

# Terraform

---

[https://github.com/DhruvinSoni30/Terraform\\_Modules\\_Jenkins](https://github.com/DhruvinSoni30/Terraform_Modules_Jenkins)

<https://awstip.com/iac-with-terraform-and-aws-overview-fb26fb4991cf>

<https://medium.com/geekculture/terraform-input-variables-7557702e3e4c>

<https://jaffarshaik.medium.com/deploy-node-js-app-using-lambda-api-gateway-through-terraform-modules-9cf11d665aaa>

<https://aws.plainenglish.io/terraform-using-variables-and-count-b161e0ce61e1>

<https://medium.com/appgambit/provisioning-a-jenkins-server-on-aws-with-terraform-bf04a6a6ef7f>

<https://medium.com/schibsted-engineering/ultimate-terraform-project-structure-9fc7e79f6bc6>

<https://aws.plainenglish.io/the-basics-of-terraform-57a66b269c43>

<https://medium.com/@cyber-security/devops-01-example-project-with-terraform-ade540824db1>

<https://yankeexe.medium.com/how-rolling-and-rollback-deployments-work-in-kubernetes-8db4c4dce599>

<https://medium.com/swlh/k8s-volumes-claims-part1-138012b6f52>

<https://komodor.com/learn/kubernetes-troubleshooting-the-complete-guide/>

<https://devopslearners.com/kubernetes-ingress-tutorial-for-beginners-26c2f7727bc>

<https://medium.com/@omar.egal/how-to-create-a-highly-available-3-tier-aws-architecture-e8f39af22aab>

<https://aws.plainenglish.io/the-basics-of-terraform-57a66b269c43>

<https://medium.com/appgambit/provisioning-a-jenkins-server-on-aws-with-terraform-bf04a6a6ef7f>

<https://code.egym.de/horizontal-pod-autoscaler-in-kubernetes-part-2-advanced-autoscaling-using-prometheus-adapter-d2b8bc3a019d>

<https://lakhansiddharth94.medium.com/kubernetes-ingress-setup-bf950dbe1fe5>

<https://devopslearners.com/kubernetes-ingress-tutorial-for-beginners-26c2f7727bc>

<https://medium.com/@omar.egal/how-to-create-a-highly-available-3-tier-aws-architecture-e8f39af22aab>

<https://awstip.com/sre-devops-interview-questions-linux-troubleshooting-extended-c12cb5ded3b0>

<https://medium.com/appgambit/provisioning-a-jenkins-server-on-aws-with-terraform-bf04a6a6ef7f>

<https://aws.plainenglish.io/the-basics-of-terraform-57a66b269c43>

<https://medium.com/@cyber-security/devops-01-example-project-with-terraform-ade540824db1>

<https://towardsdev.com/still-doing-infrastructure-manually-check-this-infrastructure-as-code-tool-to-speed-up-workflow-6d3967eaf794>

<https://blog.devops.dev/terraform-state-file-and-remote-backend-with-hands-on-best-practices-of-authentication-and-4f16c2ae06ef>

## Provisioning a Jenkins Server on AWS With Terraform

In this article, we are talking about How we can deploy the Jenkins server on AWS EC2 instance using Terraform script.



### Prerequisites

- We require AWS IAM API keys (access key and secret key) with EC2 permissions.
- Terraform should be installed on the machine. If Terraform does not exist you can download and install it from [here](#).

### Resources Created Using Terraform

- AWS VPC
- Public Subnet
- EC2 Key pair
- EC2 Instance
- Security Group
- Elastic IP

### Let's Start!

#### 1. Create a "provider.tf"

This is the provider file that tells Terraform, which provider you are using.

All infrastructure will be on the AWS because of *provider "aws"*. If you want to use another cloud provider such as GCP or Azure, you need to change this.

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

We already declared the region of AWS where we are creating a

VPC network.

you can declare a profile also if you are working on multiple AWS accounts.

profile = "<PROFILE NAME>"

by default, it will take your default profile.

## 2. Create "variables.tf"

All variables will be in this file. Now, there is only one region but there will be more...



```
variable "aws_region" {  
  default = "us-east-2"  
}
```

You, 3 hours ago | 1 author (You)

```
variable "vpc_cidr_block" {  
  description = "CIDR block for VPC"  
  type = string  
  default = "10.0.0.0/16"  
}
```

You, 3 hours ago | 1 author (You)

```
variable "public_subnet_cidr_block" {  
  description = "CIDR block for public subnet"  
  type = string  
  default = "10.0.1.0/24"  
}
```

You, 3 hours ago | 1 author (You)

```
variable "my_ip" {  
  description = "Your IP address"  
  type = string  
  sensitive = true  
}
```

If you are using terraform.tfvars you just need to add a description only.

## 3. Create "terraform.tfvars"

To persist variable values, create a file, and assign variables within this file. Create a file named terraform.tfvars with the

following contents:

```
aws_region      = "us-east-1"
vpc_cidr_block  = "10.0.0.0/16"
public_subnet_cidr_block = "10.0.1.0/24"
my_ip           = "192.168.0.224"
```

For all files which match terraform.tfvars or \*.auto.tfvars present in the current directory, Terraform automatically loads them to populate variables. If the file is named something else, you can use the -var-file flag directly to specify a file.

Personally, I don't recommend saving usernames and passwords to version control, but you can create a local secret variables file and use -var-file to load it.

#### 4. Create "modules > VPC" Folder

A [module](#) is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that you can describe your infrastructure in terms of its architecture, rather than directly in terms of physical objects.

#### 5. Create "Main.tf" in the VPC folder.

```
data "aws_availability_zones" "available" {
  state = "available"
}

resource "aws_vpc" "aws_vpc" {
  cidr_block = var.vpc_cidr_block
  enable_dns_hostnames = true

  tags = {
    Name = "aws_vpc"
  }
}

resource "aws_internet_gateway" "aws_igw" {
  vpc_id = aws_vpc.aws_vpc.id

  tags = {
    Name = "aws_igw"
  }
}
```

```

}

resource "aws_subnet" "aws_public_subnet" {
  vpc_id = aws_vpc.aws_vpc.id
  cidr_block = var.public_subnet_cidr_block
  availability_zone = data.aws_availability_zones.available.names[0]

  tags = {
    Name = "aws_public_subnet_1"
  }
}

resource "aws_route_table" "aws_public_rt" {
  vpc_id = aws_vpc.aws_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.aws_igw.id
  }
}

resource "aws_route_table_association" "public" {
  route_table_id = aws_route_table.aws_public_rt.id
  subnet_id = aws_subnet.aws_public_subnet.id
}

```

**Note: Here we are using basic VPC setup if you want to explore Full VPC setup with public private subnets with NAT please follow [Terraform — AWS VPC with Private, Public Subnets with NAT](#)**

Terraform is so popular nowadays. Terraform enables you to create and manage infrastructure with code and codes can be...  
[medium.com](https://medium.com)

## 6. Create “variables.tf” in the VPC folder.

This is the same as the above variable.tf file just declare all variables that we are using in main.tf a file so we can use get all variables value from main main.tf file.

```

You, 3 hours ago | 1 author (You)
variable "vpc_cidr_block" {
  description = "CIDR block of the VPC"
}

You, 3 hours ago | 1 author (You)
variable "public_subnet_cidr_block" {
  description = "CIDR block of the public subnet"
}

```

## 7. Create "output.tf" in the VPC folder.

We can export any details from created resources and give that as an input of another module.

```

output "public_subnet_id" {
  description = "ID of the public subnet"
  value = aws_subnet.aws_public_subnet.id
}

You, 3 hours ago | 1 author (You)
output "vpc_id" {
  description = "ID of the VPC"
  value = aws_vpc.aws_vpc.id
}

```

We can access output value in another submodule like

```
vpc_cidr      = "${module.vpc.vpc_id}"
```

## 8. Create "modules > compute" Folder

A [module](#) is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that you can describe your infrastructure in terms of its architecture, rather than directly in terms of physical objects.

## 9. Create "Main.tf" in the compute folder.

```

data "aws_ami" "ubuntu" {
  most_recent = "true"

  filter {
    name = "name"
  }
}

```

```

    values = ["ubuntu/images/hvm-ssd/ubuntu-*-amd64-server-*"]
}

filter {
    name = "virtualization-type"
    values = ["hvm"]
}

owners = ["amazon"]
}

resource "aws_instance" "jenkins_server" {
    ami = data.aws_ami.ubuntu.id
    subnet_id = var.public_subnet
    instance_type = "t2.micro"
    vpc_security_group_ids = [aws_security_group.aws_jenkins_sg.id]
    key_name = aws_key_pair.aws_kp.key_name
    user_data = "${file("${path.module}/install_jenkins.sh")}"

    tags = {
        Name = "jenkins_server"
    }
}

resource "aws_key_pair" "aws_kp" {
    key_name = "aws_kp"
    public_key = file("${path.module}/aws_kp.pub")
}

resource "aws_eip" "jenkins_eip" {
    instance = aws_instance.jenkins_server.id
    vpc      = true

    tags = {
        Name = "jenkins_eip"
    }
}

resource "aws_security_group" "aws_jenkins_sg" {
    name = "aws_jenkins_sg"
    description = "Security group for jenkins server"
    vpc_id = var.vpc_id

    ingress {
        description = "Allow all traffic through port 8080"
        from_port = "8080"
        to_port = "8080"
    }
}

```

```

    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}

ingress {
    description = "Allow SSH from my computer"
    from_port = "22"
    to_port = "22"
    protocol = "tcp"
    cidr_blocks = ["${var.my_ip}/32"]
}

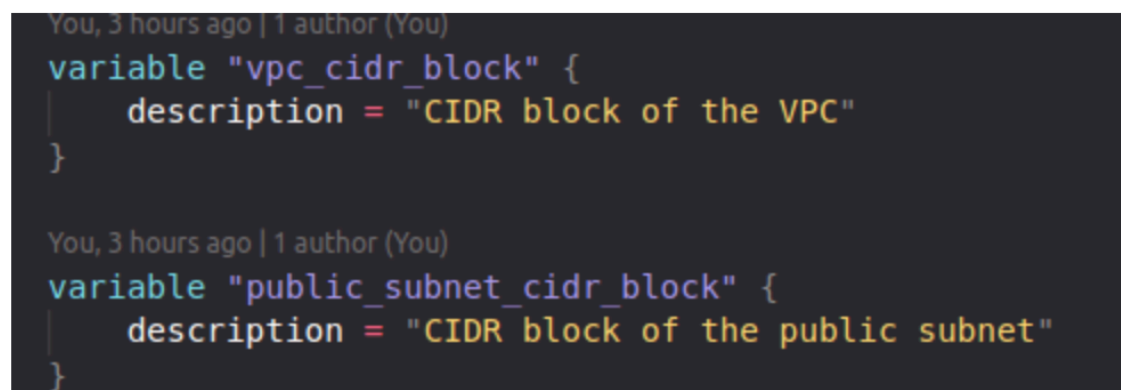
egress {
    description = "Allow all outbound traffic"
    from_port = "0"
    to_port = "0"
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}

tags = {
    Name = "aws_jenkins_sg"
}
}

```

## 10. Create "variables.tf" in the compute folder.

This is the same as the above variable.tf file just declare all variables that we are using in main.tf a file so we can use get all variables value from main main.tf file.



```

variable "vpc_cidr_block" {
    description = "CIDR block of the VPC"
}

variable "public_subnet_cidr_block" {
    description = "CIDR block of the public subnet"
}

```

## 11. Create "output.tf" in the compute folder.

We can export any details from created resources and give that



as an input of another module.

```
output "public_ip" {  
  description = "The public IP address of the Jenkins server"  
  value = aws_eip.jenkins_eip.public_ip  
}
```

**11. Create "install\_jenkins.sh" in the compute folder.**  
**Jenkins installation script that deploy Jenkins on our EC2 instance.**

```
#!/bin/bash  
  
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 upgrade -y  
sudo apt install default-jre -y  
sudo apt install jenkins -y  
sudo systemctl start jenkins
```

**12. Create "main.tf"**

**main.tf** files in your working directory when you run **terraform plan** or **terraform apply** together form the root module. That module may call other modules and connect them by passing output values from one to the input values of another. To learn how to use modules, see the **Modules configuration** section.

You, 3 hours ago | 1 author (You)

```
module "vpc" {  
  source = "../modules/vpc"  
  
  vpc_cidr_block      = var.vpc_cidr_block  
  public_subnet_cidr_block = var.public_subnet_cidr_block  
}
```

You, 3 hours ago | 1 author (You)

```
module "ec2_instance" {  
  source      = "../modules/compute"  
  public_subnet = module.vpc.public_subnet_id  
  vpc_id      = module.vpc.vpc_id  
  my_ip       = var.my_ip  
}
```

## 12. Create Key pair for EC2 key

Following command will generate Key pair files and move that to our compute module.

```
ssh-keygen -t rsa -b 4096 -m pem -f aws_kp && mv aws_kp.pub modules/  
compute/aws_kp.pub && mv aws_kp aws_kp.pem && chmod 400 aws_kp.pem
```

**Now, We are ready to init!**

Run ``terraform init`` that download all modules information and download terraform in your project file.



After that, you can see the `.terraform` folder in your project directory that contains terraform setup and modules information.

`terraform plan`

The [terraform plan](#) command is used to create an execution plan. Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.

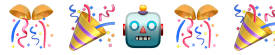
This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or the state. For example, `terraform plan` might be run before committing a change to version control, to create confidence that it will behave as expected.



`terraform apply`

The [terraform apply](#) command is used to apply the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform

**plan execution plan.**



**Our VPC Setup is ready on AWS.**

**Just you need to follow above all steps or clone this repository to start terraforming.**

**GitHub - PrashantBhatasana/tf-jenkins-on-ec2: Terraform script to deploy Jenkins on EC2**

provider "aws" { region = "\${var.region}" profile = " " //If you run this script using AWS EC2 } ssh-keygen -t rsa -b...  
github.com

**After cloning the repo, just run the following commands.**

*generate Key pair with following command*

```
ssh-keygen -t rsa -b 4096 -m pem -f aws_kp && mv aws_kp.pub modules/  
compute/aws_kp.pub && mv aws_kp aws_kp.pem && chmod 400 aws_kp.pem
```

**change values in terraform.tfvars.**

**terraform init**

**terraform plan**

**terraform apply**

**Once it will done you can access Jenkins with EC2 Instance public IP address**

<http://<Instance public IP>:8080>

**Please follow this to configure Jenkins**

Feature	Classic Load Balancer	Application Load Balancer
Protocols	HTTP, HTTPS, TCP, SSL	HTTP, HTTPS
Platforms	EC2-Classic, EC2-VPC	EC2-VPC
Sticky sessions (cookies)	YES (you can provide your own application cookie)	Load balancer generated
Back-end server authentication	YES	NO
Back-end server encryption	YES	YES
Idle connection timeout	YES	YES
Connection draining	YES	YES
Cross-zone load balancing	YES	Always enabled
Health checks	YES	YES (Improved)
CloudWatch metrics	YES	YES (Improved)
Access logs	YES	YES (Improved)
Path-based routing	NO	YES
Route to multiple ports on a single instance	NO	YES
HTTP/2 support	NO	YES
Websockets support	NO	YES
Load balancer deletion protection	NO	YES