

VPC PEERING USING TERRAFORM

- Open aws console and go to seoul region
- Create EC2 instance
- Connect to that EC2 instance in git bash
- Go to IAM service and create an user 'terra' and then create access key as shown in below slides

The screenshot displays the AWS Management Console interface for launching an EC2 instance. The top navigation bar shows the 'Instances' section, indicating that no instances are currently running in the selected region. The main content area is titled 'Launch an instance' and provides a step-by-step guide. The 'Name and tags' section contains a text input field with the value 'terraform'. The 'Application and OS Images (Amazon Machine Image)' section features a search bar and a 'Quick Start' tab. Under the 'Quick Start' tab, several AMI options are listed: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, and SUSE Li. A 'Browse more AMIs' link is also present.

```
#  
###  
#####  
####|  
#/  
V~' ->  
- - -  
_/_m/'
```

Amazon Linux 2023

<https://aws.amazon.com/linux/amazon-linux-2023>

[ec2-user@ip-172-31-26-237 ~]\$ |





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](#)

- ☐ **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.

• 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.

Choose one or more policies to attach to your new user.

	Policy name	Type	Attached entities
<input type="checkbox"/>	 AccessAnalyzerServiceRolePolicy	AWS managed	0
<input type="checkbox"/>	 AdministratorAccess	AWS managed - job function	0
<input type="checkbox"/>	 AdministratorAccess-Amplify	AWS managed	0
<input type="checkbox"/>	 AdministratorAccess-AWSElasticBeanstalk	AWS managed	0

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

User details

User name
terra

Console password type
None

Require password reset
No

Permissions summary

< 1 >

Name ?	Type	Used as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonVPCFullAccess	AWS managed	Permissions policy

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.

Add new tag

You can add up to 50 more tags.

Cancel

Previous

Create user

[IAM](#) > [Users](#)

Users (1) [Info](#)

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

[Q Search](#)

< 1 >



<input type="checkbox"/>	User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access
<input type="checkbox"/>	terra	/	0	-	-	-	-	-

[IAM](#) > [Users](#) > [terra](#)

terra [Info](#)

Delete

Summary

ARN
[arn:aws:iam::891377141179:user/terra](#)

Created
September 08, 2024, 19:38 (UTC+05:30)

Console access
Disabled

Last console sign-in
-

Access key 1
[Create access key](#)

[Permissions](#) | [Groups](#) | [Tags](#) | [Security credentials](#) | [Access Advisor](#)

Permissions policies (2)

Permissions are defined by policies attached to the user directly or through groups.



Remove

Add permissions [v](#)

[Q Search](#)

Filter by Type

All types

< 1 >



<input type="checkbox"/>	Policy name ?	Type	Attached via ?
<input type="checkbox"/>	AmazonEC2FullAccess	AWS managed	Directly
<input type="checkbox"/>	AmazonVPCFullAccess	AWS managed	Directly

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.



Alternatives recommended

- Use [AWS CloudShell](#), a browser-based CLI, to run commands. [Learn more](#)
- Use the [AWS CLI V2](#) and enable authentication through a user in IAM Identity Center. [Learn more](#)

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved.

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

AKIA47CRXYW56DHIBP2D

Secret access key

***** [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](#).

[Download .csv file](#)

[Done](#)

- Go to EC2 server and configure the access key
- Command for configure: aws configure
- Enter the access key, secret access key and region code
- Install the AWS CLI and Terraform
- Create directory – mkdir terraform
- Change to that directory – cd terraform
- In terraform directory create 3 block i.e, 1.terrafromblock.tf, 2.provider.tf, 3.resource.tf

1.terraformblock.tf:

```
terraform {  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "5.65.0"  
    }  
  }  
}
```

2.provider.tf:

```
provider "aws" {  
  region = "us-east-1"  
  profile = "default"  
}
```

3.resource.tf:

Create VPC 1

```
resource "aws_vpc" "vpc1" {  
  cidr_block = "10.0.0.0/16"  
  tags = {  
    Name = "VPC1"  
  }  
}
```

Create Subnet for VPC 1

```
resource "aws_subnet" "subnet1" {  
  vpc_id      = aws_vpc.vpc1.id  
  cidr_block   = "10.0.1.0/24"  
  availability_zone = "us-east-1a"  
  map_public_ip_on_launch = true  
  tags = {  
    Name = "Subnet1"  
  }  
}
```

```
}  
}
```

Create Internet Gateway for VPC 1

```
resource "aws_internet_gateway" "igw1" {  
  vpc_id = aws_vpc.vpc1.id  
  tags = {  
    Name = "IGW1"  
  }  
}
```

Create Route Table for VPC 1 and associate with the subnet

```
resource "aws_route_table" "rt1" {  
  vpc_id = aws_vpc.vpc1.id  
  route {  
    cidr_block = "0.0.0.0/0"  
    gateway_id = aws_internet_gateway.igw1.id  
  }  
}
```

```
resource "aws_route_table_association" "rta1" {  
  subnet_id    = aws_subnet.subnet1.id  
  route_table_id = aws_route_table.rt1.id  
}
```

Create Security Group for VPC 1

```
resource "aws_security_group" "sg1" {  
  name = "vpc1-sg"  
  vpc_id = aws_vpc.vpc1.id  
  
  ingress {
```

```

    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}

egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}
}

# Create EC2 Instance in VPC 1
resource "aws_instance" "instance1" {
    ami          = "ami-0e86e20dae9224db8" # Replace with a valid AMI ID
    instance_type = "t2.micro"
    subnet_id    = aws_subnet.subnet1.id
    key_name     = "project" # Replace with your AWS key pair name
    vpc_security_group_ids = [aws_security_group.sg1.id]
    associate_public_ip_address = true
    tags = {
        Name = "Instance1"
    }
}

# Create VPC 2
resource "aws_vpc" "vpc2" {
    cidr_block = "10.1.0.0/16"
    tags = {
        Name = "VPC2"
    }
}

```

```

}

# Create Subnet for VPC 2
resource "aws_subnet" "subnet2" {
  vpc_id      = aws_vpc.vpc2.id
  cidr_block  = "10.1.1.0/24"
  availability_zone = "us-east-1b"
  map_public_ip_on_launch = true
  tags = {
    Name = "Subnet2"
  }
}

# Create Internet Gateway for VPC 2
resource "aws_internet_gateway" "igw2" {
  vpc_id = aws_vpc.vpc2.id
  tags = {
    Name = "IGW2"
  }
}

# Create Route Table for VPC 2 and associate with the subnet
resource "aws_route_table" "rt2" {
  vpc_id = aws_vpc.vpc2.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw2.id
  }
}

resource "aws_route_table_association" "rta2" {
  subnet_id   = aws_subnet.subnet2.id
  route_table_id = aws_route_table.rt2.id
}

```


Create Security Group for VPC 2

```
resource "aws_security_group" "sg2" {
```

```
  name = "vpc2-sg"
```

```
  vpc_id = aws_vpc.vpc2.id
```

```
  ingress {
```

```
    from_port = 22
```

```
    to_port   = 22
```

```
    protocol = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }
```

```
    egress {
```

```
      from_port = 0
```

```
      to_port   = 0
```

```
      protocol = "-1"
```

```
      cidr_blocks = ["0.0.0.0/0"]
```

```
    }
```

```
  }
```

Create EC2 Instance in VPC 2

```
resource "aws_instance" "instance2" {
```

```
  ami = "ami-0e86e20dae9224db8" # Replace with a valid AMI ID
```

```
  instance_type = "t2.micro"
```

```
  subnet_id = aws_subnet.subnet2.id
```

```
  key_name = "project" # Replace with your AWS key pair name
```

```
  vpc_security_group_ids = [aws_security_group.sg2.id]
```

```
  associate_public_ip_address = true
```

```
  tags = {
```

```
    Name = "Instance2"
```

```
  }
```

```
}
```

Create VPC Peering Connection

```
resource "aws_vpc_peering_connection" "peer" {
```

```
vpc_id      = aws_vpc.vpc1.id
peer_vpc_id = aws_vpc.vpc2.id
auto_accept = true
tags = {
  Name = "VPC1-to-VPC2-Peering"
}
}

# Update Route Tables to include Peering Connection
resource "aws_route" "route_to_vpc2" {
  route_table_id      = aws_route_table.rt1.id
  destination_cidr_block = aws_vpc.vpc2.cidr_block
  vpc_peering_connection_id = aws_vpc_peering_connection.peer.id
}

resource "aws_route" "route_to_vpc1" {
  route_table_id      = aws_route_table.rt2.id
  destination_cidr_block = aws_vpc.vpc1.cidr_block
  vpc_peering_connection_id = aws_vpc_peering_connection.peer.id
}
```

```
root@ip-172-31-26-237:~
```

```
[root@ip-172-31-26-237 ~]# aws configure
AWS Access Key ID [None]: AKIA47CRXYW56DHIBP2D
AWS Secret Access Key [None]: dRZTSRJ94E841TwteCcx6hZhs/nnZ4sMUyhSUw
Default region name [None]: table
```

```
root@ip-172-31-26-237:~/.aws
```

```
[root@ip-172-31-26-237 ~]# cd .aws
[root@ip-172-31-26-237 .aws]# ls
config  credentials
[root@ip-172-31-26-237 .aws]# cat credentials
[default]
aws_access_key_id = AKIA47CRXYW56DHIBP2D
aws_secret_access_key = dRZTSRJ94E841TwteCcx6hZhs/nnZ4sMUyhSUw
[root@ip-172-31-26-237 .aws]# |
```

```
[root@ip-172-31-26-237 ~]# cd terraform
[root@ip-172-31-26-237 terraform]# vi terraformblock.tf
[root@ip-172-31-26-237 terraform]# vi provider.tf
[root@ip-172-31-26-237 terraform]# vi resource.tf
[root@ip-172-31-26-237 terraform]# vi userdata.sh
[root@ip-172-31-26-237 terraform]# ls
provider.tf  resource.tf  terraformblock.tf  userdata.sh
[root@ip-172-31-26-237 terraform]# |
```

- Once blocks created give command-terraform init
- Terraform validate
- Terraform plan
- Terraform apply
- After terraform apply 17 resource will add as shown in below figure

```

    + requester (known after apply)
  }

Plan: 17 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

aws_vpc.vpc2: Creating...
aws_vpc.vpc1: Creating...
aws_vpc.vpc1: Creation complete after 1s [id=vpc-04f40dd44c82202a9]
aws_vpc.vpc2: Creation complete after 1s [id=vpc-01dcf746567da5690]
aws_subnet.subnet1: Creating...
aws_internet_gateway.igw1: Creating...
aws_security_group.sg1: Creating...
aws_security_group.sg2: Creating...
aws_vpc_peering_connection.peer: Creating...
aws_internet_gateway.igw2: Creating...
aws_subnet.subnet2: Creating...
aws_internet_gateway.igw1: Creation complete after 0s [id=igw-09833c987a9f9b0b3]
aws_route_table.rt1: Creating...
aws_internet_gateway.igw2: Creation complete after 0s [id=igw-07c692b1f8565c099]
aws_route_table.rt2: Creating...
aws_vpc_peering_connection.peer: Creation complete after 1s [id=pcx-0d06002cb3ebfef58]
aws_route_table.rt1: Creation complete after 1s [id=rtb-07d7bd735471c5add]
aws_route.route_to_vpc2: Creating...
aws_route_table.rt2: Creation complete after 1s [id=rtb-083ce9f3a207544c6]
aws_route.route_to_vpc1: Creating...
aws_route.route_to_vpc1: Creation complete after 0s [id=r-rtb-083ce9f3a207544c6179966490]
aws_security_group.sg1: Creation complete after 2s [id=sg-0b9ebcea47efcefad]
aws_security_group.sg2: Creation complete after 2s [id=sg-03ce024d8d8cf29ad]
aws_route.route_to_vpc2: Creation complete after 1s [id=r-rtb-07d7bd735471c5add3322942084]
aws_subnet.subnet1: Still creating... [10s elapsed]
aws_subnet.subnet2: Still creating... [10s elapsed]
aws_subnet.subnet1: Creation complete after 11s [id=subnet-081891b64a575e852]
aws_route_table_association.rta1: Creating...
aws_instance.instance1: Creating...
aws_subnet.subnet2: Creation complete after 11s [id=subnet-005fd0f688db9cb54]
aws_route_table_association.rta2: Creating...
aws_instance.instance2: Creating...
aws_route_table_association.rta2: Creation complete after 0s [id=rtbassoc-058082127bcb43e32]
aws_route_table_association.rta1: Creation complete after 0s [id=rtbassoc-036fe4e0886febf0f]
aws_instance.instance1: Still creating... [10s elapsed]
aws_instance.instance2: Still creating... [10s elapsed]
aws_instance.instance1: Still creating... [20s elapsed]
aws_instance.instance2: Still creating... [20s elapsed]
aws_instance.instance1: Still creating... [30s elapsed]
aws_instance.instance2: Still creating... [30s elapsed]
aws_instance.instance2: Creation complete after 32s [id=i-04fa7bea7bff29939]
aws_instance.instance1: Still creating... [40s elapsed]
aws_instance.instance1: Creation complete after 42s [id=i-0ec6c3184205b6410]

Apply complete! Resources: 17 added, 0 changed, 0 destroyed.
root@ip-172-31-17-248:~/terraform#

```

- Go to aws console and check resources created or not
- Once created open the instances and connect to the server and check both instances are peering or not by using curl command as shown in below figure

```
root@ip-10-0-1-94:/var/www/html# rm index.html
root@ip-10-0-1-94:/var/www/html# vi index.html
root@ip-10-0-1-94:/var/www/html# systemctl restart apache2
root@ip-10-0-1-94:/var/www/html# curl 10.0.1.94:80
hello this is webserver 1
root@ip-10-0-1-94:/var/www/html#
```

i-02eee3e7708c52cce (Instance1)

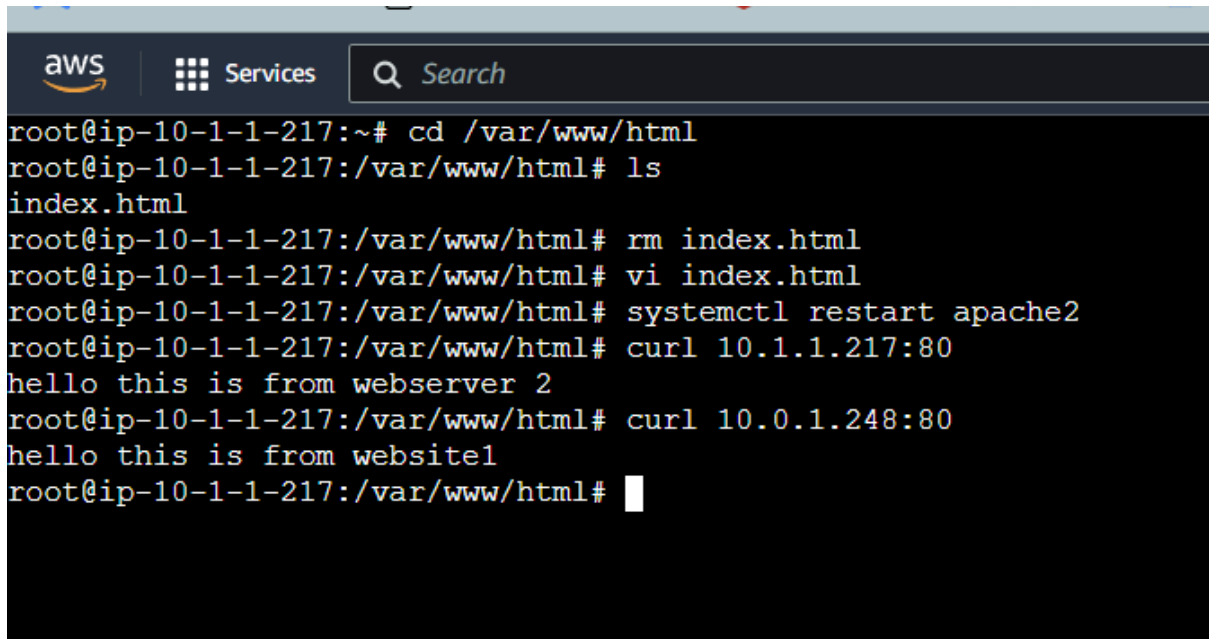
PublicIPs: 3.80.230.62 PrivateIPs: 10.0.1.94

```
aws Services Search
root@ip-10-1-1-68:/var/www/html# curl 10.1.1.68:80
hello this is from webserver2
root@ip-10-1-1-68:/var/www/html#
```

```
ASUS Software Port... MyASUS Software -... McAfee LiveSafe Gmail
aws Services Search
root@ip-10-0-1-248:~# cd /var/www/html
root@ip-10-0-1-248:/var/www/html# ls
index.html
root@ip-10-0-1-248:/var/www/html# rm index.html
root@ip-10-0-1-248:/var/www/html# vi index.html
root@ip-10-0-1-248:/var/www/html# systemctl restart apache2
root@ip-10-0-1-248:/var/www/html# curl 10.0.1.248
hello this is from website1
root@ip-10-0-1-248:/var/www/html#
```

```
aws Services Search
root@ip-10-1-1-217:~# cd /var/www/html
root@ip-10-1-1-217:/var/www/html# ls
index.html
root@ip-10-1-1-217:/var/www/html# rm index.html
root@ip-10-1-1-217:/var/www/html# vi index.html
root@ip-10-1-1-217:/var/www/html# systemctl restart apache2
root@ip-10-1-1-217:/var/www/html# curl 10.1.1.217:80
hello this is from webserver 2
root@ip-10-1-1-217:/var/www/html#
```

```
aws Services Search
root@ip-10-0-1-248:/var/www/html# curl 10.0.1.248
hello this is from website1
root@ip-10-0-1-248:/var/www/html# curl 10.1.1.217:80
hello this is from webserver 2
root@ip-10-0-1-248:/var/www/html#
```



```
aws Services Search
root@ip-10-1-1-217:~# cd /var/www/html
root@ip-10-1-1-217:/var/www/html# ls
index.html
root@ip-10-1-1-217:/var/www/html# rm index.html
root@ip-10-1-1-217:/var/www/html# vi index.html
root@ip-10-1-1-217:/var/www/html# systemctl restart apache2
root@ip-10-1-1-217:/var/www/html# curl 10.1.1.217:80
hello this is from webserver 2
root@ip-10-1-1-217:/var/www/html# curl 10.0.1.248:80
hello this is from website1
root@ip-10-1-1-217:/var/www/html#
```

- As shown in above slide we can able to connect from one instance to another so vpc peering was successfully done