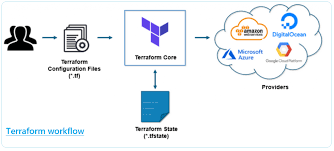
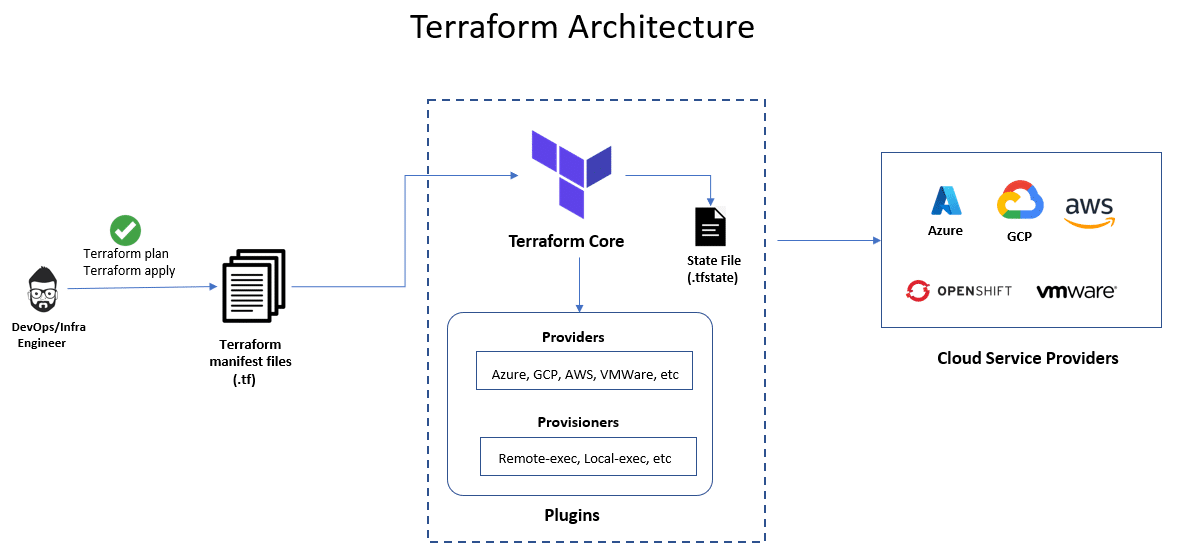
Terraform

**➤ What is Terraform?**

Terraform is an **open-source** tool that **helps implement IaC**.

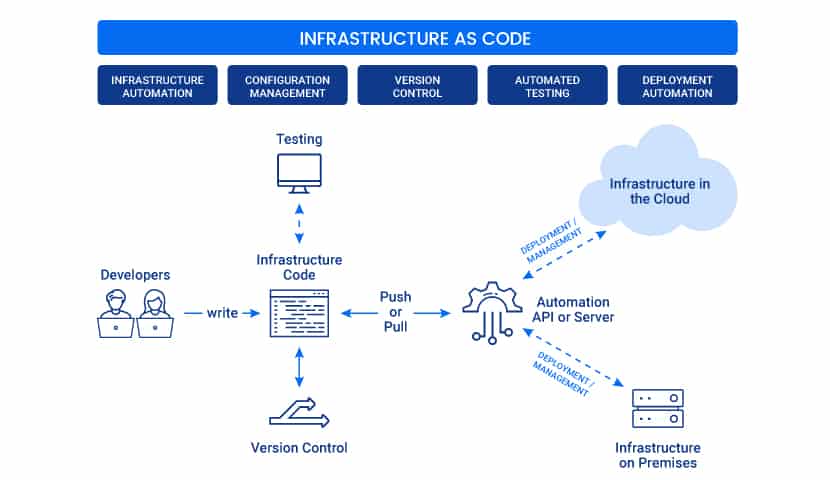
It allows you to ***write code***in a specific language called **HashiCorp Configuration Language** (HCL) to describe the desired state of your infrastructure.





## **What Is Infrastructure as Code (IaC)?**

**Infrastructure as Code (IaC)** is a widespread terminology among [DevOps](https://k21academy.com/devops-foundation/overview-devops-devops-principles-advantages-of-devops-why-devops/) professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine-readable definition files. It helps in automating the complete data center by using programming scripts.



### **Popular IaC Tools:**

**1. Terraform** An open-source declarative tool that offers pre-written modules to build and manage an infrastructure.  
**2. Chef:** A configuration management tool that uses cookbooks and recipes to deploy the desired environment. Best used for Deploying and configuring applications using a pull-based approach.  
**3. Puppet:** Popular tool for configuration management that follows a Client-Server Model. Puppet needs agents to be deployed on the target machines before the puppet can start managing them.  
**4. Ansible:** Ansible is used for building infrastructure as well as deploying and configuring applications on top of them. Best used for Ad hoc analysis.  
**5. Packer:** Unique tool that generates VM images (not running VMs) based on steps you provide. Best used for Baking compute images.  
**6. Vagrant:** Builds VMs using a workflow. Best used for Creating pre-configured developer VMs within VirtualBox.

Read our blog to know why Terraform is preferred over other IaC tools [**Terraform vs Ansible**](https://k21academy.com/ansible/terraform-vs-ansible/)

## **What Is Terraform?**

**Terraform** is one of the most popular **Infrastructure-as-code (IaC) tool**, used by DevOps teams to automate infrastructure tasks. It is used to automate the provisioning of your cloud resources. Terraform is an open-source, cloud-agnostic provisioning tool developed by HashiCorp and written in GO language.



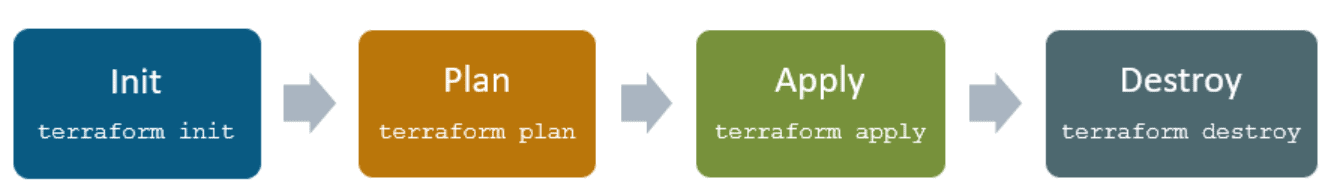
**Benefits of using Terraform:**

* Does orchestration, not just configuration management
* Supports multiple providers such as AWS, Azure, Oracle, GCP, and many more
* Provide immutable infrastructure where configuration changes smoothly
* Uses easy to understand language, HCL (HashiCorp configuration language)
* Easily portable to any other provider

Check out our blog for everything you need to know about Terraform Certification [**Terraform Certification**](https://k21academy.com/terraform-iac/hashicorp-infrastructure-automation-certification-terraform-associate/)

## **Terraform Lifecycle**

Terraform lifecycle consists of – **init**, **plan**, **apply**, and **destroy**.



1. **Terraform init** initializes the (local) Terraform environment. Usually executed only once per session.  
2. **Terraform plan** compares the Terraform state with the as-is state in the cloud, build and display an  
execution plan. This does not change the deployment (read-only).  
3. **Terraform apply** executes the plan. This potentially changes the deployment.  
4. **Terraform destroy** deletes all resources that are governed by this specific terraform environment.

## **Terraform Core Concepts**

**1. Variables**: Terraform has input and output variables, it is a key-value pair. Input variables are used as parameters to input values at run time to customize our deployments. Output variables are return values of a terraform module that can be used by other configurations.  
Read our blog on [**Terraform Variables**](https://k21academy.com/terraform-iac/variables-in-terraform/)

**2. Provider**: Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A provider is a plugin that interacts with the various APIs required to create, update, and delete various resources.  
Read our blog to know more about [**Terraform Providers**](https://k21academy.com/terraform-iac/terraform-providers-overview/)

**3. Module**: Any set of Terraform configuration files in a folder is a module. Every Terraform configuration has at least one module, known as its **root module.**

**4. State**: Terraform records information about what infrastructure is created in a Terraform state file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.

**5. Resources**: Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.

**6. Data Source**: Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.

**7. Plan**: It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.

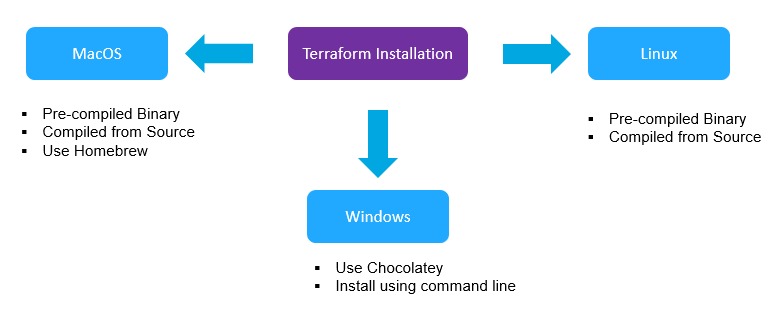
**8. Apply**: It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.

**Check Out:**Our previous blog post on [**Terraform Cheat Sheet**](https://k21academy.com/terraform-iac/terraform-cheat-sheet/).

## **Terraform Installation**

Before you start working, make sure you have Terraform installed on your machine, it can be installed on any OS, say Windows, macOS, Linux, or others. Terraform installation is an easy process and can be done in a few minutes.

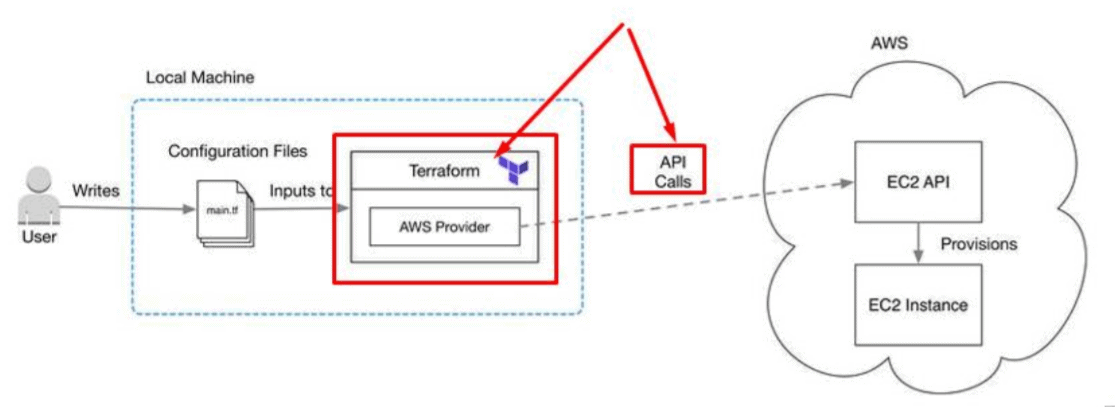
Read our blog to know how to [install Terraform](https://k21academy.com/terraform-iac/terraform-installation-overview/) in Linux, Mac, Windows



We cover the step-by-step Terraform installation in all these ways in our Terraform training. Check out our blog for all the Hands-on Labs that we cover in our training [HashiCorp Certified Terraform Associate](https://k21academy.com/terraform-iac/hashicorp-certified-terraform-associate-step-by-step-activity-guides/" \t "_blank)-Step By Step Activity Guides.

## **Terraform Providers**

A provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with the API of the service. Terraform configurations must declare which providers they require so that Terraform can install and use them.

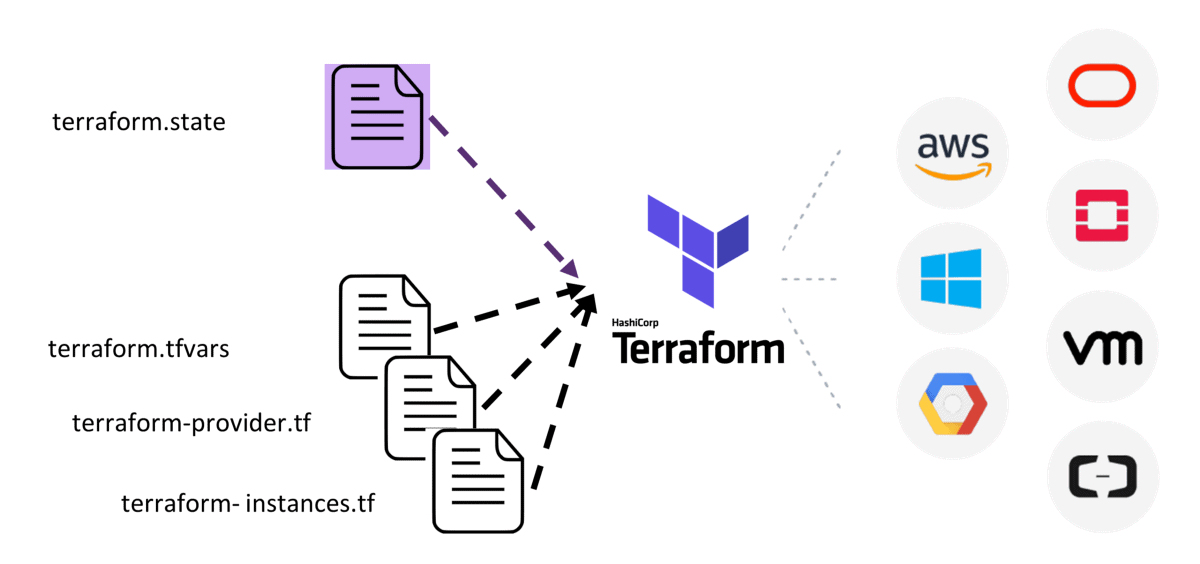


Terraform has over a hundred providers for different technologies, and each provider then gives terraform user access to its resources. So through AWS provider, for example, you have access to hundreds of AWS resources like EC2 instances, the AWS users, etc.

**Read More:**About [**Terraform Workflow**](https://k21academy.com/terraform-iac/terraform-workflow-and-its-use-case/).

## **Terraform Configuration Files**

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions **.tf** and **.tf.json**. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.



A Terraform configuration is made up of one or more files in a directory, provider binaries, plan files, and state files once Terraform has run the configuration.

**1. Configuration file (\*.tf files):** Here we declare the provider and resources to be deployed along with the type of resource and all resources specific settings

**2. Variable declaration file (variables.tf or variables.tf.json):** Here we declare the input variables required to provision resources

**3. Variable definition files (terraform.tfvars):** Here we assign values to the input variables

**4. State file (terraform.tfstate):** a state file is created once after Terraform is run. It stores state about our managed infrastructure.

**Also Read:**Our blog post on [**Terraform Create VM**](https://k21academy.com/terraform-iac/terraform-automate-aws-vm/).

## Getting started using Terraform

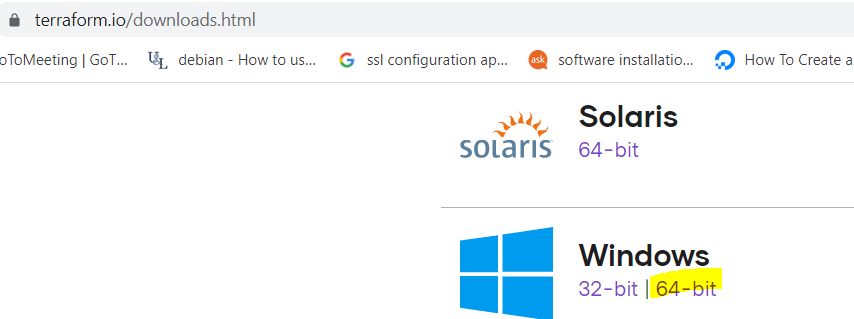
To get started building infrastructure resources using Terraform, there are few things that you should take care of. The general steps to deploy a resource(s) in the cloud are:

1. Set up a Cloud Account on any cloud provider ([AWS](https://k21academy.com/amazon-web-services/aws-solutions-architect/create-aws-free-tier-account/), [Azure](https://k21academy.com/microsoft-azure/create-free-microsoft-azure-trial-account/), [OCI](https://k21academy.com/oracle-cloud/video-register-free-oracle-cloud-trial-account/))
2. Install Terraform
3. Add a provider – AWS, Azure, OCI, GCP, or others
4. Write configuration files
5. Initialize Terraform Providers
6. PLAN (DRY RUN) using terraform plan
7. APPLY (Create a Resource) using terraform apply
8. DESTROY (Delete a Resource) using terraform destroy

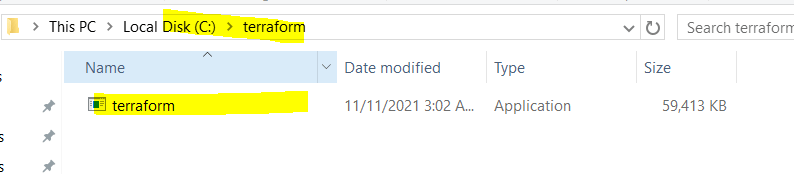
Installation

## Step #1: Download Terraform for Windows

Download the terraform from official terraform website from [Terraform Download page](https://www.terraform.io/downloads.html)



You can see the windows 32 bit and 64 bit source files. Download depends on your system here i am downloading 64 bit terraform. it will be downloaded as zip file. Extract the Terraform setup.

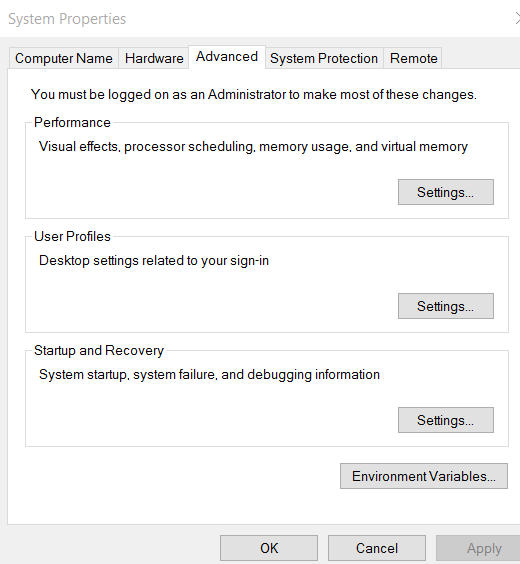


Download Terraform for Windows

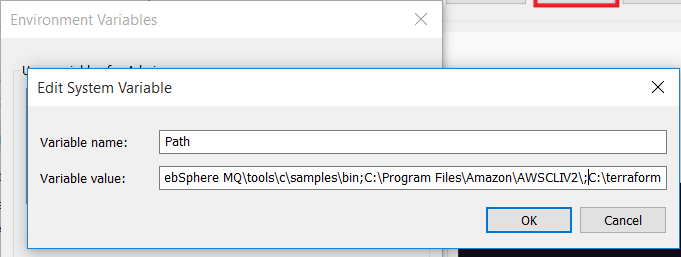
## Step #2: Install Terraform on Windows

To Configure environment variables for terraform navigate to

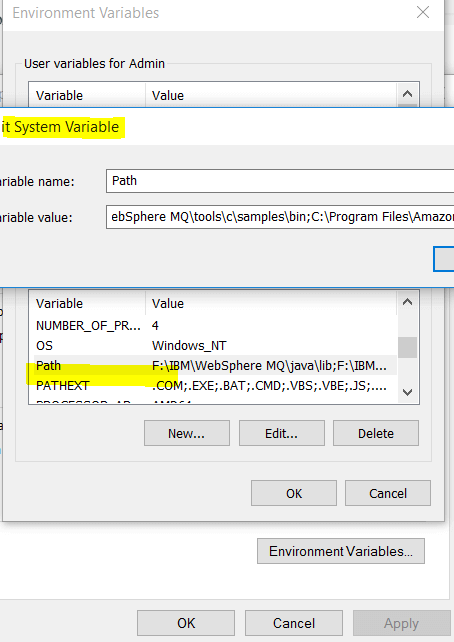
This PC(MyComputer)—>properties —>advanced system settings–>environment variables—>system variables—>path–edit–>new as shown below



Click on Edit System Variable, add terraform path

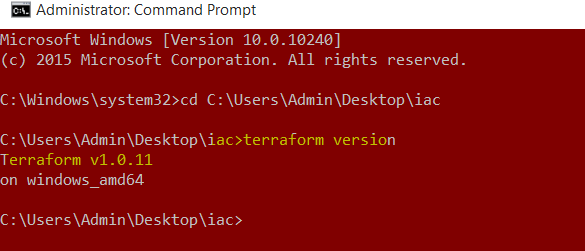


You can see added Terraform setup path.



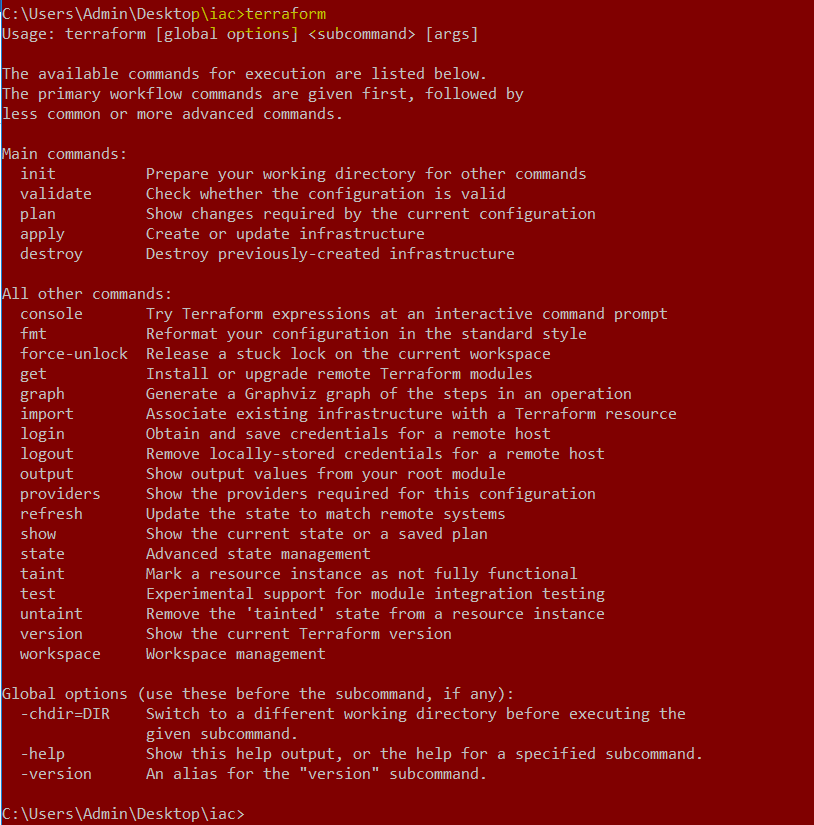
To check terraform version on windows

terraform version



To check terraform commands on windows

terraform



We have covered Install Terraform on Windows.

**Conclusion:**

In this article we have covered How to Install Terraform on Windows and Configure Environment variable on Terraform.

# How to Install Terraform on CentOS 7/8

March 11, 2021 by [FOSS TechNix](https://www.fosstechnix.com/author/skyhighs/)

[Home](https://www.fosstechnix.com/) » How to Install Terraform on CentOS 7/8

In this article, We are going to perform, How to Install Terraform on CentOS 7/8 LTS or any Cloud Instance like Amazon EC2, Azure VM, Google Compute Engine,etc. with preinstalled CentOS.

Table of Contents



# ****Introduction****

Terraform is free and open source Infrastructure automation tool by Hashicorp which is used to create, build and change using infrastructure as a code across various cloud Providers like AWS, Microsoft Azure, GCP, etc.,

## **Prerequisites**

* CentOS with Minimal Installation
* 2 CPU and 4 GB RAM

Update the system packages

$ sudo yum update

Install the **wget and unzip** package to download and extract terraform setup

$ sudo yum install wget unzip -y

## **How to Install Terraform on CentOS**

First download the latest terraform setup from [Terraform Official Site](https://www.terraform.io/downloads.html) using wget.

$ sudo wget https://releases.hashicorp.com/terraform/0.12.26/terraform\_0.12.26\_linux\_amd64.zip

**Output:**

https://releases.hashicorp.com/terraform/0.12.26/terraform\_0.12.26\_linux\_amd64.zip

Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.153.183, 2a04:4e42:24::439

Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.153.183|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 16838433 (16M) [application/zip]

Saving to: ‘terraform\_0.12.26\_linux\_amd64.zip’

terraform\_0.12.26\_linux\_amd64.zip 100%[==============================================================================>] 16.06M --.-KB/s in 0.1s

 (150 MB/s) - ‘terraform\_0.12.26\_linux\_amd64.zip’ saved [16838433/16838433]

Extract the downloaded setup using unzip

$ sudo unzip terraform\_0.12.26\_linux\_amd64.zip

**Output:**

Archive: terraform\_0.12.26\_linux\_amd64.zip

inflating: terraform

Move the extracted setup to /usr/local/bin directory

$ sudo mv terraform /usr/local/bin/

Lets verify Terraform version using below command

$ terraform -v

**Output:**

Terraform v0.12.26

So we have performed How to Install Terraform on CentOS.

Lets check terraform commands

$ terraform

**Output:**

terraform

Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.

The most common, useful commands are shown first, followed by

less common or more advanced commands. If you're just getting

started with Terraform, stick with the common commands. For the

other commands, please read the help and docs before usage.

Common commands:

apply Builds or changes infrastructure

console Interactive console for Terraform interpolations

destroy Destroy Terraform-managed infrastructure

env Workspace management

fmt Rewrites config files to canonical format

get Download and install modules for the configuration

graph Create a visual graph of Terraform resources

import Import existing infrastructure into Terraform

init Initialize a Terraform working directory

login Obtain and save credentials for a remote host

logout Remove locally-stored credentials for a remote host

output Read an output from a state file

plan Generate and show an execution plan

providers Prints a tree of the providers used in the configuration

refresh Update local state file against real resources

show Inspect Terraform state or plan

taint Manually mark a resource for recreation

untaint Manually unmark a resource as tainted

validate Validates the Terraform files

version Prints the Terraform version

workspace Workspace management

All other commands:

0.12upgrade Rewrites pre-0.12 module source code for v0.12

debug Debug output management (experimental)

force-unlock Manually unlock the terraform state

push Obsolete command for Terraform Enterprise legacy (v1)

state Advanced state management

## **Conclusion**

In this article, We have Covered How to Install Terraform on CentOS 7/8, configured terraform, check terraform version and Terraform Commands.

# How to Install Terraform on Ubuntu 20.04/18.04/16.04 LTS

April 8, 2021 by [FOSS TechNix](https://www.fosstechnix.com/author/skyhighs/)

[Home](https://www.fosstechnix.com/) » How to Install Terraform on Ubuntu 20.04/18.04/16.04 LTS

In this article, We are going to perform, How to Install Terraform on Ubuntu 20.04/18.04/16.04 LTS or any Cloud Instance like Amazon EC2, Azure VM, Google Compute Engine,etc. with preinstalled Ubuntu.

Table of Contents



# ****Introduction****

Terraform is free and open source Infrastructure automation tool by Hashicorp which is used to create, build and change using infrastructure as a code across various cloud Providers like AWS, Microsoft Azure, GCP, etc.,

## **Prerequisites**

* Ubuntu Server with 20.04/18.04/16.04 LTS
* 2 CPU and 4 GB RAM

Update the system packages

sudo apt update

Install the **wget and unzip** package to download and extract terraform setup

sudo apt-get install wget unzip -y

## **How to Install Terraform on Ubuntu**

First download the latest terraform version setup from [Terraform Official Site](https://www.terraform.io/downloads.html) using wget.

sudo wget https://releases.hashicorp.com/terraform/0.14.7/terraform\_0.14.7\_linux\_amd64.zip

**Output:**

https://releases.hashicorp.com/terraform/0.14.7/terraform\_0.14.7\_linux\_amd64.zip

Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.153.183, 2a04:4e42:24::439

Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.153.183|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 16838433 (16M) [application/zip]

Saving to: ‘terraform\_0.12.26\_linux\_amd64.zip’

terraform\_0.14.7\_linux\_amd64.zip 100%[==============================================================================>] 16.06M --.-KB/s in 0.1s

 (150 MB/s) - ‘terraform\_0.14.7\_linux\_amd64.zip’ saved [16838433/16838433]

Extract the downloaded setup using unzip

sudo unzip terraform\_0.14.7\_linux\_amd64.zip

**Output:**

Archive: terraform\_0.14.7\_linux\_amd64.zip

inflating: terraform

Move the extracted setup to /usr/local/bin directory

sudo mv terraform /usr/local/bin/

Lets verify Terraform version using below command

terraform -v

**Output:**

Terraform v0.14.7

So we have performed How to Install Terraform on Ubuntu.

Lets check terraform commands

terraform

**Output:**

terraform

Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.

The most common, useful commands are shown first, followed by

less common or more advanced commands. If you're just getting

started with Terraform, stick with the common commands. For the

other commands, please read the help and docs before usage.

Common commands:

apply Builds or changes infrastructure

console Interactive console for Terraform interpolations

destroy Destroy Terraform-managed infrastructure

env Workspace management

fmt Rewrites config files to canonical format

get Download and install modules for the configuration

graph Create a visual graph of Terraform resources

import Import existing infrastructure into Terraform

init Initialize a Terraform working directory

login Obtain and save credentials for a remote host

logout Remove locally-stored credentials for a remote host

output Read an output from a state file

plan Generate and show an execution plan

providers Prints a tree of the providers used in the configuration

refresh Update local state file against real resources

show Inspect Terraform state or plan

taint Manually mark a resource for recreation

untaint Manually unmark a resource as tainted

validate Validates the Terraform files

version Prints the Terraform version

workspace Workspace management

All other commands:

0.12upgrade Rewrites pre-0.12 module source code for v0.12

debug Debug output management (experimental)

force-unlock Manually unlock the terraform state

push Obsolete command for Terraform Enterprise legacy (v1)

state Advanced state management

## How to Install Terraform on Ubuntu/Debian using APT

Add the Hashicorp signing key in Ubuntu/Debian

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

Add the Hashicorp official apt reposity

sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

Install the latest version of terraform on ubuntu/terraform

sudo apt install terraform

To check current and all versions of terraform

apt policy terraform

select the specific terraform version and install it

sudo apt install terraform=0.14.0

## **Conclusion**

In this article, We have Covered How to Install Terraform on Ubuntu, configured terraform, check terraform version and Terraform Commands.

# Creating EC2 Instance using Terraform

December 31, 2021 by [FOSS TechNix](https://www.fosstechnix.com/author/skyhighs/)

[Home](https://www.fosstechnix.com/) » Creating EC2 Instance using Terraform

In this article we are going to cover Creating EC2 Instance using Terraform, connecting to ec2 instance and destroy the resources.

## **Prerequisites**

* AWS Account
* Create IAM User with Admin Access
* Preinstalled Terraform
* Create new Key pairs or use existing

Follow this link to create [free aws account](https://aws.amazon.com/free)

Follow below article to create IAM user in AWS and allow admin access

[How to Create IAM User in AWS Step by Step and setup IAM Policy to user](https://www.devopshint.com/how-to-create-iam-user-in-aws-step-by-step/)

Install Terraform on Linux

[How to Install Terraform on Ubuntu 20.04/18.04/16.04 LTS](https://www.fosstechnix.com/how-to-install-terraform-on-ubuntu/)

## **Creating EC2 Instance using Terraform using minimum variable configuration**

once all above prerequisites done then create a folder/directory

sudo mkdir ec2instance

change the folder/directory permission

sudo chmod 777 -R ec2instance

Navigate to **ec2instance** directory

cd ec2instance

then create **ec2demo.tf** file

sudo nano **ec2demo.tf**

paste the below configuration into it

provider "aws" {

access\_key = "ACCESS\_KEY"

secret\_key = "SECRET\_KEY"

region = "ap-south-1"

}

resource "aws\_instance" "Demo" {

ami = "ami-0d758c1134823146a"

instance\_type = "t2.micro"

key\_name = "key\_pair\_name"

tags = {

Name = "Demo"

}

}

Replace the **ACCESS\_KEY, SECRET\_KEY, ami , instance type, key\_name** and **Instance Name** value.

Initialize the working directory for terraform using below command

terraform init

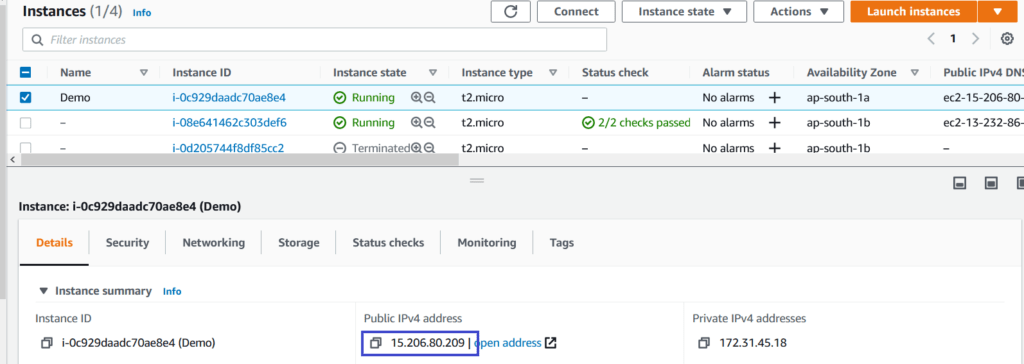
Check the execution plan before applying

terraform plan

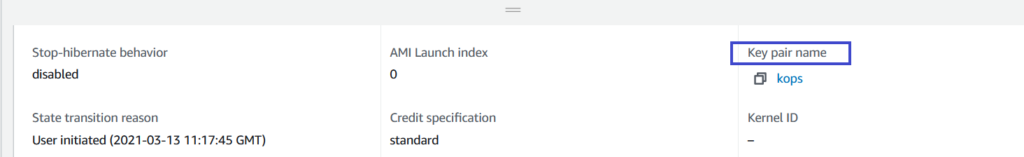
Now build the infrastructure and Type “**yes**” to confirm, it will create ec2 instance in Default VPC, subnets and security groups given by AWS account.

terraform apply

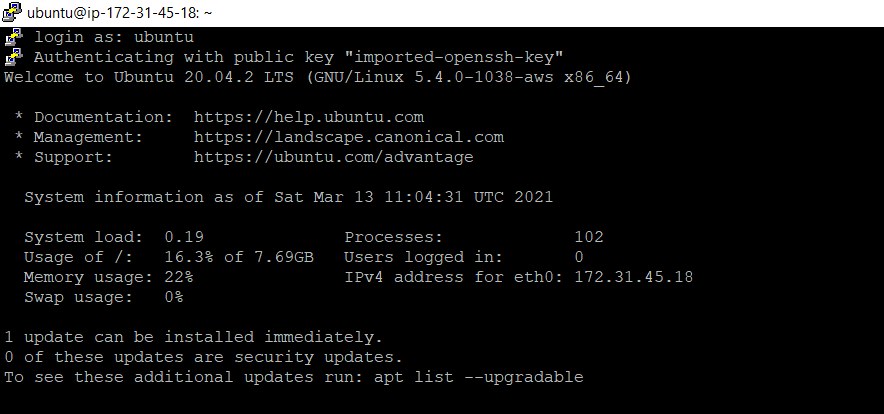
you will see instance is created in AWS



copy the **public address** , select **key pair** name , open the SSH port (22) in security group if not enabled.



Connect to EC2 instance using putty



To destroy the resource use below command and Type “**Yes**“

terraform destroy

To destroy the resources without confirmation use below command

terraform destroy -auto-approve

**Error: Unsupported block type**

Blocks of type “tags” are not expected here. Did you mean to define argument

“tags”? If so, use the equals sign to assign it a value.

**Solution:**

tags = {

Name = "value"

}

**Conclusion:**

We have covered Creating EC2 Instance using Terraform, connecting to ec2 instance and destroy the resources.

# How to Create VPC in AWS using Terraform [7 Steps]

by [FOSS TechNix](https://www.fosstechnix.com/author/skyhighs/)

[Home](https://www.fosstechnix.com/) » How to Create VPC in AWS using Terraform [7 Steps]

In this article we are going to cover Install terraform, Install and configure AWS CLI, Creating IAM user in AWS , configuring AWS credentials and How to Create VPC in AWS using Terraform

Table of Contents



## **Step #1: Install Terraform**

Follow below article to Install Terraform on Ubuntu

[How to Install Terraform on Ubuntu 20.04/18.04/16.04 LTS](https://www.fosstechnix.com/how-to-install-terraform-on-ubuntu/)

## **Step #2: Install AWS CLI** and Configure AWS Credentials

Follow below article to Install AWS CLI and Configure AWS CLI

[How to Install AWS CLI on Linux](https://www.fosstechnix.com/how-to-install-aws-cli-on-linux/)

Follow below article to create IAM user in AWS and allow admin access

[How to Create IAM User in AWS Step by Step and setup IAM Policy to user](https://www.devopshint.com/how-to-create-iam-user-in-aws-step-by-step/)

Export the AWS IAM User credentials using command line

export AWS\_ACCESS\_KEY\_ID=”accesskey”

export AWS\_SECRET\_ACCESS\_KEY=”secretkey”

## **Step #3: Creating a directory and versions.tf file**

Create a directory to save all terraform files which required to create VPC in AWS using terraform.

sudo mkdir dev-vpc

Navigate to directory

cd dev-vpc

Create **version.tf** file

sudo nano versions.tf

paste the below lines into it

terraform {

required\_version = ">= 0.12"

}

## **Step #4: Creating vars.tf file and Adding AWS Region name**

Create **vars.tf** file and add AWS region name where you want to create VPC.

sudo nano vars.tf

Paste below lines into it.

variable "AWS\_REGION" {

default = "ap-south-1"

}

## **Step #5: Creating provider.tf file and adding AWS Region name Variable**

Create **provider.tf** file to add AWS region name variable

sudo nano provider.tf

paste the below lines into it.

provider "aws" {

region = var.AWS\_REGION

}

## **Step #6: How to Create VPC in AWS using Terraform**

Create vpc.tf file to add VPC details

sudo nano vpc.tf

Paste the below lines into it.

# Creating VPC,name, CIDR and Tags

resource "aws\_vpc" "dev" {

cidr\_block = "10.0.0.0/16"

instance\_tenancy = "default"

enable\_dns\_support = "true"

enable\_dns\_hostnames = "true"

enable\_classiclink = "false"

tags = {

Name = "dev"

}

}

# Creating Public Subnets in VPC

resource "aws\_subnet" "dev-public-1" {

vpc\_id = aws\_vpc.dev.id

cidr\_block = "10.0.1.0/24"

map\_public\_ip\_on\_launch = "true"

availability\_zone = "ap-south-1a"

tags = {

Name = "dev-public-1"

}

}

resource "aws\_subnet" "dev-public-2" {

vpc\_id = aws\_vpc.dev.id

cidr\_block = "10.0.2.0/24"

map\_public\_ip\_on\_launch = "true"

availability\_zone = "ap-south-1b"

tags = {

Name = "dev-public-2"

}

}

# Creating Private Subnets in VPC

resource "aws\_subnet" "dev-private-1" {

vpc\_id = aws\_vpc.dev.id

cidr\_block = "10.0.3.0/24"

map\_public\_ip\_on\_launch = "false"

availability\_zone = "ap-south-1a"

tags = {

Name = "dev-private-1"

}

}

resource "aws\_subnet" "dev-private-2" {

vpc\_id = aws\_vpc.dev.id

cidr\_block = "10.0.4.0/24"

map\_public\_ip\_on\_launch = "false"

availability\_zone = "ap-south-1b"

tags = {

Name = "dev-private-2"

}

}

# Creating Internet Gateway in AWS VPC

resource "aws\_internet\_gateway" "dev-gw" {

vpc\_id = aws\_vpc.dev.id

tags = {

Name = "dev"

}

}

# Creating Route Tables for Internet gateway

resource "aws\_route\_table" "dev-public" {

vpc\_id = aws\_vpc.dev.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.dev-gw.id

}

tags = {

Name = "dev-public-1"

}

}

# Creating Route Associations public subnets

resource "aws\_route\_table\_association" "dev-public-1-a" {

subnet\_id = aws\_subnet.dev-public-1.id

route\_table\_id = aws\_route\_table.dev-public.id

}

resource "aws\_route\_table\_association" "dev-public-2-a" {

subnet\_id = aws\_subnet.dev-public-2.id

route\_table\_id = aws\_route\_table.dev-public.id

}

## **Step #7: Creating NAT Gateway and Routes Tables in AWS VPC**

Create **nat.tf** file to add VPC NAT gateway and route tables for private subnets

sudo nano nat.tf

Paster the below lines into it.

# Creating Nat Gateway

resource "aws\_eip" "nat" {

vpc = true

}

resource "aws\_nat\_gateway" "nat-gw" {

allocation\_id = aws\_eip.nat.id

subnet\_id = aws\_subnet.dev-public-1.id

depends\_on = [aws\_internet\_gateway.dev-gw]

}

# Add routes for VPC

resource "aws\_route\_table" "dev-private" {

vpc\_id = aws\_vpc.dev.id

route {

cidr\_block = "0.0.0.0/0"

nat\_gateway\_id = aws\_nat\_gateway.nat-gw.id

}

tags = {

Name = "dev-private-1"

}

}

# Creating route associations for private Subnets

resource "aws\_route\_table\_association" "dev-private-1-a" {

subnet\_id = aws\_subnet.dev-private-1.id

route\_table\_id = aws\_route\_table.dev-private.id

}

resource "aws\_route\_table\_association" "dev-private-2-a" {

subnet\_id = aws\_subnet.dev-private-2.id

route\_table\_id = aws\_route\_table.dev-private.id

}

Run the **terraform init** in current directory

terraform init

Run the **terraform plan** in current directory

terraform plan

**Sample Output:**

# aws\_vpc.dev will be created

+ resource "aws\_vpc" "dev" {

+ arn = (known after apply)

+ assign\_generated\_ipv6\_cidr\_block = false

+ cidr\_block = "10.0.0.0/16"

+ default\_network\_acl\_id = (known after apply)

+ default\_route\_table\_id = (known after apply)

+ default\_security\_group\_id = (known after apply)

+ dhcp\_options\_id = (known after apply)

+ enable\_classiclink = false

+ enable\_classiclink\_dns\_support = (known after apply)

+ enable\_dns\_hostnames = true

+ enable\_dns\_support = true

+ id = (known after apply)

+ instance\_tenancy = "default"

+ ipv6\_association\_id = (known after apply)

+ ipv6\_cidr\_block = (known after apply)

+ main\_route\_table\_id = (known after apply)

+ owner\_id = (known after apply)

+ tags = {

+ "Name" = "main"

}

+ tags\_all = {

+ "Name" = "dev"

}

}

Plan: 14 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Run **terraform apply** to create VPC in AWS using Terraform

terraform apply

**Sample Output:**

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\_eip.nat: Creating...

aws\_vpc.dev: Creating...

aws\_eip.nat: Creation complete after 0s [id=eipalloc-05379158933e51915]

aws\_vpc.dev: Still creating... [10s elapsed]

aws\_vpc.dev: Creation complete after 11s [id=vpc-0ce7096b7733ce0c8]

aws\_subnet.dev-private-2: Creating...

aws\_subnet.dev-public-2: Creating...

aws\_subnet.dev-public-1: Creating...

aws\_subnet.dev-private-1: Creating...

aws\_internet\_gateway.dev-gw: Creating...

aws\_internet\_gateway.dev-gw: Creation complete after 0s [id=igw-0353f0af211521751]

aws\_route\_table.dev-public: Creating...

aws\_subnet.dev-private-1: Creation complete after 0s [id=subnet-009eba556c3a60c6c]

aws\_subnet.dev-private-2: Creation complete after 0s [id=subnet-058b33281fd02d4d1]

aws\_route\_table.dev-public: Creation complete after 0s [id=rtb-0bd1d40c43fe1d549]

aws\_subnet.dev-public-2: Still creating... [10s elapsed]

aws\_subnet.dev-public-1: Still creating... [10s elapsed]

aws\_subnet.dev-public-2: Creation complete after 10s [id=subnet-0d2df9e3589ddb413]

aws\_route\_table\_association.dev-public-2-a: Creating...

aws\_subnet.dev-public-1: Creation complete after 10s [id=subnet-0e6e2a39eb75e2cf8]

aws\_route\_table\_association.dev-public-1-a: Creating...

aws\_nat\_gateway.nat-gw: Creating...

aws\_route\_table\_association.dev-public-2-a: Creation complete after 0s [id=rtbassoc-0e3a08d1bee9d404a]

aws\_route\_table\_association.dev-public-1-a: Creation complete after 0s [id=rtbassoc-0a8a84a1f3a4d9cf0]

aws\_nat\_gateway.nat-gw: Still creating... [10s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [20s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [30s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [40s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [50s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [1m0s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [1m10s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [1m20s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [1m30s elapsed]

aws\_nat\_gateway.nat-gw: Still creating... [1m40s elapsed]

aws\_nat\_gateway.nat-gw: Creation complete after 1m44s [id=nat-09d99bb965579549c]

aws\_route\_table.dev-private: Creating...

aws\_route\_table.dev-private: Creation complete after 0s [id=rtb-09d5b62e2df8116bc]

aws\_route\_table\_association.dev-private-2-a: Creating...

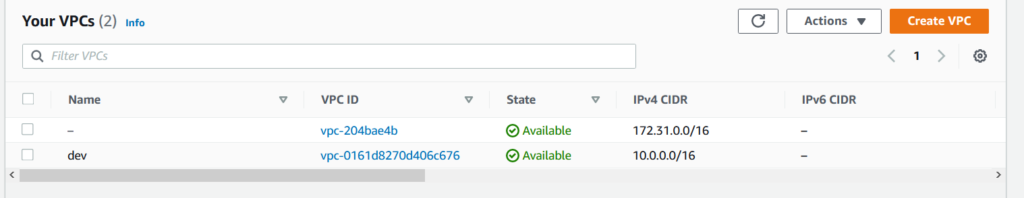
aws\_route\_table\_association.dev-private-1-a: Creating...

aws\_route\_table\_association.dev-private-1-a: Creation complete after 1s [id=rtbassoc-02e46253919662c35]

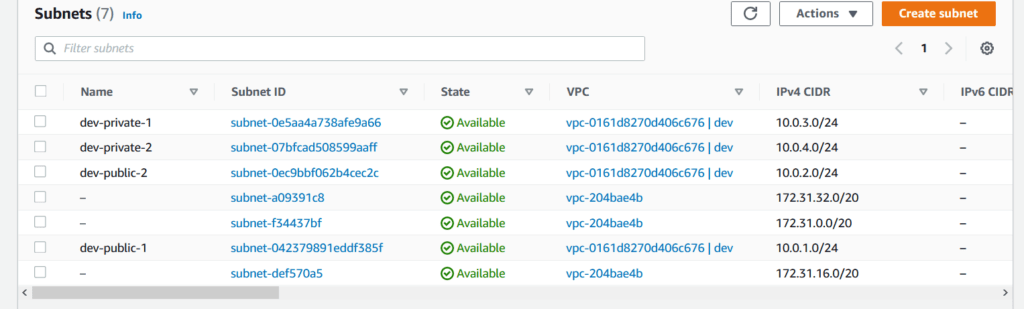
aws\_route\_table\_association.dev-private-2-a: Creation complete after 1s [id=rtbassoc-0c806d1ed44a30eff]

Apply complete! Resources: 14 added, 0 changed, 0 destroyed.

Login to AWS console you will see VPC is created



Click on Subnets in VPC



Run below command to destroy the VPC in AWS

terraform destroy

**Conclusion:**

We have covered Install terraform, Install and configure AWS CLI, Creating IAM user in AWS , configuring AWS credentials and How to Create VPC in AWS using Terraform.