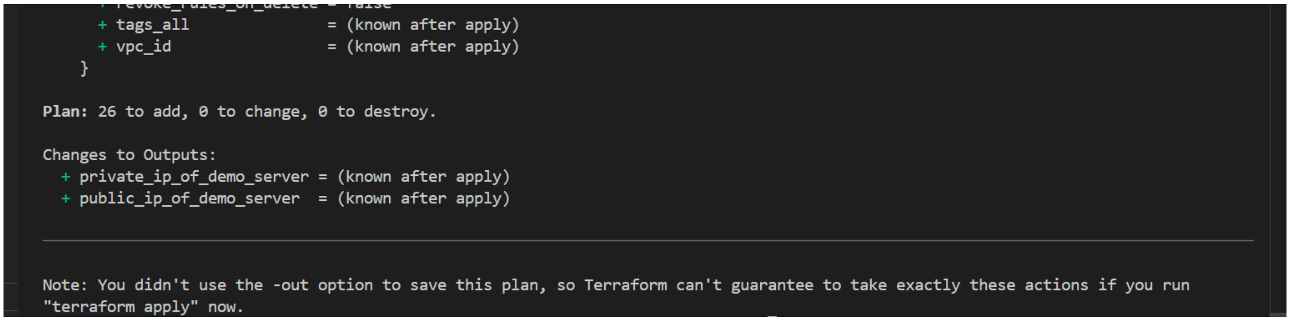
This command will create a .terraform and .terraform.lock.hcl

5) Now run command

terraform plan

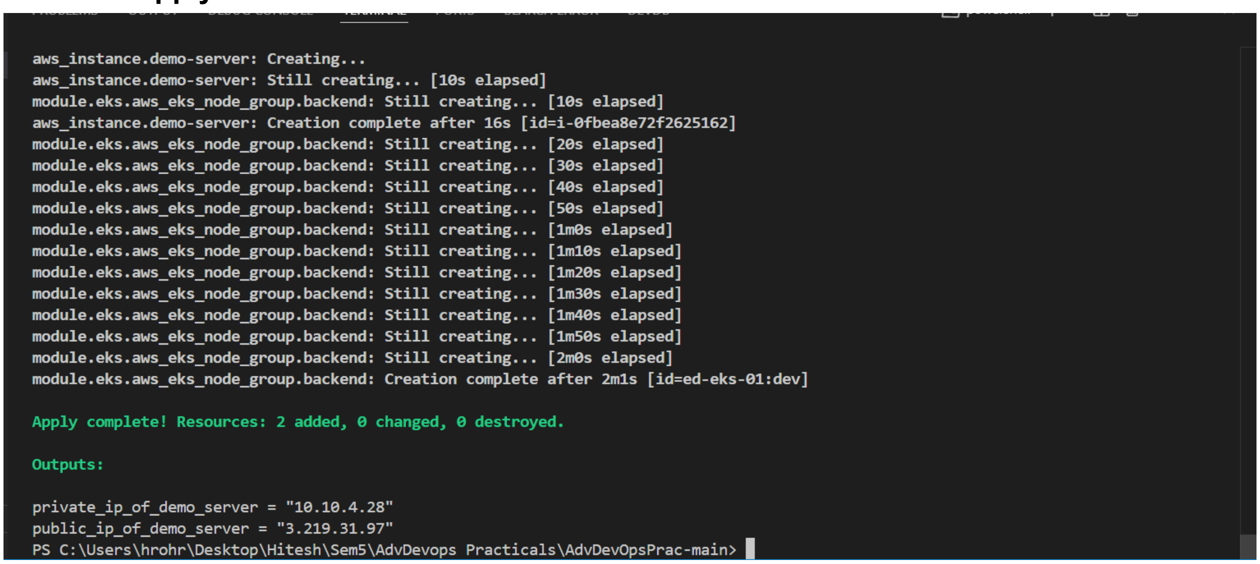


```
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.demo-server will be created
  + resource "aws_instance" "demo-server" {
      + ami                                  = "ami-06b21ccaeff8cd686"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = true
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```



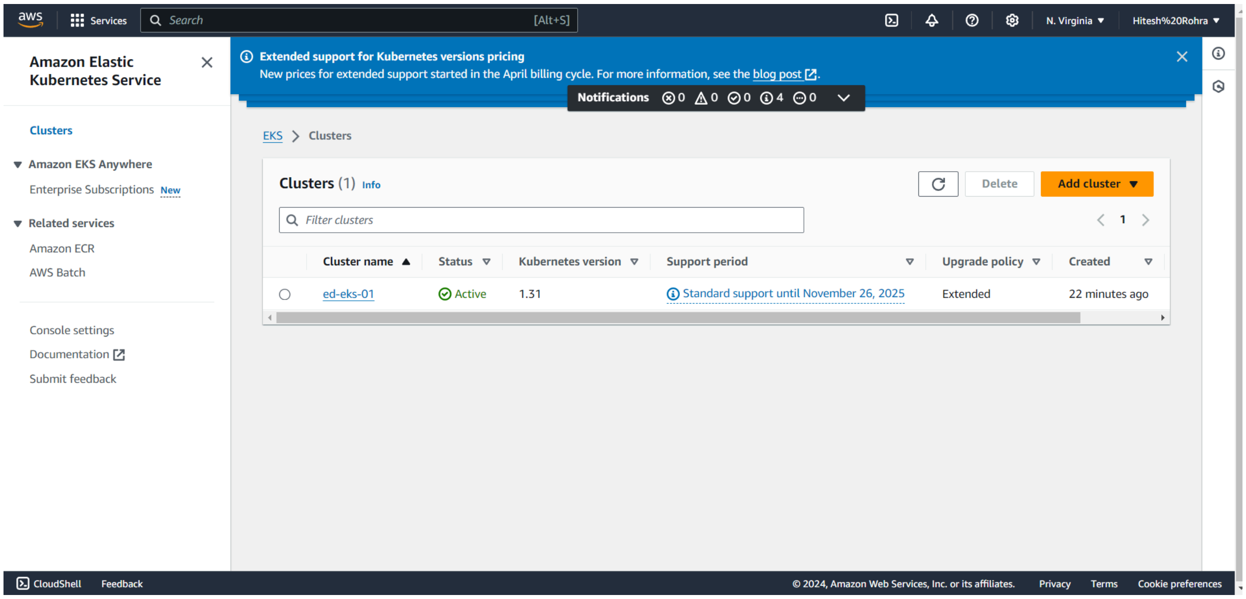
```
      + revoke_rules_on_delete = false
      + tags_all               = (known after apply)
      + vpc_id                 = (known after apply)
    }

Plan: 26 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + private_ip_of_demo_server = (known after apply)
  + public_ip_of_demo_server  = (known after apply)

─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run
"terraform apply" now.
```

6) Now, run command
terraform apply



```
aws_instance.demo-server: Creating...
aws_instance.demo-server: Still creating... [10s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [10s elapsed]
aws_instance.demo-server: Creation complete after 16s [id=i-0fbea8e72f2625162]
module.eks.aws_eks_node_group.backend: Still creating... [20s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [30s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [40s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [50s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m0s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m10s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m20s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m30s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m40s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [1m50s elapsed]
module.eks.aws_eks_node_group.backend: Still creating... [2m0s elapsed]
module.eks.aws_eks_node_group.backend: Creation complete after 2m1s [id=ed-eks-01:dev]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

private_ip_of_demo_server = "10.10.4.28"
public_ip_of_demo_server = "3.219.31.97"
PS C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main>
```

7) Now if we visit the Elastic Kubernetes Service (EKS) on AWS console, we can see our EKS created.



Extended support for Kubernetes versions pricing
New prices for extended support started in the April billing cycle. For more information, see the blog post.

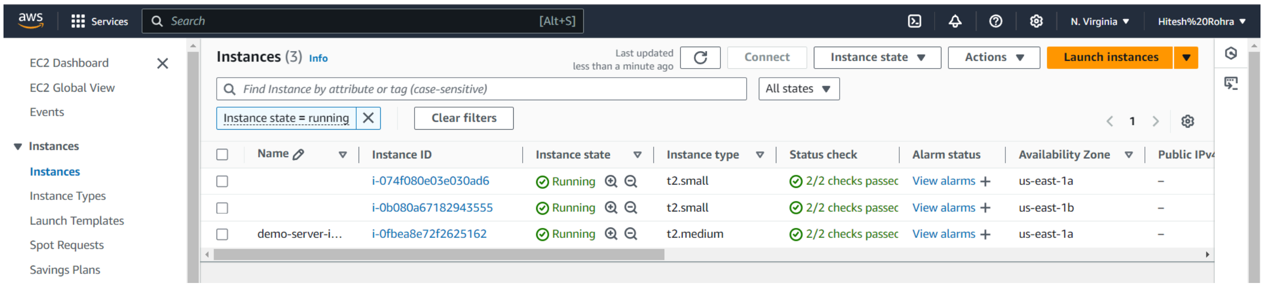
Clusters (1) Info

	Cluster name	Status	Kubernetes version	Support period	Upgrade policy	Created
○	ed-eks-01	Active	1.31	Standard support until November 26, 2025	Extended	22 minutes ago

Step 5:
Use AWS Cloud9 to configure kubectl for the newly created cluster.

As the Cloud9 service is no longer running on AWS, we would be using the EC2 instance created and use SSH to open it on our local terminal.

1) From the above script, an EC2 instance of the name ‘demo-server-instance’ will be created. Click on the name, then on connect and then SSH client.



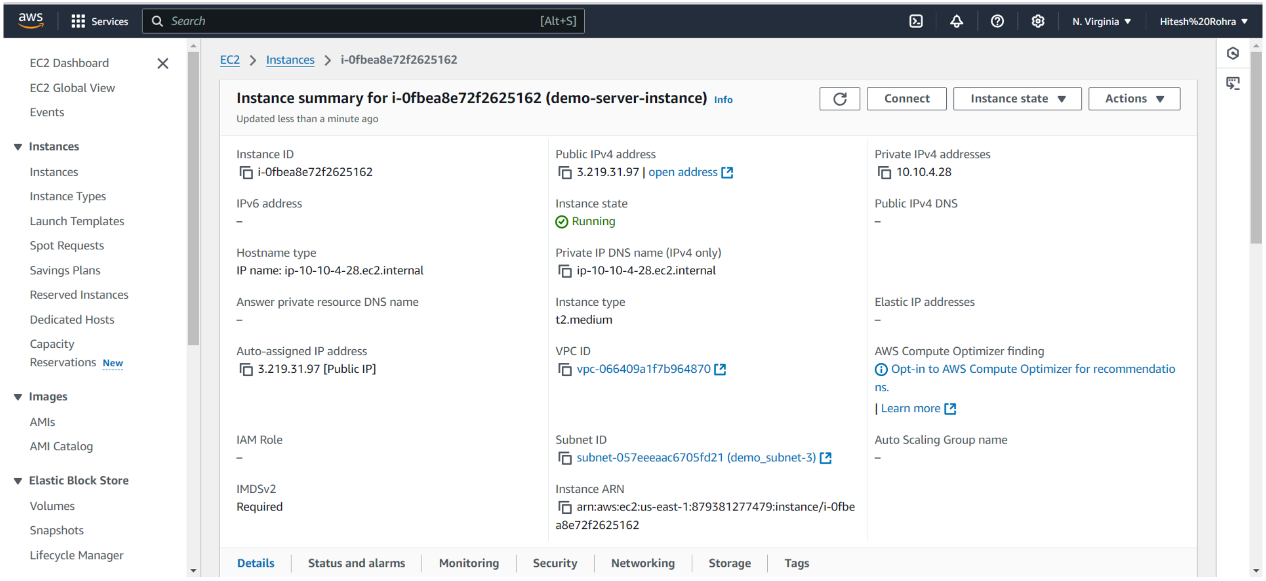
Instances (3) Info

Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
	i-074f080e03e030ad6	Running	t2.small	2/2 checks passec	View alarms +	us-east-1a	–
	i-0b080a67182943555	Running	t2.small	2/2 checks passec	View alarms +	us-east-1b	–
demo-server-i...	i-0fbea8e72f2625162	Running	t2.medium	2/2 checks passec	View alarms +	us-east-1a	–



EC2 > Instances > i-0fbea8e72f2625162

Instance summary for i-0fbea8e72f2625162 (demo-server-instance) Info

Updated less than a minute ago

Instance ID
i-0fbea8e72f2625162

Public IPv4 address
3.219.31.97 | open address

Private IPv4 addresses
10.10.4.28

IPv6 address
–

Instance state
Running

Public IPv4 DNS
–

Hostname type
IP name: ip-10-10-4-28.ec2.internal

Private IP DNS name (IPv4 only)
ip-10-10-4-28.ec2.internal

Answer private resource DNS name
–

Instance type
t2.medium

Elastic IP addresses
–

Auto-assigned IP address
3.219.31.97 [Public IP]

VPC ID
vpc-066409a1f7b964870

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations.
| Learn more

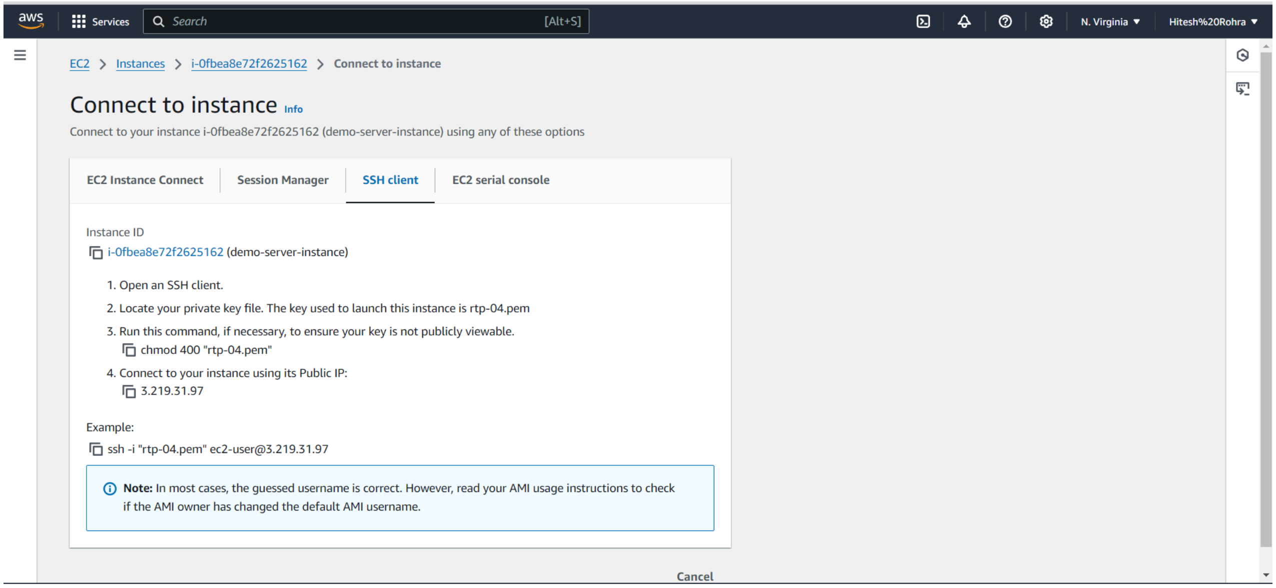
IAM Role
–

Subnet ID
subnet-057eeeaac6705fd21 (demo_subnet-3)

Auto Scaling Group name
–

IMDSv2
Required

Instance ARN
arn:aws:ec2:us-east-1:879381277479:instance/i-0fbea8e72f2625162



EC2 > Instances > i-0fbea8e72f2625162 > Connect to instance

Connect to instance Info

Connect to your instance i-0fbea8e72f2625162 (demo-server-instance) using any of these options

Instance ID
i-0fbea8e72f2625162 (demo-server-instance)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is rtp-04.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   ```
   chmod 400 "rtp-04.pem"
   ```
4. Connect to your instance using its Public IP:
   3.219.31.97

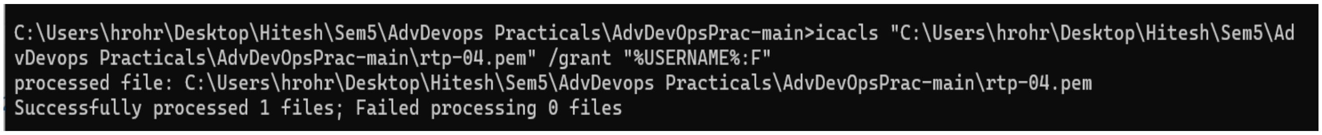
Example:

```
ssh -i "rtp-04.pem" ec2-user@3.219.31.97
```

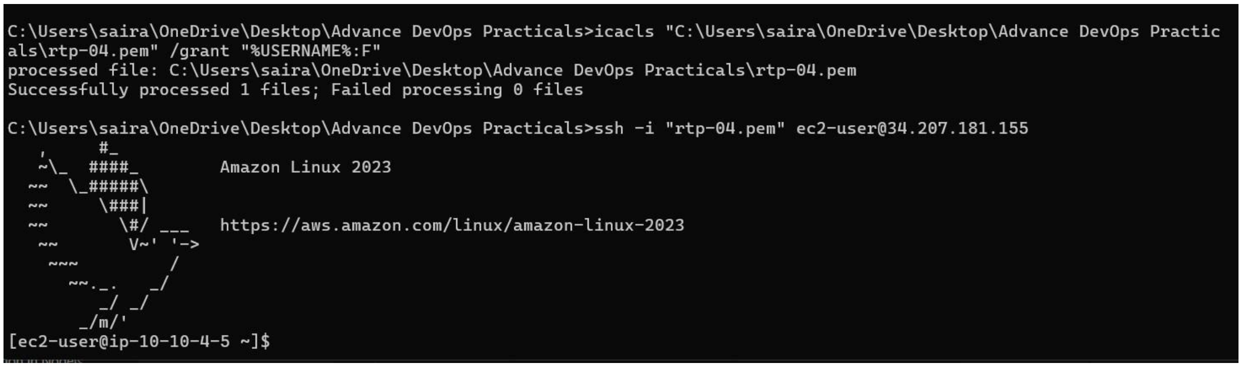
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

2) Use the ssh command to connect to terminal.

   icacls "C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main\rtp-04.pem" /grant "%USERNAME%:F"



```
C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main>icacls "C:\Users\hrohr\Desktop\Hitesh\Sem5\Ad
vDevops Practicals\AdvDevOpsPrac-main\rtp-04.pem" /grant "%USERNAME%:F"
processed file: C:\Users\hrohr\Desktop\Hitesh\Sem5\AdvDevops Practicals\AdvDevOpsPrac-main\rtp-04.pem
Successfully processed 1 files; Failed processing 0 files
```

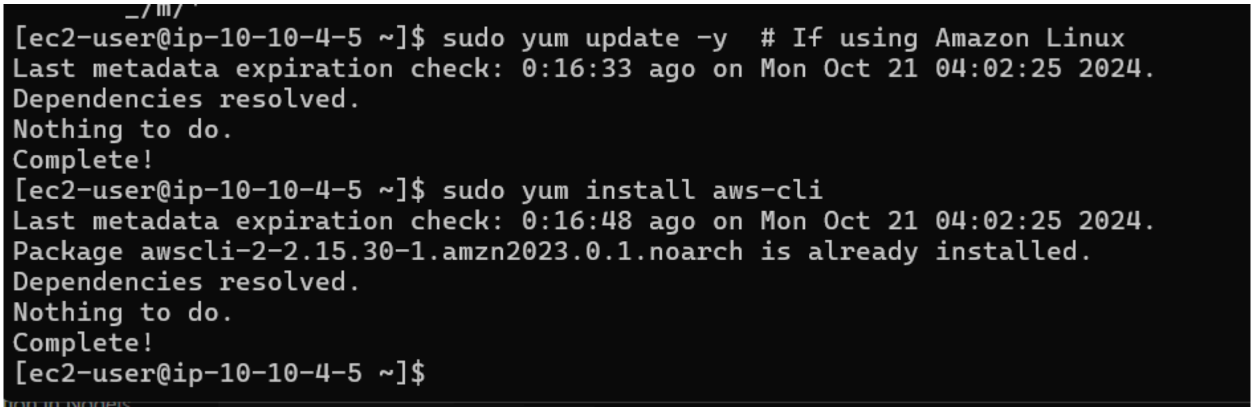


```
C:\Users\saira\OneDrive\Desktop\Advance DevOps Practicals>icacls "C:\Users\saira\OneDrive\Desktop\Advance DevOps Practic
als\rtp-04.pem" /grant "%USERNAME%:F"
processed file: C:\Users\saira\OneDrive\Desktop\Advance DevOps Practicals\rtp-04.pem
Successfully processed 1 files; Failed processing 0 files

C:\Users\saira\OneDrive\Desktop\Advance DevOps Practicals>ssh -i "rtp-04.pem" ec2-user@34.207.181.155
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-10-10-4-5 ~]$
```

3) First, update the services on your terminal/instance.

   sudo yum update -y

   sudo yum install

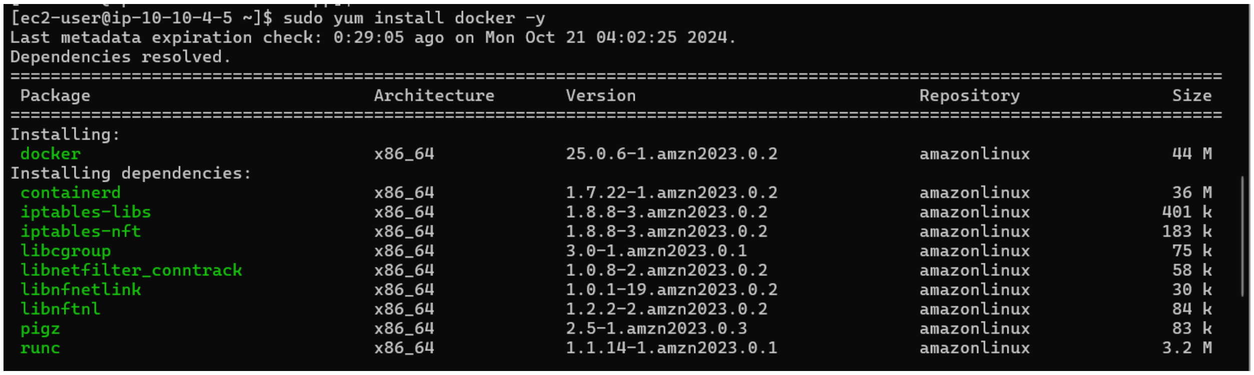


```
[ec2-user@ip-10-10-4-5 ~]$ sudo yum update -y  # If using Amazon Linux
Last metadata expiration check: 0:16:33 ago on Mon Oct 21 04:02:25 2024.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-10-4-5 ~]$ sudo yum install aws-cli
Last metadata expiration check: 0:16:48 ago on Mon Oct 21 04:02:25 2024.
Package awscli-2-2.15.30-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-10-4-5 ~]$
```

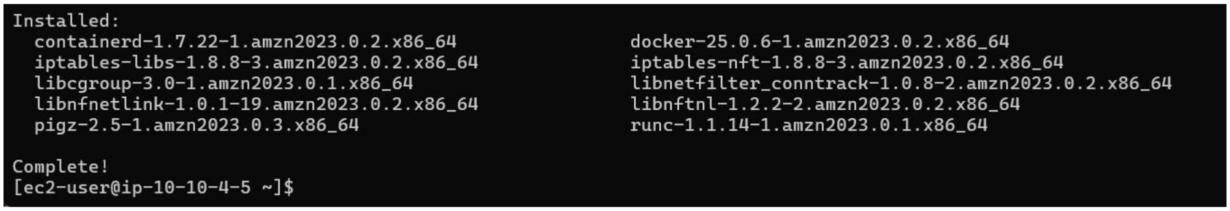
   aws-cli

4) Install docker

   sudo yum install docker -y

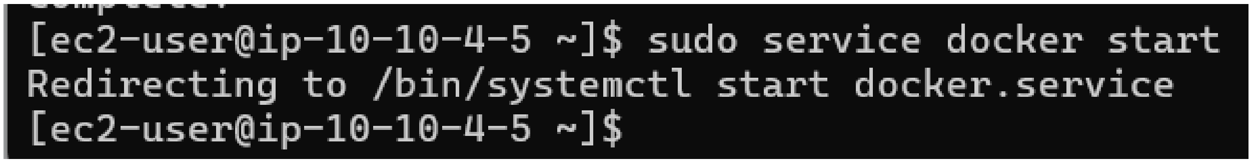


```
[ec2-user@ip-10-10-4-5 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:29:05 ago on Mon Oct 21 04:02:25 2024.
Dependencies resolved.
========================================================================================================================
 Package                            Architecture       Version                             Repository             Size
========================================================================================================================
Installing:
 docker                             x86_64             25.0.6-1.amzn2023.0.2               amazonlinux            44 M
Installing dependencies:
 containerd                         x86_64             1.7.22-1.amzn2023.0.2               amazonlinux            36 M
 iptables-libs                      x86_64             1.8.8-3.amzn2023.0.2                amazonlinux           401 k
 iptables-nft                       x86_64             1.8.8-3.amzn2023.0.2                amazonlinux           183 k
 libcgroup                          x86_64             3.0-1.amzn2023.0.1                  amazonlinux            75 k
 libnetfilter_conntrack             x86_64             1.0.8-2.amzn2023.0.2                amazonlinux            58 k
 libnfnetlink                       x86_64             1.0.1-19.amzn2023.0.2               amazonlinux            30 k
 libnftnl                           x86_64             1.2.2-2.amzn2023.0.2                amazonlinux            84 k
 pigz                               x86_64             2.5-1.amzn2023.0.3                  amazonlinux            83 k
 runc                               x86_64             1.1.14-1.amzn2023.0.1               amazonlinux           3.2 M
```



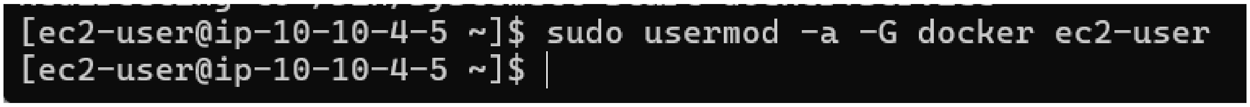
```
Installed:
  containerd-1.7.22-1.amzn2023.0.2.x86_64                  docker-25.0.6-1.amzn2023.0.2.x86_64
  iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  libcgroup-3.0-1.amzn2023.0.1.x86_64                      libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  pigz-2.5-1.amzn2023.0.3.x86_64                           runc-1.1.14-1.amzn2023.0.1.x86_64

Complete!
[ec2-user@ip-10-10-4-5 ~]$
```

5) Start the docker service.
sudo service docker start



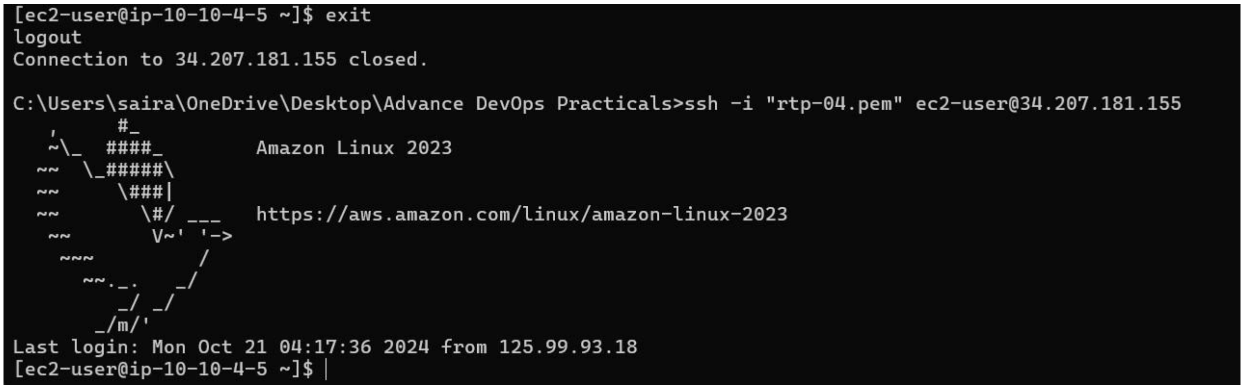
```
[ec2-user@ip-10-10-4-5 ~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
[ec2-user@ip-10-10-4-5 ~]$
```

6) Add the user to the docker group
sudo usermod -a -G docker ec2-user



```
[ec2-user@ip-10-10-4-5 ~]$ sudo usermod -a -G docker ec2-user
[ec2-user@ip-10-10-4-5 ~]$
```

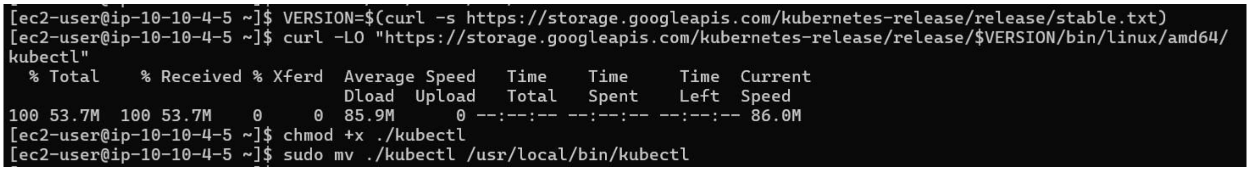
7) Use the **exit** command to logout of the ssh terminal. Use ssh again to relogin.



```
[ec2-user@ip-10-10-4-5 ~]$ exit
logout
Connection to 34.207.181.155 closed.

C:\Users\saira\OneDrive\Desktop\Advance DevOps Practicals>ssh -i "rtp-04.pem" ec2-user@34.207.181.155
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Mon Oct 21 04:17:36 2024 from 125.99.93.18
[ec2-user@ip-10-10-4-5 ~]$
```

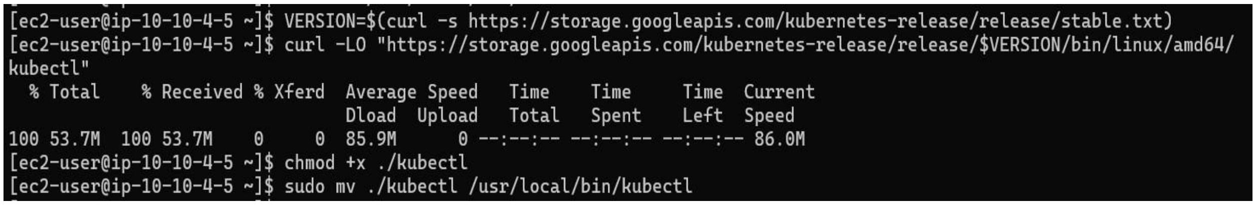
8) Install kubectl

- **VERSION=$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)**
- **curl -LO "https://storage.googleapis.com/kubernetes-release/release/$VERSION/bin/linux/amd64/kubectl"**
- **chmod +x ./kubectl**
- **sudo mv ./kubectl /usr/local/bin/kubect**



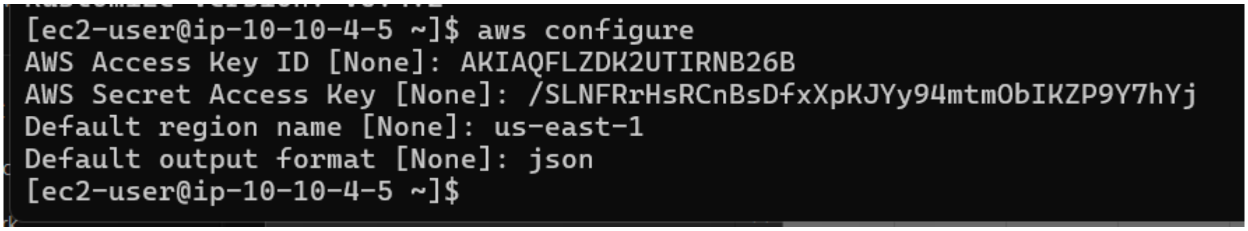
```
[ec2-user@ip-10-10-4-5 ~]$ VERSION=$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)
[ec2-user@ip-10-10-4-5 ~]$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$VERSION/bin/linux/amd64/
kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0  85.9M      0 --:--:-- --:--:-- --:--:-- 86.0M
[ec2-user@ip-10-10-4-5 ~]$ chmod +x ./kubectl
[ec2-user@ip-10-10-4-5 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
```

9) Verify the kubectl installation by using the command
   kubectl version –client



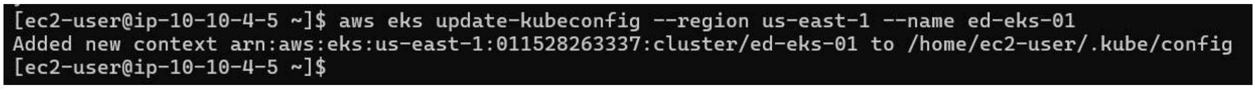
```
[ec2-user@ip-10-10-4-5 ~]$ VERSION=$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)
[ec2-user@ip-10-10-4-5 ~]$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$VERSION/bin/linux/amd64/
kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0  85.9M      0 --:--:-- --:--:-- --:--:-- 86.0M
[ec2-user@ip-10-10-4-5 ~]$ chmod +x ./kubectl
[ec2-user@ip-10-10-4-5 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
```

10) Next, configure the IAM credentials of the user taht we had created before
    aws configure



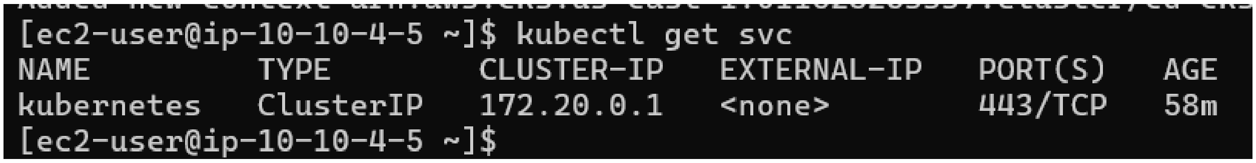
```
[ec2-user@ip-10-10-4-5 ~]$ aws configure
AWS Access Key ID [None]: AKIAQFLZDK2UTIRNB26B
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-10-10-4-5 ~]$
```

11) Now, we need to get the eks cluster and ink to this kubectl, for that, use the following command
    aws eks update-kubeconfig --region us-east-1 --name ed-eks-01



```
[ec2-user@ip-10-10-4-5 ~]$ aws eks update-kubeconfig --region us-east-1 --name ed-eks-01
Added new context arn:aws:eks:us-east-1:011528263337:cluster/ed-eks-01 to /home/ec2-user/.kube/config
[ec2-user@ip-10-10-4-5 ~]$
```

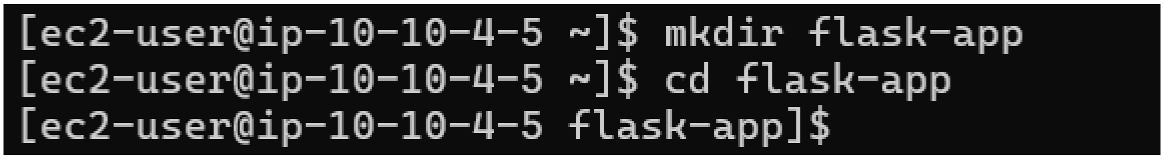
12) Use the following command to verify the kubeconfig setup
    kubectl get svc



```
[ec2-user@ip-10-10-4-5 ~]$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   58m
[ec2-user@ip-10-10-4-5 ~]$
```

Step 6:
Create and deploy a simple flask application on this cluster

1) On this terminal, create a folder for the flask app and navigate to it
   mkdir flask-app
   cd flask-app

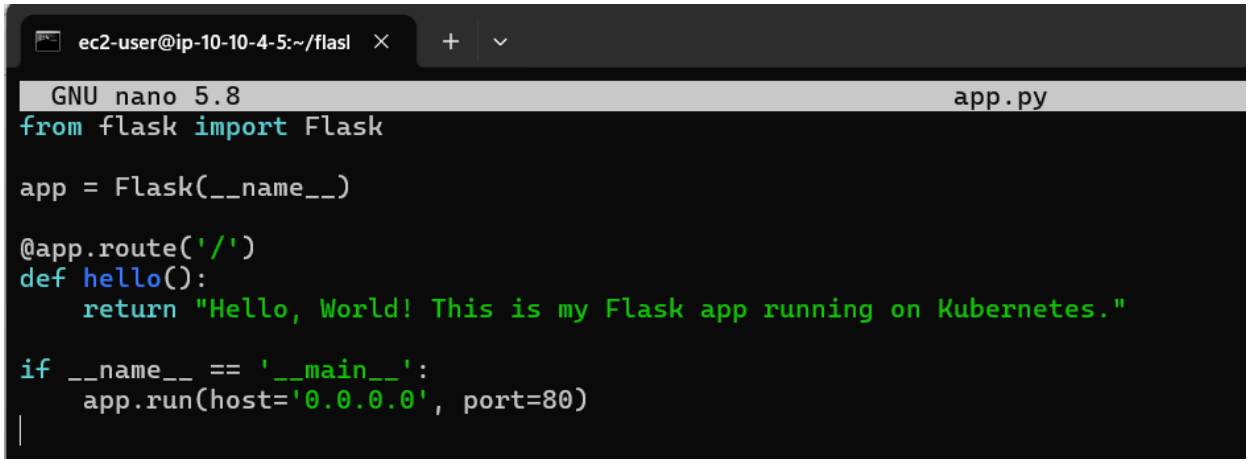


```
[ec2-user@ip-10-10-4-5 ~]$ mkdir flask-app
[ec2-user@ip-10-10-4-5 ~]$ cd flask-app
[ec2-user@ip-10-10-4-5 flask-app]$
```

2) We need to create a app.py file to make the flask application. Use
nano app.py
to create the file and paste the following code.
from flask import
Flask app = Flask(
name_) @app.route('/')
def hello():
return "Hello, World! This is my Flask app running on Kubernetes."

if_name____== '_main_':
app.run(host='0.0.0.0',
port=80)

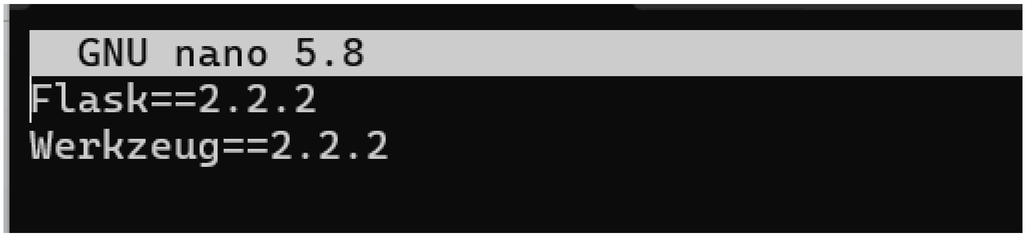
Save the file.



```
  GNU nano 5.8                                                  app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World! This is my Flask app running on Kubernetes."

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

3) Now, a requirements.txt file is required to let docker know which modules are required. Use
nano requirements.txt
to create this file and add the following text.

Flask==2.2.2
Werkzeug==2.2.2



```
  GNU nano 5.8
Flask==2.2.2
Werkzeug==2.2.2
```

4) Now, create a Dockerfile that would guide docker on how to run the commands and in what order.
   nano Dockerfile
   Paste the following code

   # Use the official Python image from the Docker Hub
   FROM python:3.9-slim

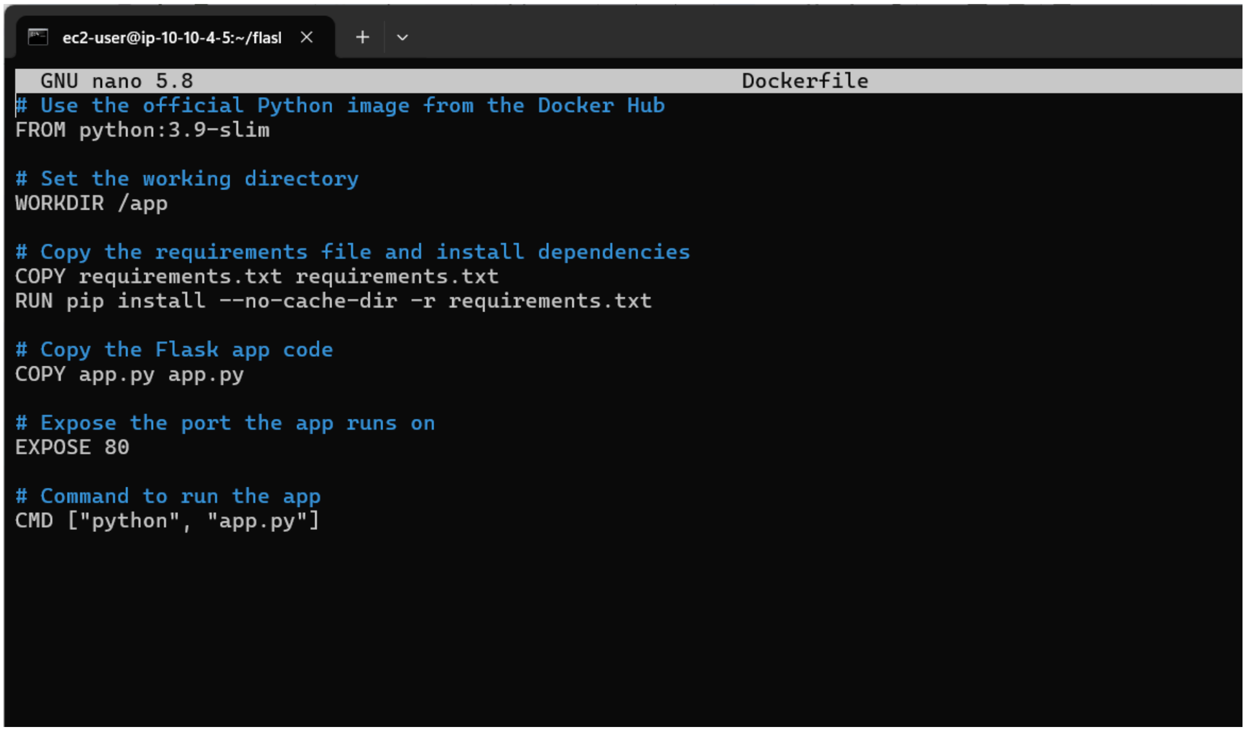
   # Set the working directory
   WORKDIR /app

   # Copy the requirements file and install dependencies
   COPY requirements.txt requirements.txt
   RUN pip install --no-cache-dir -r requirements.txt

   # Copy the Flask app code
   COPY app.py app.py

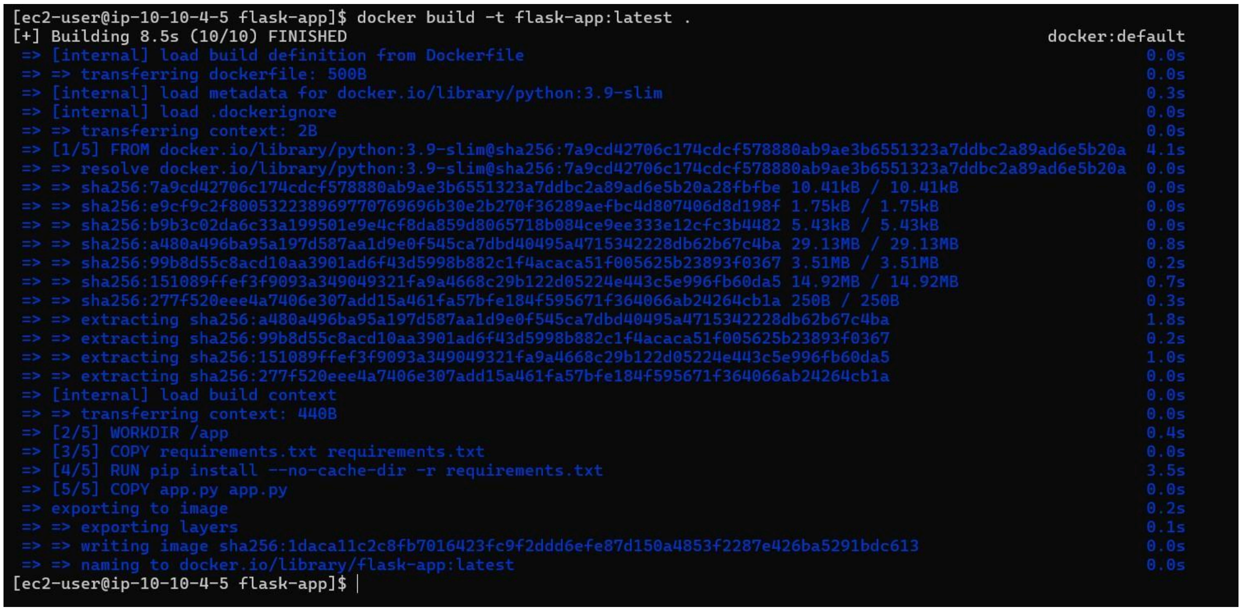
   # Expose the port the app runs on
   EXPOSE 80

   # Command to run the app
   CMD ["python", "app.py"]



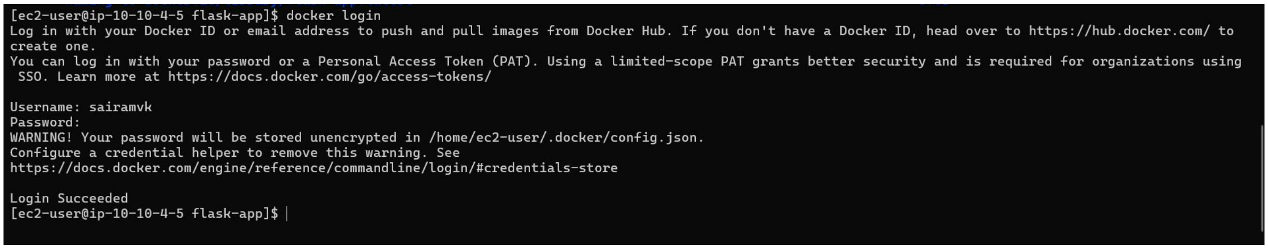
```
  GNU nano 5.8                                         Dockerfile
# Use the official Python image from the Docker Hub
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the Flask app code
COPY app.py app.py

# Expose the port the app runs on
EXPOSE 80

# Command to run the app
CMD ["python", "app.py"]
```

5) Build the Docker Image
   docker build -t flask-app:latest .



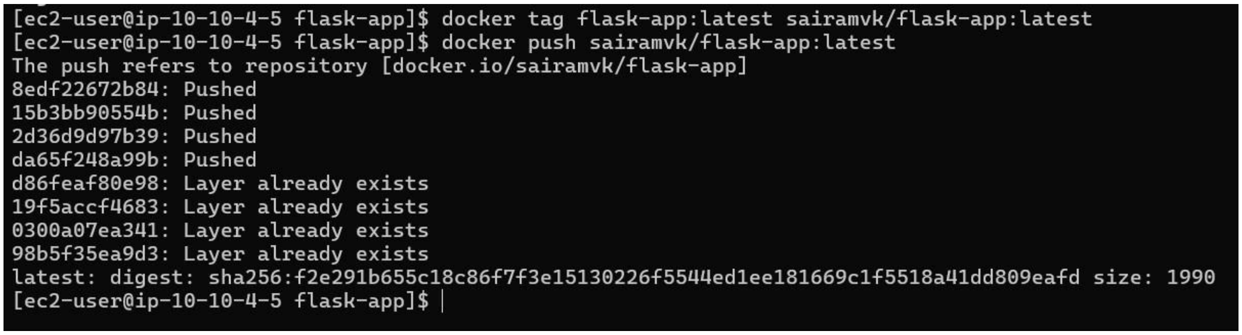
```
[ec2-user@ip-10-10-4-5 flask-app]$ docker build -t flask-app:latest .
[+] Building 8.5s (10/10) FINISHED                                                                              docker:default
 => [internal] load build definition from Dockerfile                                                                      0.0s
 => => transferring dockerfile: 500B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                        0.3s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                           0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad6e5b20a  4.1s
 => => resolve docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad6e5b20a  0.0s
 => => sha256:7a9cd42706c174cdcf578880ab9ae3b6551323a7ddbc2a89ad6e5b20a28fbfbe 10.41kB / 10.41kB                          0.0s
 => => sha256:e9cf9c2f800532238969770769696b30e2b270f36289aefbc4d807406d8d198f 1.75kB / 1.75kB                            0.0s
 => => sha256:b9b3c02da6c33a199501e9e4cf8da859d8065718b084ce9ee333e12cfc3b4482 5.43kB / 5.43kB                            0.0s
 => => sha256:a480a496ba95a197d587aa1d9e0f545ca7dbd40495a4715342228db62b67c4ba 29.13MB / 29.13MB                          0.8s
 => => sha256:99b8d55c8acd10aa3901ad6f43d5998b882c1f4acaca51f005625b23893f0367 3.51MB / 3.51MB                            0.2s
 => => sha256:151089ffef3f9093a349049321fa9a4668c29b122d05224e443c5e996fb60da5 14.92MB / 14.92MB                          0.7s
 => => sha256:277f520eee4a7406e307add15a461fa57bfe184f595671f364066ab24264cb1a 250B / 250B                                0.3s
 => => extracting sha256:a480a496ba95a197d587aa1d9e0f545ca7dbd40495a4715342228db62b67c4ba                                 1.8s
 => => extracting sha256:99b8d55c8acd10aa3901ad6f43d5998b882c1f4acaca51f005625b23893f0367                                 0.2s
 => => extracting sha256:151089ffef3f9093a349049321fa9a4668c29b122d05224e443c5e996fb60da5                                 1.0s
 => => extracting sha256:277f520eee4a7406e307add15a461fa57bfe184f595671f364066ab24264cb1a                                 0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 440B                                                                                         0.0s
 => [2/5] WORKDIR /app                                                                                                    0.4s
 => [3/5] COPY requirements.txt requirements.txt                                                                          0.0s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                              3.5s
 => [5/5] COPY app.py app.py                                                                                              0.0s
 => exporting to image                                                                                                    0.2s
 => => exporting layers                                                                                                   0.1s
 => => writing image sha256:1daca11c2c8fb7016423fc9f2ddd6efe87d150a4853f2287e426ba5291bdc613                              0.0s
 => => naming to docker.io/library/flask-app:latest                                                                       0.0s
[ec2-user@ip-10-10-4-5 flask-app]$ |
```

6) Use **docker login** to login to your docker account



```
[ec2-user@ip-10-10-4-5 flask-app]$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to
create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using
 SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: sairamvk
Password:
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[ec2-user@ip-10-10-4-5 flask-app]$ |
```

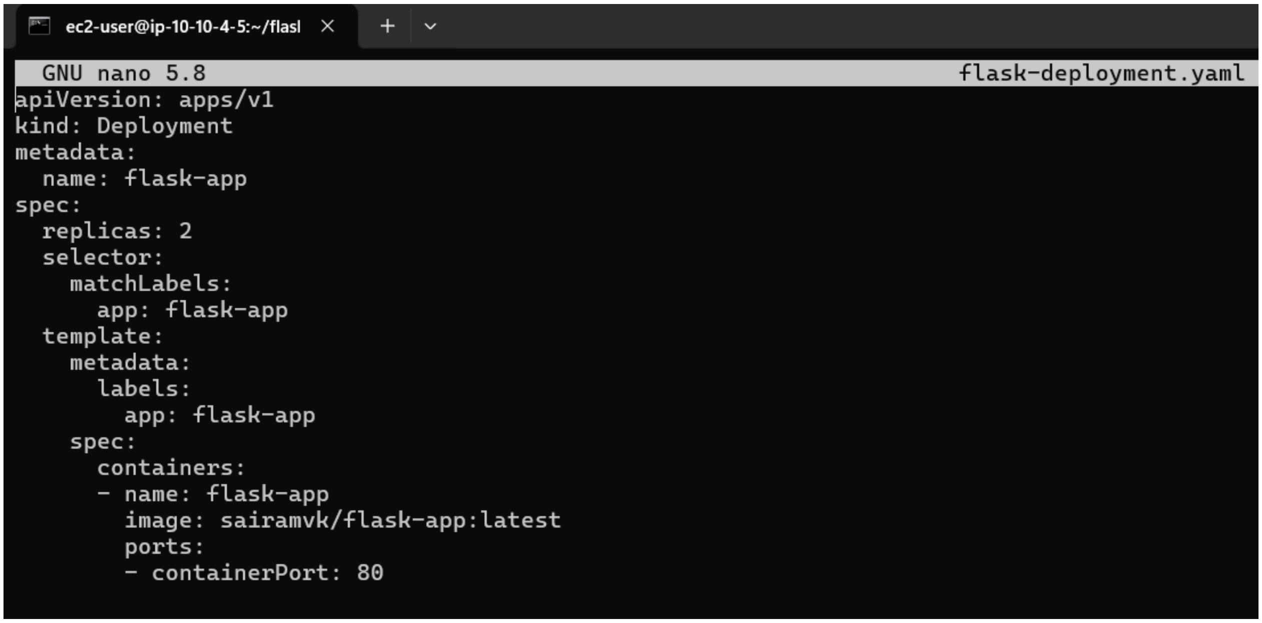
7) Tag the image created and push it.



```
[ec2-user@ip-10-10-4-5 flask-app]$ docker tag flask-app:latest sairamvk/flask-app:latest
[ec2-user@ip-10-10-4-5 flask-app]$ docker push sairamvk/flask-app:latest
The push refers to repository [docker.io/sairamvk/flask-app]
8edf22672b84: Pushed
15b3bb90554b: Pushed
2d36d9d97b39: Pushed
da65f248a99b: Pushed
d86feaf80e98: Layer already exists
19f5accf4683: Layer already exists
0300a07ea341: Layer already exists
98b5f35ea9d3: Layer already exists
latest: digest: sha256:f2e291b655c18c86f7f3e15130226f5544ed1ee181669c1f5518a41dd809eafd size: 1990
[ec2-user@ip-10-10-4-5 flask-app]$ |
```

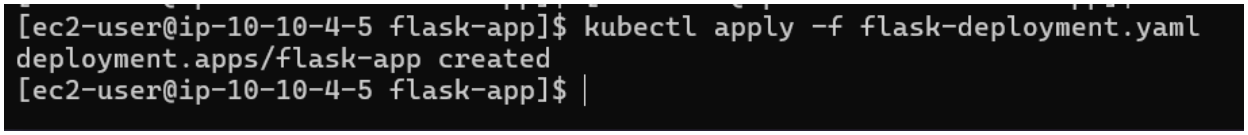
8) Now, deploy the application to kubernetes. Create flask-deployment.yaml file using **nano flask-deployment.yaml** and add the following content.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: <your-dockerhub-username>/flask-app:latest
        ports:
        - containerPort: 80
```



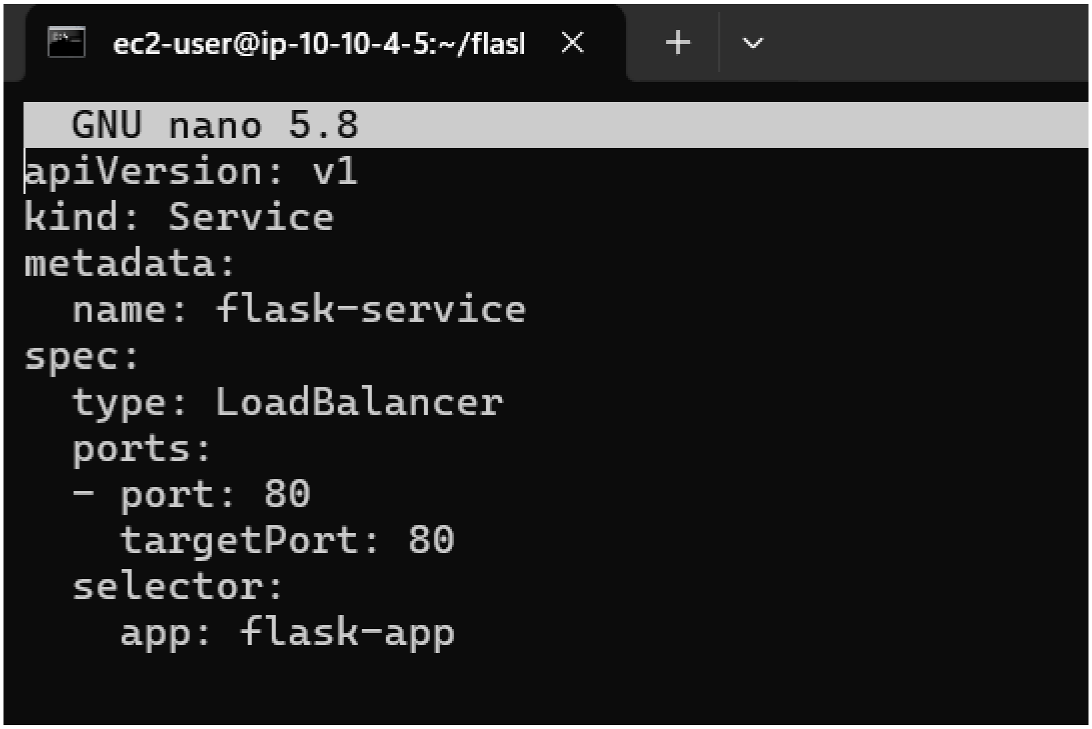
```
  GNU nano 5.8                                      flask-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: sairamvk/flask-app:latest
        ports:
        - containerPort: 80
```

9) Apply the created deployment
kubectl apply -f flask-deployment.yaml



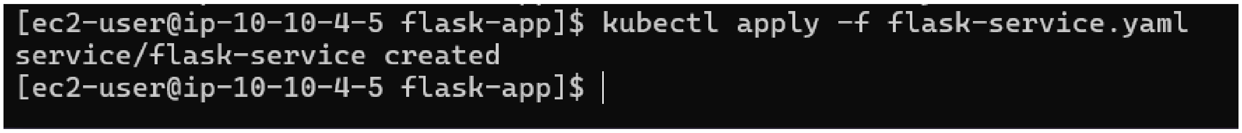
```
[ec2-user@ip-10-10-4-5 flask-app]$ kubectl apply -f flask-deployment.yaml
deployment.apps/flask-app created
[ec2-user@ip-10-10-4-5 flask-app]$
```

10) To expose this application, we need to create a flask-service.yaml file. Use **nano flask-service.yaml** and add the following code.
apiVersion: v1
kind: Service
metadata:
name:
flask-service spec:
type:
LoadBalancer
ports:
- port: 80
targetPort: 80
selector:
app: flask-app



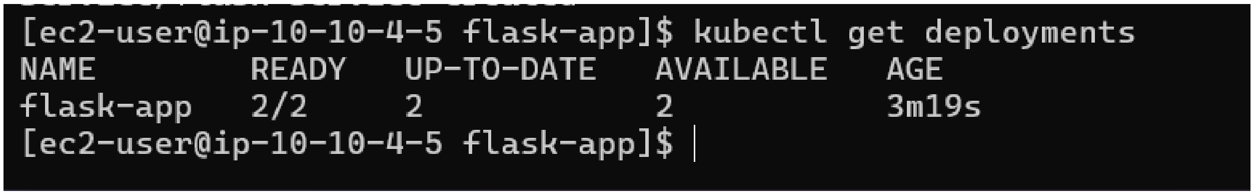
```
  GNU nano 5.8
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: flask-app
```

11) Apply this service to kubernetes.
kubectl apply -f flask-service.yaml



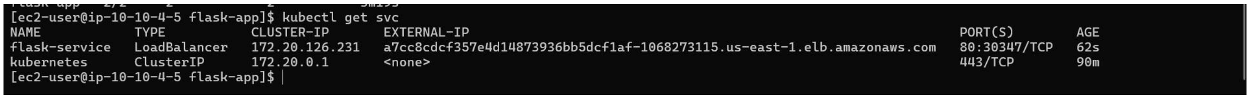
```
[ec2-user@ip-10-10-4-5 flask-app]$ kubectl apply -f flask-service.yaml
service/flask-service created
[ec2-user@ip-10-10-4-5 flask-app]$
```

12) Get the status of your deployment.
kubectl get deployments



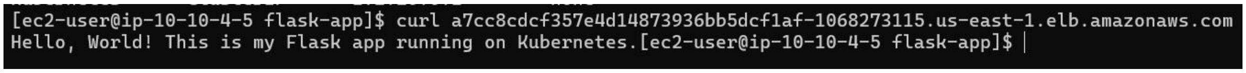
```
[ec2-user@ip-10-10-4-5 flask-app]$ kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
flask-app   2/2     2            2           3m19s
[ec2-user@ip-10-10-4-5 flask-app]$ |
```

13) Get the status of your service
kubectl get svc



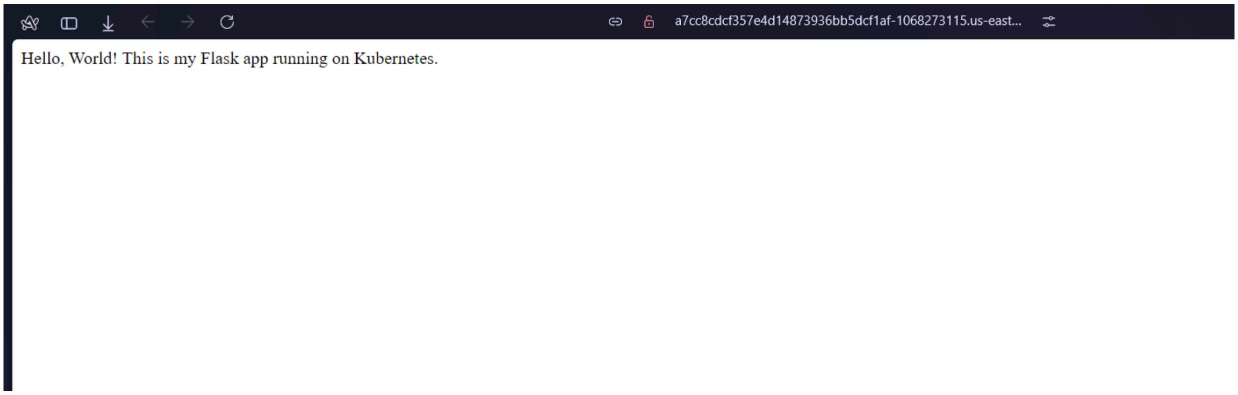
```
[ec2-user@ip-10-10-4-5 flask-app]$ kubectl get svc
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
flask-service   LoadBalancer   172.20.126.231  a7cc8cdcf357e4d14873936bb5dcf1af-1068273115.us-east-1.elb.amazonaws.com   80:30347/TCP   62s
kubernetes      ClusterIP      172.20.0.1      <none>                                                                     443/TCP        90m
[ec2-user@ip-10-10-4-5 flask-app]$ |
```

14) From the services part, we get the external IP of the service we created. Now run command
curl a7cc8cdcf357e4d14873936bb5dcf1af-1068273115.us-east-1.elb.amazonaws.com
To test the deployment.
It should give output: **"Hello, World! This is my Flask app running on Kubernetes."**



```
[ec2-user@ip-10-10-4-5 flask-app]$ curl a7cc8cdcf357e4d14873936bb5dcf1af-1068273115.us-east-1.elb.amazonaws.com
Hello, World! This is my Flask app running on Kubernetes.[ec2-user@ip-10-10-4-5 flask-app]$ |
```

Access this link on a browser and check this.



Hello, World! This is my Flask app running on Kubernetes.

Guidelines:

- The setup of IAM user group and user is required for this case study.
- The user group must have all the permissions required to access and modify various services in AWS.
- Make sure to keep the Access keys of the IAM user safe on your system.
- Keep the .pem file of teh private key safely in a folder that you can access.
- Make sure that the EC2 instance has SSH permissions (Port 22).
- Install all the required services on the instance.

Key Points:

One of the most important things is that in this case study, AWS Cloud9 IDE could not be used as it has been deprecated by AWS themselves from 25th July 2024. To tackle this issue, we have created an EC2 instance that is linked with the VPC linked with the Kubernetes cluster and then with the help of SSH link that instance to our local terminal. On this instance we are setting up the kubectl and flask app deployment.

Conclusion:

Thus, we have understood the interrealtion between various services provided in DevOps such as Terraform, Kubernetes, AWS, Docker, etc. We have seen how Terraform is used to create the cluster on an AWS console with the help of the access keys of an IAM user. Through this, we have understood the meaning of Terraform being a Infrastructure as Code (IaC) service. After the code is run, we access the instance using SSH on our local machine where we install docker. The main reason for Docker is to containerize whatever application that needs to be deployed has to be set up. This is done by using the existing images (in this case, Python) from the Docker registry). After that, to use the Kubernetes cluster that has been set up, we use 2 CLI tools, kubectl and aws cli. With the help of both, we acces the already set up Cluster and deploy our flask application, whose output can be seen using the curl command.