

Terraform

Introduction

1) IAC

- 1) IAC (Infrastructure as a code) IAC is a widespread terminology among Devops professionals.
 - 2) It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine readable definition files.
 - 3) It is a software engineering approach towards operation.
 - 4) It helps in automating the complete data enter by using programming scripts.
-

2) Infrastructure as a code has multiple challenges

Need to learn to code.

Don't know the change impact

Need to revert the change.

Cannot track changes.

Cannot automate a resource

Multiple environments for infrastructure.

Terraform has been created to solve these challenges.

3) Terraform is an open-source infrastructure as code software tool developed by HashiCorp. It is used to define and provision the complete infrastructure using easy to learn declarative languages.

Terraform v1.3.6 9 (Version)

Command – `terraform --version`

4) Benefits of using terraform:

- 1) Supports multiple providers such as AWS, Azure, GCP, digital motions and etc.
 - 2) Provides immutable infrastructure where configuration changes smoothly
 - 3) Uses easy to understand language HCL (HashiCorp configuration language)
 - 4) Easily portable to any other provider.
 - 5) Supports client only architecture. So, no need of additional configuration management on a server.
-

5) Terraform core concepts:

Below are the core concepts/terminologies used in Terraform:

- 1) **Variables:** Also used as input-variables, it is key-value pair used by Terraform modules to allow customization.
- 2) **Provider:** A provider in Terraform is a plugin that enables interaction with an API (Application Programming Interface)
- 3) **Module:** It is a folder with Terraform templates where all the configurations are defined.
- 4) **State:** It consists of cached information about the infrastructure managed by

Terraform and the related configurations.

5) Resources: It refers to a block of one or more infrastructure objects (compute instances, virtual networks, etc.), which are used in configuring and managing the infrastructure.

6) Data Source: It is implemented by providers to return information on external objects to terraform.

7) Output Values: These are return values of a terraform module that can be used by other configurations.

6) Lifecycle

1) Terraform init : which initializes the working directory which consists of all the configuration files.

2) Terraform Plan is used to create and execute the plan to reach desire state of the infrastructure. Changes in the configuration files are done in order to achieve the desired state.

3) Terraform Apply : The Terraform apply command executes changes to the actual environment. Apply will look for a Terraform configuration and create an execution plan based on the desired state. By default it should be executed after Terraform plan

4) Terraform destroy command is the opposite to Terraform apply, in which it terminates all the resources specified in Terraform state.

5) Terraform core performing all API calls to the providers (Access key or roles from IAM)

7) Difference between Ansible and Terraform

Ansible is an multi-purpose automation tool, whereas Terraform is an infrastructure as code tool.

Steps:

1) Installation - <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>

Follow the link steps to install Terraform

Install Terraform

Manual installation

Homebrew on OS X

Chocolatey on Windows

Linux

2) Create terraform folder

3) cd terraform and terraform init (commands)

4) create a user in AWS (IAM) by clicking on add user

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name* myterraformuser

+ Add another user

Select AWS access type

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from accessing the console using an assumed role. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Select AWS credential type* ☒ Access key - Programmatic access
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☒ Password - AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

Console password* ☒ Autogenerated password
☐ Custom password

5) Give Administrator access, VPC full access, EC2 access

Filter policies		Q ad	Showing 234 results	
	Policy name	Type	Used as	
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	None	
<input checked="" type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	None	
<input checked="" type="checkbox"/>	AdministratorAccess-AWSElasticBeanstalk	AWS managed	None	

Filter policies		Q vpc	Showing 7 results	
	Policy name	Type	Used as	
<input type="checkbox"/>	AmazonDMSVPCManagementRole	AWS managed	None	
<input type="checkbox"/>	AmazonDRSVPCManagement	AWS managed	None	
<input type="checkbox"/>	AmazonEKSVPCResourceController	AWS managed	None	
<input type="checkbox"/>	AmazonVPCCrossAccountNetworkInterfaceOperations	AWS managed	None	
<input checked="" type="checkbox"/>	AmazonVPCFullAccess	AWS managed	None	
<input type="checkbox"/>	AmazonVPCReadOnlyAccess	AWS managed	None	

Filter policies		Q EC2	Showing 28 results	
	Policy name	Type	Used as	
<input type="checkbox"/>	AmazonEC2ContainerRegistryFullAccess	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerRegistryPowerUser	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerRegistryReadOnly	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerServiceAutoscaleRole	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerServiceEventsRole	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerServiceforEC2Role	AWS managed	None	
<input type="checkbox"/>	AmazonEC2ContainerServiceRole	AWS managed	None	
<input checked="" type="checkbox"/>	AmazonEC2FullAccess	AWS managed	None	

6) Excel file will get download for username and password

Add user

1 2 3 4 **5**



Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://527173071048.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key	Password	Email login instructions
▶	✔ myterraformu...	AKIAVPPWWDEHCSEXOT7	***** Show	***** Show	Send email

2) Steps - To create IAM role

1) Link for Terraform:

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs> - Full details

Note: In IAM allow IAMAllow permission and attach to the instances

Steps:

1) In IAM click on myterraform

Identity and Access Management (IAM) x

Search IAM

Dashboard

▼ Access management

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

▼ Access reports

- Access analyzer
- Archive rules
- Analizers

IAM > Roles

Roles (Selected 1/7) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Search

< 1 > ⚙

	Role name	Trusted entities	Last action
<input type="checkbox"/>	aws-ec2-spot-fleet-tagging-role	AWS Service: spotfleet	-
<input type="checkbox"/>	AWSServiceRoleForAmazonElasticFileSystem	AWS Service: elasticfilesystem (Service-Linked Role)	8 days ago
<input type="checkbox"/>	AWSServiceRoleForBackup	AWS Service: backup (Service-Linked Role)	4 hours ago
<input type="checkbox"/>	AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
<input type="checkbox"/>	AWSServiceRoleForVPCTransitGateway	AWS Service: transitgateway (Service-Linked Role)	11 days ago
<input checked="" type="checkbox"/>	myroterraform	AWS Service: ec2	44 minutes ago

2) Click on add permissions and Attach Policies

Later Add permission and click on attach

Permissions policies (4) [Info](#)

You can attach up to 10 managed policies.

[Refresh](#) [Simulate](#) [Remove](#) [Add permissions](#)

Filter policies by property or policy name and press enter.

<input type="checkbox"/>	Policy name ↗	Type	Description
<input type="checkbox"/>	+ AmazonEC2FullAccess	AWS manag...	Provides full access to Amazon EC2 via the AWS Management Co...
<input type="checkbox"/>	+ IAMFullAccess	AWS manag...	Provides full access to IAM via the AWS Management Console.
<input type="checkbox"/>	+ AmazonS3FullAccess	AWS manag...	Provides full access to all buckets via the AWS Management Cons...
<input type="checkbox"/>	+ AmazonVPCFullAcce...	AWS manag...	Provides full access to Amazon VPC via the AWS Management Co...

3)

EC2 > Instances > i-0128d40cdcca20481 > Modify IAM role

Modify IAM role [Info](#)

Attach an IAM role to your instance.

Instance ID

[i-0128d40cdcca20481](#) (Slave2 Terraform)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

[Create new IAM role](#)

[Cancel](#) [Update IAM role](#)

3) Steps

If want to use other providers than I need to use **alias**

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "ap-south-1"
}

provider "aws" {
  alias   = "tokyo"
  region = "ap-northeast-1"
}
```

```
resource "aws_vpc" "myvpc" {
  cidr_block      = var.vpc_cidr_block
  enable_dns_hostnames = true
  enable_dns_support = true
  tags = {
    Name = "myvpc"
  }
}

resource "aws_vpc" "myvpc2" {
  provider = aws.tokyo
  cidr_block = var.vpc_cidr_block
  tags = {
    Name = "myvpc2"
  }
}
```

4) Steps

1) Providers

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "ap-south-1"
}
```

2) To Create VPC. Variables are saved in variable tap

```
resource "aws_vpc" "myvpc" {
  cidr_block      = var.vpc_cidr_block
  enable_dns_hostnames = true
  enable_dns_support = true
  tags = {
    Name = "myvpc"
  }
}
```

Variables

```
variable "vpc_cidr_block" {
  default = "200.20.60.0/24"
}
```

3) To Create Internet Gateway, subnet, route table, route and associates route table. Variables is saved in variable tap

```
resource "aws_internet_gateway" "ig" {
  vpc_id = aws_vpc.myvpc.id
  tags = {
    Name = "myig"
  }
}
```

```
resource "aws_subnet" "pub_subnet" {
  vpc_id      = aws_vpc.myvpc.id
  cidr_block = var.pub_subnet
  tags = {
    Name = "pub_subnet"
  }
}
```

Variables

```
variable "pub_subnet" {
  default = "200.20.60.0/25"
}
```

```
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.myvpc.id
  tags = {
    Name = "mypub_route"
  }
}
```

```
resource "aws_route" "public_internet_gateway" {
  route_table_id      = aws_route_table.public.id
  destination_cidr_block = var.destination
  gateway_id          = aws_internet_gateway.ig.id
}
```

Variables

```
variable "destination" {
  default = "0.0.0.0/0"
}
```

```
resource "aws_route_table_association" "public" {
  subnet_id      = aws_subnet.pub_subnet.id
  route_table_id = aws_route_table.public.id
}
```

4) To Create NAT Gateway, subnet, route table, route and associates route table.

Variables is saved in variable tap

```
resource "aws_nat_gateway" "nat" {
  connectivity_type = "private"
  subnet_id        = aws_subnet.pub_subnet.id
}
```

```
resource "aws_subnet" "pvt_subnet" {
  vpc_id      = aws_vpc.myvpc.id
  cidr_block = var.pvt_subnet
  tags = {
    Name = "private_subnet"
  }
}
```

Variables

```
variable "pvt_subnet" {
  default = "200.20.60.128/25"
}
```

```
resource "aws_route_table" "private" {
  vpc_id = aws_vpc.myvpc.id
  tags = {
    Name = "mypvt_route"
  }
}
```

```
resource "aws_route" "private_nat_gateway" {
  route_table_id      = aws_route_table.private.id
  destination_cidr_block = var.destination
  nat_gateway_id      = aws_nat_gateway.nat.id
}
```

Variables

```
variable "destination" {
  default = "0.0.0.0/0"
}
```

```
resource "aws_route_table_association" "private" {
  subnet_id      = aws_subnet.pvt_subnet.id
  route_table_id = aws_route_table.private.id
}
```

5) Creating Instances

sg - Name given for Security Group

```
resource "aws_instance" "ec2_public" {
  count                = var.instance_count
  ami                 = var.amis[var.region]
  associate_public_ip_address = true
  instance_type       = var.instance
  key_name             = var.key
  subnet_id           = aws_subnet.pub_subnet.id
  vpc_security_group_ids = ["${aws_security_group.sg.id}"]
  tags = {
    "Name" = "yetish_instance"
  }
}
```

```
variable "instance_count" {
  default = "3"
}
variable "instance" {
  default = "t2.micro"
}
variable "key" {
  default = "Mumbai_Yetish"
}
```



```
variable "amis" {
  type = map
  default = {
    "ap-northeast-1" = "ami-0590f3a1742b17914"
    "ap-south-1" = "ami-07ffb2f4d65357b42"
  }
}
variable "region" {
  default = "ap-south-1"
}
```

6) Adding Security groups

Ingress is inbound

egress is outbound

sg - Name given for Security Group

```
resource "aws_security_group" "sg" {
  vpc_id = aws_vpc.myvpc.id
  ingress {
    from_port = "0"
    to_port   = "0"
    protocol  = "-1"
    self      = true
  }

  egress {
    from_port = "0"
    to_port   = "0"
    protocol  = "-1"
    self      = "true"
  }
  tags = {
    "Name" = "My Security_group"
  }
}
```

Later

Terraform Plan

Terraform apply

Terraform destroy

4) Terraform State File : The state file is used by Terraform to keep track of resources information about the infrastructure **or** Statefile will have current state of the infrastructure created from the Terraform.

Subcommands:	
list	List resources in the state
mv	Move an item in the state
pull	Pull current state and output to stdout
push	Update remote state from a local state file
replace-provider	Replace provider in the state
rm	Remove instances from the state
show	Show a resource in the state

State file will have existing infrastructure (**Note:** never modify the state file, that

holds the current infrastructure)

Question: Have u modified state file?

Ans: No, terraform state file should not modify in general, we have commands to replace and remove the resources

Commands

1) terraform state list - The terraform state list command is used to list resources within a Terraform state.

```
root@ip-172-31-44-245:~/provisioner# terraform state list
aws_instance.ys-instance
aws_security_group.zlsecurity
null_resource.my_null
```

2) terraform state show aws_vpc.myvpc - The Terraform state show command is used to show the attributes of a single resource in the Terraform state.

```
root@ip-172-31-44-245:~/provisioner# terraform state show null_resource.my_null
# null_resource.my_null:
resource "null_resource" "my_null" {
  id = "6771193588441867380"
}
```

(aws_vpc.myvpc - resources name)

3) terraform state rm aws_vpc.myvpc - The Terraform state rm command is used to remove items from the Terraform state.

(aws_vpc.myvpc - resources name)

4) terraform fmt filename - will format the file

5) In order to destroy specific resource

a) Command: terraform state list

b) Command: Terraform destroy --target resource name

6) Command: when = destroy - command will execute, whenever we are destroying the particular resource.

5) To create IAM role

Note: In IAM allow **IAMAllow permission** and attach to the instances

Steps:

1) In IAM click on myterraform

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

Access reports

- Access analyzer
- Archive rules
- Analyzers

IAM > Roles

Roles (Selected 1/7) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Search

< 1 >

	Role name	Trusted entities	Last acti...
<input type="checkbox"/>	aws-ec2-spot-fleet-tagging-role	AWS Service: spotfleet	-
<input type="checkbox"/>	AWSServiceRoleForAmazonElasticFileSystem	AWS Service: elasticfilesystem (Service-Linked Role)	8 days ago
<input type="checkbox"/>	AWSServiceRoleForBackup	AWS Service: backup (Service-Linked Role)	4 hours ago
<input type="checkbox"/>	AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
<input type="checkbox"/>	AWSServiceRoleForVPCTransitGateway	AWS Service: transitgateway (Service-Linked Role)	11 days ago
<input checked="" type="checkbox"/>	myroleterraform	AWS Service: ec2	44 minutes ago

2) Click on add permissions and Attach Policies
Later Add permission and click on attach

Permissions policies (4) Info

You can attach up to 10 managed policies.

Simulate

Remove

Add permissions

Filter policies by property or policy name and press enter.

< 1 >

	Policy name	Type	Description
<input type="checkbox"/>	AmazonEC2FullAccess	AWS manag...	Provides full access to Amazon EC2 via the AWS Management Co...
<input type="checkbox"/>	IAMFullAccess	AWS manag...	Provides full access to IAM via the AWS Management Console.
<input type="checkbox"/>	AmazonS3FullAccess	AWS manag...	Provides full access to all buckets via the AWS Management Cons...
<input type="checkbox"/>	AmazonVPCFullAcce...	AWS manag...	Provides full access to Amazon VPC via the AWS Management Co...

3)

```

#Create an IAM Role
resource "aws_iam_role" "yetish" {
  name = "ec2_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = "RoleForEC2"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      },
    ]
  })
}

resource "aws_iam_policy_attachment" "yetish_attach" {
  name      = "yetish_attachment"
  roles     = [aws_iam_role.yetish.name]
  policy_arn = aws_iam_policy.policy.arn
}

resource "aws_iam_instance_profile" "usr_yetish" {
}

resource "aws_iam_instance_profile" "usr_yetish" {
  name = "usr_yetish"
  role = aws_iam_role.yetish.name
}

resource "aws_instance" "import_Terraform1" {
  ami = "unknown"
  instance_type = "unkonwn"
}

resource "aws_instance" "import_Terraform2" {
  ami = "unknown"
  instance_type = "unkonwn"
}

```

4) terraform plan

5) terraform apply

6) In Instances

1) Click on actions and security


Later roles will be attached

EC2 > Instances > i-0128d40cdcca20481 > Modify IAM role

Modify IAM role [Info](#)

Attach an IAM role to your instance.


Instance ID

 i-0128d40cdcca20481 (Slave2 Terraform)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

▼


[Create new IAM role](#)

Cancel
Update IAM role

7) terraform destroy

6) Terraform Taint – used to destroy and recreate the resource (If the resource as been created long back or resource changed due to some issue, if I want to recreate the instances) than I will use **Taint command**

Command – Terraform taint resource name

1) Question: I want restart the resources, which are created?

Answer: First we will list and use taint command

- terraform state list
- terraform taint resource name
- terraform Plan
- terraform Apply

```
root@ip-172-31-44-245:~/provisioner# terraform taint null_resource.my_null
Resource instance null_resource.my_null has been marked as tainted.
root@ip-172-31-44-245:~/provisioner# terraform plan
aws_security_group.zlsecurity: Refreshing state... [id=sg-0868b532623b00052]
aws_instance.ys-instance: Refreshing state... [id=i-06394d9fe66c080e6]
null_resource.my_null: Refreshing state... [id=6771193588441867380]
root@ip-172-31-44-245:~/provisioner# terraform apply
aws_security_group.zlsecurity: Refreshing state... [id=sg-0868b532623b00052]
aws_instance.ys-instance: Refreshing state... [id=i-06394d9fe66c080e6]
null_resource.my_null: Refreshing state... [id=6771193588441867380]

Terraform used the selected providers to generate the following execution plan
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # null_resource.my_null is tainted, so must be replaced
-/+ resource "null_resource" "my_null" {
    ~ id = "6771193588441867380" -> (known after apply)
  }

Plan: 1 to add, 0 to change, 1 to destroy.
```

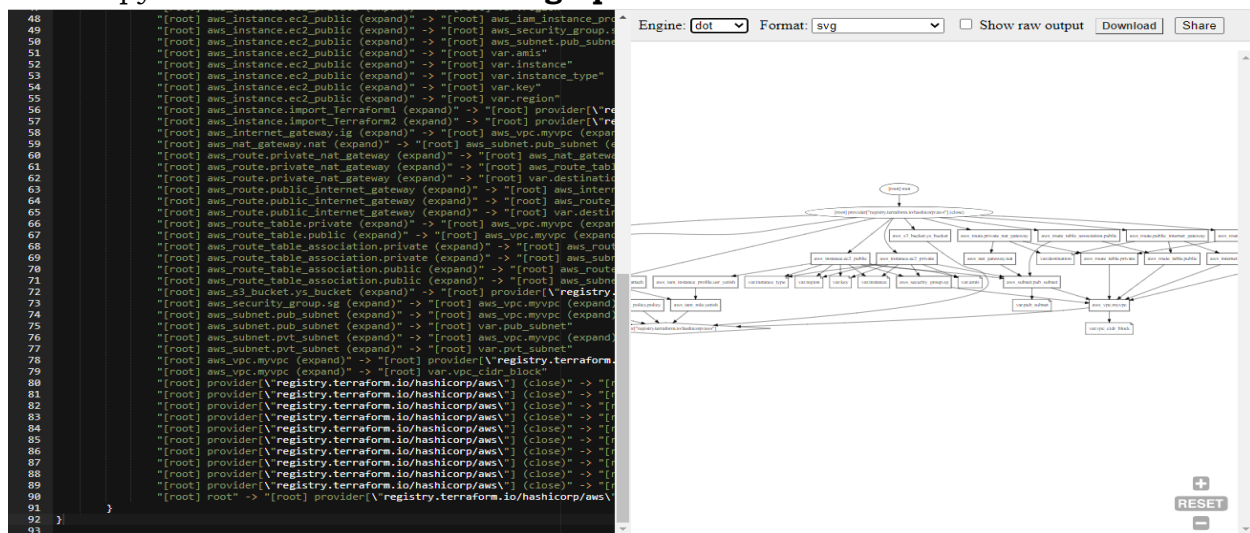
7) Terraform Graph – What all we create will be in form of graph.

Command - terraform graph

Link -

https://dreampuf.github.io/GraphvizOnline/#digraph%20G%20%7B%0A%0A%20%20subgraph%20cluster_0%20%7B%0A%20%20%20%20style%3Dfilled%3B%0A%20%20%20color%3Dlightgrey%3B%0A%20%20%20%20node%20%20%5Bstyle%3Dfilled%20Ccolor%3Dwhite%5D%3B%0A%20%20%20%20a0%20-%3E%20a1%20-%3E%20a2%20-%3E%20a3%3B%0A%20%20%20%20label%20%3D%20%22process%20%231%22%3B%0A%20%20%20%20%7D%0A%0A%20%20subgraph%20cluster_1%20%7B%0A%20%20%20%20node%20%20%5Bstyle%3Dfilled%5D%3B%0A%20%20%20%20b0%20-%3E%20b1%20-%3E%20b2%20-%3E%20b3%3B%0A%20%20%20%20%20

Later copy the contents in **terraform graph** to above mentioned link



8) Terraform Import – it is used to bring existing infrastructure to Terraform.

Command - terraform import aws_instance.import_Terraform1 i-07dcebf96350f76e4

1) **Terraform 1** is the instances

2) **i-07dcebf96350f76e4** is ID of the instances

```
resource "aws_instance" "import_Terraform2" {
  ami = "unknown"
  instance_type = "unknown"
}

#Create an IAM Role
resource "aws_s3_bucket" "ys_bucket" {
  bucket = "mys3-bucket113"
```

```
root@ip-172-31-44-245:~/myterraform# terraform import aws_instance.import_Terraform1 i-07dcebf96350f76e4
aws_instance.import_Terraform1: Importing from ID "i-07dcebf96350f76e4"...
aws_instance.import_Terraform1: Import prepared!
  Prepared aws_instance for import
aws_instance.import_Terraform1: Refreshing state... [id=i-07dcebf96350f76e4]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

9) Terraform output - Manage sensitive data in state file. Output used to extract the value from the state file.

10) Module: It is a folder with Terraform templates where all the configurations are defined.

Steps:

1) Create a new folder **new_module** and move **main.tf, provider.tf** and **variable.tf** files

```
root@ip-172-31-44-245:~/myterraform# cd new_module/  
root@ip-172-31-44-245:~/myterraform/new_module# ls  
main.tf  provider.tf  variable.tf
```

2) create **main.tf** file for module

```
module "services" {  
    source = "./new_module"  
}
```

3) Terraform init

4) Terraform plan

5) Terraform apply

6) Terraform destroy

11) Create Tomcat using Terraform

Steps:

1) Create tomcat folder

2) create **main.tf** file (In 6 lines are instance details)

```
resource "aws_instance" "my-instance" {  
    ami = "ami-0ecc74ecald66d8a6"  
    instance_type = "t2.micro"  
    key_name = "test_terraform"  
    vpc_security_group_ids = [aws_security_group.z1security.id]  
    user_data = <<-EOF
```

Shell script for tomcat

```

#!/bin/bash
sudo apt update -y
sudo apt install default-jre -y
sudo wget https://dlcdn.apache.org/tomcat/tomcat-10/v10.0.27/bin/apache-tomcat-10.0.27.tar.gz
sudo tar -xvzf apache-tomcat-10.0.27.tar.gz
sudo rm -rf apache-tomcat-10.0.27.tar.gz
sudo mv apache-tomcat-10.0.27 tomcat
sudo sh tomcat/bin/startup.sh
sudo rm -rf conf-and-webapps-file
sudo git clone https://github.com/syedwaliuddin/conf-and-webapps-file.git
sudo rm -rf tomcat/conf/tomcat-users.xml
sudo cp conf-and-webapps-file/tomcat-users.xml tomcat/conf/
sudo sh tomcat/bin/startup.sh
sudo rm -rf tomcat/webapps/manager/META-INF/context.xml
sudo cp conf-and-webapps-file/context.xml tomcat/webapps/manager/META-INF/
sudo rm -rf tomcat/webapps/host-manager/META-INF/context.xml
sudo cp conf-and-webapps-file/contextthm.xml tomcat/webapps/host-manager/META-INF/
sudo sh tomcat/bin/startup.sh
EOF
tags = {
  Name = "Tomcat"
}
}

```

3) Create **security.tf** file (From and to traffic is 0)

```

resource "aws_security_group" "z1security" {
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

```

- 4) Terraform init
- 5) Terraform plan
- 6) terraform apply
- 7) New Instance will be created, so we can launch Tomcat and Jenkins
- 8) Terraform destroy

12) Create Jenkins using Terraform

Steps:

- 1) Create jenkins folder
- 2) create **main.tf** file (In 6 lines are instance details)

Shell script for Jenkins


```

resource "aws_instance" "my-instance" {
  ami = "ami-0ecc74ecald66d8a6"
  instance_type = "t2.micro"
  key_name = "test_terraform"
  vpc_security_group_ids = [aws_security_group.zlsecurity.id]
  user_data = <<EOF
sudo apt update
sudo apt install -y openjdk-11-jdk
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt update
sudo apt -y install jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins
EOF
  tags = {
    Name = "Jenkins"
  }
}

```

3) Create security.tf file (From and to traffic is 0)

```

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

```

- 4) Terraform init
- 5) Terraform plan
- 6) terraform apply
- 7) New Instance will be created, so we can launch Tomcat and Jenkins
- 8) Terraform destroy

13) Terraform Provisioners

When a resource is created, we may have some scripts or operations that need to be performed locally, so at this time Terraform provisioners are used.

There are two types in provisioners

- 1) local-exec provisioner
- 2) remote-exec provisioner

1) local-exec provisioner

(own words)

we have created a instances or resources, if I want to print a private or public IP address of the instances. Private IP will create in local machine.

```

provisioner "local-exec" {
  command = "echo ${aws_instance.my-instance.private_ip} >> privateip.txt"
}

```

```
resource "aws_instance" "ys-instance" {
  ami = "ami-07ffb2f4d65357b42"
  instance_type = "t2.micro"
  key_name = "Mumbai_Yetish1"
  vpc_security_group_ids = [aws_security_group.z1security.id]
  provisioner "local-exec" {
    when = destroy
    command = "echo ${self.private_ip} >> privateip.txt"
  }
}
```

Result: Privateip.txt

```
root@ip-172-31-44-245:~/provisioner# cat privateip.txt
172.31.47.66
172.31.40.45
172.31.43.219
```

when = destroy - command will execute, whenever we are destroying the particular resource.

2) remote-exec provisioner

remote-exec provisioner - remote-exec provisioner invokes a script on a remote resource after it is created.

we need connection block (its need to be connect to the remote server) and pem key (terraform.pem) should be used.

```
resource "aws_instance" "ys-instance" {
  ami = "ami-07ffb2f4d65357b42"
  instance_type = "t2.micro"
  key_name = "Mumbai_Yetish1"
  vpc_security_group_ids = [aws_security_group.z1security.id]

  connection {
    type      = "ssh"
    user      = "ubuntu"
    private_key = file("./terraform.pem")
    host      = self.public_ip
  }
  provisioner "remote-exec" {
    inline = [
      "sudo apt update",
    ]
  }
}
```

3) Null_resource terraform

Null resource help to execute the provisioner (It will work only instances is up and running).

```
resource "aws_instance" "ys-instance" {
  ami = "ami-07ffb2f4d65357b42"
  instance_type = "t2.micro"
  key_name = "Mumbai_Yetish1"
  vpc_security_group_ids = [aws_security_group.z1security.id]
```

```
resource "null_resource" "my_null" {
  connection {
    type      = "ssh"
    user      = "ubuntu"
    private_key = file("./terraform.pem")
    host      = aws_instance.ys-instance.public_ip
  }
}
```

14) terraform.tfstate

It contains all sensitive information (ID of resource and other details), we cannot store in Github.

All the source code is stored in Github and **state file can be stored s3 bucket.**

Steps:

1) Create s3 bucket

The screenshot shows the Amazon S3 console 'Buckets' page. At the top, there's an 'Account snapshot' section with a 'View Storage Lens dashboard' button. Below that, the 'Buckets (1)' section shows a table with one bucket named 'yetishsss' in the 'Asia Pacific (Mumbai) ap-south-1' region. The access is 'Bucket and objects not public', and it was created on 'January 2, 2023, 11:34:27 (UTC+05:30)'. Action buttons like 'Copy ARN', 'Empty', 'Delete', and 'Create bucket' are visible.

2) Create a folder in yetishsss

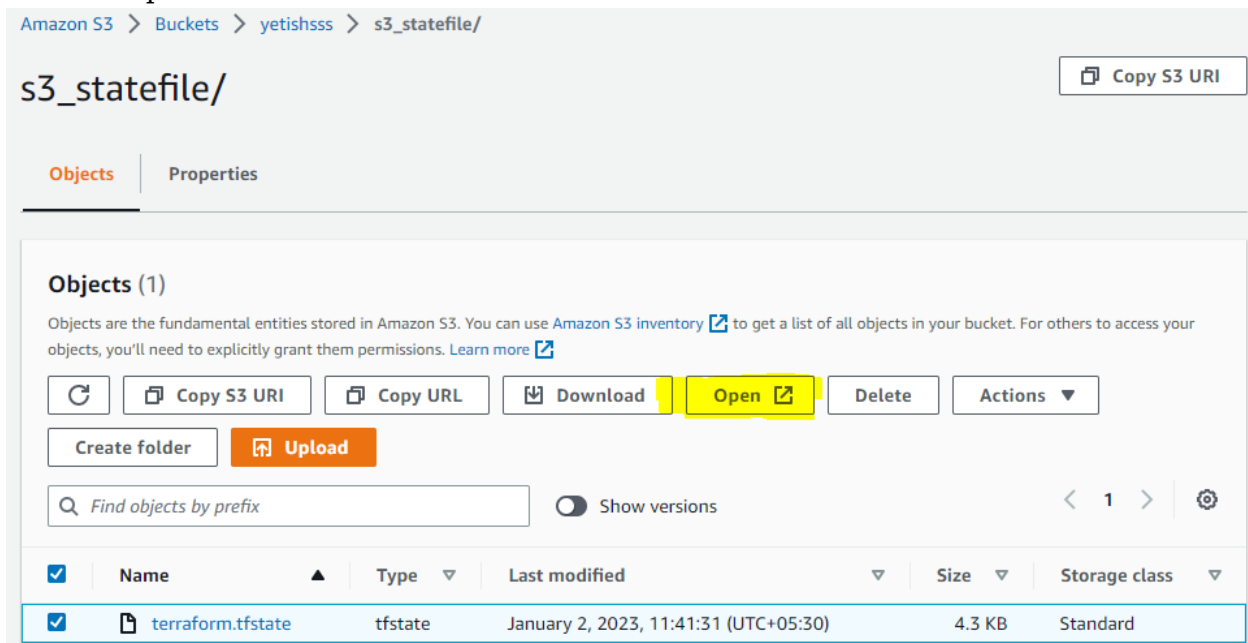
The screenshot shows the 'yetishsss' bucket page in the Amazon S3 console. The 'Objects' tab is selected. It shows 'Objects (1)' with a list containing a folder named 's3_statefile/'. Above the list, there are buttons for 'Create folder' and 'Upload'. The console also shows various action buttons like 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', and 'Actions'.

3) In instances under provider.tf.
mention below command

```
terraform {  
  backend "s3" {  
    encrypt = false  
    bucket  = "yetishsss"  
    region  = "ap-south-1"  
    key     = "s3_statefile/terraform.tfstate"  
  }  
}
```

4) Terraform init
Terraform plan
Terraform apply

5) In S3 bucket Terraform.tfstate will be available (protected)
Click on open



6) Commands (Pull and Push)

Delete the state file in local machine

1) **Command: terraform state pull > terraform.ftstate** – pulling s3 bucket to our machine.

output should be redirect to new terraform.ftstate

2) **Command: terraform state push filename** – push to s3 bucket
filename (terraform.ftstate)

15) Locals

A local value assigns a name to an expression, so you can use the name multiple times within a module instead of repeating the expression.

Example: key_name="Mumbai_Yetish"

```

locals {
  key_name="Mumbai_Yetish"
}
module "ec2_instance" {
  source = "../new_module"
  region = "ap-south-1"
  key_name = "${local.key_name}"
  instance_type = "t2.micro"
}

```

16) Workspace – we will be creating multiple environment for our infrastructure, if want to test, before going to the production I will do in the local environment.

Commands

1) terraform workspace show

```

root@ip-172-31-44-245:~/s3bucket# terraform workspace show
default

```

2) To create new workspace

a) terraform workspace new qa

b) terraform workspace new prod

c) terraform workspace new dev

3) To check all workspace

a) terraform workspace list

```

root@ip-172-31-44-245:~/provisioner# terraform workspace list
default
* dev
  prod
  qa

```

4) To switch to other workspace

a) terraform workspace select prod

```

root@ip-172-31-44-245:~/provisioner# terraform workspace select prod
Switched to workspace "prod".
root@ip-172-31-44-245:~/provisioner# terraform workspace list
default
  dev
* prod
  qa

```

Note: Every workspace (**qa, dev, prod and default**) state file will be different, so this is the reason we will save in s3 bucket.

```

provider "aws" {
  region="ap-south-1"
}

locals {
  env="${terraform.workspace}"

  counts = {
    "default"=1
    "prod"=3
    "dev"=2
  }

  instances = {
    "default"="t2.micro"
    "prod"="t2.small"
    "dev"="t2.micro"
  }

```

```

  tags = {
    "default"="default"
    "prod"="prod"
    "dev"="dev"
  }

  instance_type="${lookup(local.instances,local.env) }"
  count="${lookup(local.counts,local.env) }"
  mytag="${lookup(local.tags,local.env) }"
}

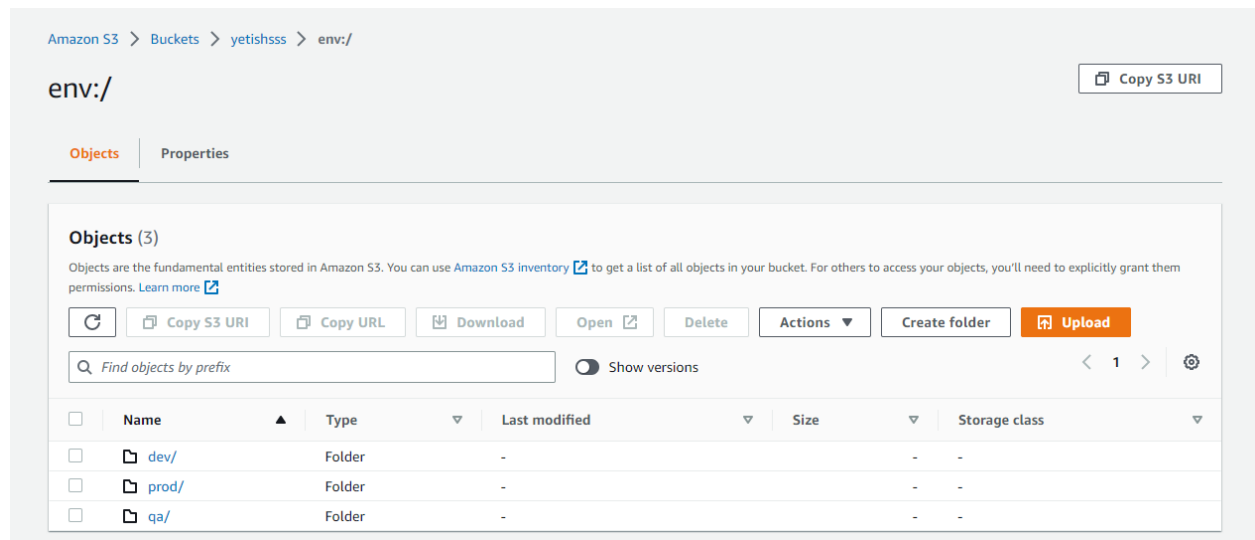
```

```

resource "aws_instance" "ys_instance" {
  ami="ami-0447a12f28fddb066"
  instance_type="${local.instance_type}"
  count="${local.count}"
  tags = {
    Name="${local.mytag}"
  }
}

```

later switch to other workspace and run 1) terraform init 2) terraform plan 3) terraform apply and later terraform destroy



17) State File Locking – (to protect the statefile from getting corrupted from writing)

Whenever you are performing write operation, Terraform would lock the state file. This is very important as otherwise during your ongoing Terraform apply operations, if others also try for the same, it can corrupt state file.