What is terraform provisioner?

Terraform Provisioners are used for executing scripts or shell commands on a local or remote machine as part of resource creation/deletion. They are similar to “EC2 instance user data” scripts that only run once on the creation and if it fails terraform marks it tainted.

Terraform Provisioners are used to performing certain custom actions and tasks either on the local machine or on the remote machine.

**The custom actions can vary in nature and it can be -**

1. Running custom shell script on the local machine
2. Running custom shell script on the remote machine
3. Copy file to the remote machine

**Also there are two types of provisioners -**

1. Generic Provisioners (file, local-exec, and remote-exec)
2. Vendor Provisioners (chef, habitat, puppet, salt-masterless)

**Generic Provisioners -** Generally vendor independent and can be used with any cloud vendor(GCP, AWS, AZURE)

**Vendor Provisioners -** It can only be used only with its vendor. For example, chef provisioner can only be used with

Table of Content

1. [*file* provisioner](https://jhooq.com/terraform-provisioner/#1-file-provisioner)
2. [*local-exec* provisioner](https://jhooq.com/terraform-provisioner/#2-local-exec-provisioner)
3. [*remote-exec* provisioner](https://jhooq.com/terraform-provisioner/#3-remote-exec-provisioner)
4. [chef provisoners](https://jhooq.com/terraform-provisioner/#4-chef-provisioner)
5. [habitat provisoners](https://jhooq.com/terraform-provisioner/#5-habitat-provisioner)
6. [puppet provisoners](https://jhooq.com/terraform-provisioner/#6-puppet-provisioner)
7. [salt-masterless provisoners](https://jhooq.com/terraform-provisioner/#7-salt-masterless)

(*Note - All the provisioners must be used in moderation, it is not advisable to use provisioners in excess*)

1. file provisioner

As the name suggests *file provisioner* can be used for transferring and copying the files from one machine to another machine.

Not only file but it can also be used for transferring/uploading the directories.

So when we talk about copying files or directories from one machine to another machine then it has to be secured and *file provisioner* supports for ssh and winrm type of connections which can help you to achieve secure file transfer between the source machine and destination machine.

Let us take an example to understand how to implement terraform file provisioner. The following code snippet shows -

1. How to write your file provisioner
2. How to specify source and destination` for copying/transferring the file.

provisioner "file" {

source = "c:\Users\Admin\Downloads\Aws\_keys\Hello.txt"

destination = "/home/ubuntu/Hello.txt"

}

In the above code snippet, we are trying to copy file Hello.txt from its **source =** c:\Users\Admin\Downloads\Aws\_keys\Hello.txt to its **destination =**/home/ubuntu/Hello.txt

Here is the complete terraform script which demonstrates on how to use *terraform file provisioner*

provider "aws" {

region = "ap-south-1"

access\_key = "xxxxxxxxxxxxxxxxxxxxx"

secret\_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-06984ea821ac0a879"

instance\_type = "t2.micro"

key\_name= "vcube"

vpc\_security\_group\_ids = [aws\_security\_group.main.id]

provisioner "file" {

source = " c:\Users\Admin\Downloads\Aws\_keys\Hello.txt"

destination = "/home/ubuntu/Hello.txt"

}

connection {

type = "ssh"

host = self.public\_ip

user = "ubuntu"

private\_key = file("c:\Users\Admin\Downloads\Aws\_keys\vcube")

timeout = "4m"

}

}

resource "aws\_security\_group" "main" {

egress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 0

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "-1"

security\_groups = []

self = false

to\_port = 0

}

]

ingress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 22

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "tcp"

security\_groups = []

self = false

to\_port = 22

}

]

}

resource "aws\_key\_pair" "deployer" {

key\_name = "vcube"

public\_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDbvRN/gvQBhFe+dE8p3Q865T/xTKgjqTjj56p1IIKbq8SDyOybE8ia0rMPcBLAKds+wjePIYpTtRxT9UsUbZJTgF+SGSG2dC6+ohCQpi6F3xM7ryL9fy3BNCT5aPrwbR862jcOIfv7R1xVfH8OS0WZa8DpVy5kTeutsuH5FMAmEgba4KhYLTzIdhM7UKJvNoUMRBaxAqIAThqH9Vt/iR1WpXgazoPw6dyPssa7ye6tUPRipmPTZukfpxcPlsqytXWlXm7R89xAY9OXkdPPVsrQA0XFQnY8aFb9XaZP8cm7EOVRdxMsA1DyWMVZOTjhBwCHfEIGoePAS3jFMqQjGWQd cloud@cloudbook-HP-ZBook-77-G7"

}

...

*BASH*

**Here is one thing to note -** *You need to generate the ssh keys to connect to your EC2 instance running in the AWS cloud. You can use the command ssh-keygen -t aws\_key to generate the key-pair. You can read this blog post on*[*Terraform how to do SSH in AWS EC2 instance?*](https://jhooq.com/terraform-ssh-into-aws-ec2/)

Supporting arguments for file provisioners

***1. source -*** The source argument is used to specify the location from where you want to pick the file. The source location can be relative to your project structure.

Here are some examples where I have used relative path for the source arguments -

provisioner "file" {

source = "../../../cloud/keys/aws/Hello.txt"

destination = "/home/ubuntu/Hello.txt"

}

*BASH*

***2. content -*** The content argument is useful when you do not want to copy or transfer the file instead you only want to copy the content or string.

Here is an example of a content resource argument -

provisioner "file" {

content = "I want to copy this string to the destination file server.txt"

destination = "/home/ubuntu/server.txt"

}

*BASH*

The above provisioner script will copy the string I want to copy this string to the destination file server.txt to the destination file /home/ubuntu/server.txt

***3. destination -*** As the name suggest you need to input the final destination path where you want your file to be.

2. local-exec provisioner

The next provisioner we are gonna talk about is *local-exec provisioner*. Basically, this provisioner is used when you want to perform some tasks onto your local machine where you have installed the terraform.

So local-exec provisioner is never used to perform any kind task on the remote machine. It will always be used to perform local operations onto your local machine.

***Example -*** Consider the following example where we are trying to create a file hello-cloud.txt on the local machine

provisioner "local-exec" {

command = "touch hello-cloud.txt"

}

*BASH*

In the command section, we can write a bash script. In the above example, I am trying to create a hello-cloud.txt file on the local machine.

Here is the complete terraform script for the above example -

provider "aws" {

region = "ap-south-1"

access\_key = "xxxxxxxxxxxxxxxx"

secret\_key = "xxxxxxxxxxxxxxxxxxxxxxxx"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0767046d1677be5a0"

instance\_type = "t2.micro"

tags = {

Name = "Terraform EC2"

}

provisioner "local-exec" {

command = "touch hello-cloud.txt"

}

}

*BASH*

3. remote-exec provisioner

As the name suggests remote-exec it is always going to work on the remote machine. With the help of the remote-exec you can specify the commands of shell scripts that want to execute on the remote machine.

As we discussed ssh and winrm for secure data transfer in [local-exec](https://jhooq.com/terraform-provisioner/#1-file-provisioner), here also all the communication and file transfer is done securely.

Let us take an example of how to implement the remote-exec provisioner -

provisioner "remote-exec" {

inline = [

"touch hello.txt",

"echo helloworld remote provisioner >> hello.txt",

“mkdir vcube-book”,

]

}

*BASH*

In the above example -

1. First we are going to create a file named hello.txt
2. We are going to write the message helloworld remote provisioner inside the hello.txt file.
3. Everything will happen on the remote machine

Here is the complete example of remote-exec -

provider "aws" {

region = "ap-south-1"

access\_key = "xxxxxxxxxxxxxxx"

secret\_key = "xxxxxxxxxxxxxxxxxxxxxxxx"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0767046d1677be5a0"

instance\_type = "t2.micro"

key\_name= "aws\_key"

vpc\_security\_group\_ids = [aws\_security\_group.main.id]

provisioner "remote-exec" {

inline = [

"touch hello.txt",

"echo helloworld remote provisioner >> hello.txt",

]

}

connection {

type = "ssh"

host = self.public\_ip

user = "ubuntu"

private\_key = file("c:\Users\Admin\Downloads\Aws\_keys\cloud\_key")

timeout = "4m"

}

}

resource "aws\_security\_group" "main" {

egress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 0

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "-1"

security\_groups = []

self = false

to\_port = 0

}

]

ingress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 22

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "tcp"

security\_groups = []

self = false

to\_port = 22

}

]

}

resource "aws\_key\_pair" "deployer" {

key\_name = "aws\_key"

public\_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDbvRN/gvQBhFe+dE8p3Q865T/xTKgjqTjj56p1IIKbq8SDyOybE8ia0rMPcBLAKds+wjePIYpTtRxT9UsUbZJTgF+SGSG2dC6+ohCQpi6F3xM7ryL9fy3BNCT5aPrwbR862jcOIfv7R1xVfH8OS0WZa8DpVy5kTeutsuH5FMAmEgba4KhYLTzIdhM7UKJvNoUMRBaxAqIAThqH9Vt/iR1WpXgazoPw6dyPssa7ye6tUPRipmPTZukfpxcPlsqytXWlXm7R89xAY9OXkdPPVsrQA0XFQnY8aFb9XaZP8cm7EOVRdxMsA1DyWMVZOTjhBwCHfEIGoePAS3jFMqQjGWQd cloud@cloudbook-HP-ZBook-77-G7"

}

...

*BASH*

Supporting arguments for remote provisioners

***1. inline -*** With the help of an inline argument you can specify the multiple commands which you want to execute in an ordered fashion.

Here is an example in which I have added two separate commands -

provisioner "remote-exec" {

inline = [

"touch hello.txt",

"echo helloworld remote provisioner >> hello.txt",

]

}

*BASH*

***2. script -*** It can be used to copy the script from local machine to remote machine and it always contains a relative path.

In the script, you can not specify multiple scripts. You can only mention one script which needs to be copied to the remote machine.

***3. scripts -*** Here you can specify the multiple local scripts which want to copy or transfer to the remote machine and execute over there.

Always remember the order of the file will not change and it going to execute in the same order way you have mentioned.

# Terraform how to do SSH in AWS EC2 instance?

**how to do SSH into AWS EC2 instance using**[**Terraform**](https://www.terraform.io/). It is quite often that you [create your terraform script for setting up your **EC2 instance**](https://jhooq.com/terraform-ec2-instance-setup/) but after starting your EC2 can not SSH into EC2 instance created with Terraform.

When you are working on the cloud ([AWS](https://aws.amazon.com/), [Google Cloud](https://cloud.google.com/)) you are not provided with a password for the root user. You have to use [SSH key-pair(public key, private key)](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html) for authentication and after successful authentication, you can SSH into your EC2 machine using the private key.

In this blog post, we will go through the following steps -

1. [Generate key-pair(public key, private key) using ssh keygen](https://jhooq.com/terraform-ssh-into-aws-ec2/#1-generate-key-pairpublic-key-private-key-using-_ssh-keygen_)
2. [Use public key to start EC2 instance](https://jhooq.com/terraform-ssh-into-aws-ec2/#2-use-public-key-to-start-ec2-instance)
3. [Use the private key to SSH into EC2 instance](https://jhooq.com/terraform-ssh-into-aws-ec2/#3-use-private-key-aws_key-to-ssh-into-ec2-instance)
4. [Generate pem file from AWS console and use the file to SSH into EC2](https://jhooq.com/terraform-ssh-into-aws-ec2/#4-generate-pem-file-from-aws-console-and-use-the-file-to-ssh-into-ec2)

(\*Note - The first 3 steps in which we are going to generate the ssh key-pair manually and in the 4th step we will generate the key pair from AWS console.If you do not want to generate the SSH keys manually then you can refer to this blog post where I described on [*how to use terraform's tls\_private\_key module to generate and upload private, public key for AWS*](https://jhooq.com/terraform-generate-ssh-key))

## 1. Generate key-pair(public key, private key) using ssh keygen

#### 1.1 Generate the public key and private key

Before you start playing with AWS console and terraform script we need to first generate the key-pair(public key, private key) using ssh-keygen.

Later we are going to associate both public and private keys with AWS EC2 instances.

Let us generate the key pair using the following command

ssh-keygen -t rsa -b 2048

*BASH*

By default, the above command will generate the public as well as private key at location '/home//.ssh'

But we can override the end destination with a custom path. (I have assigned my custom path */home/cloud/ajay/keys/aws* followed my key name .i.e. *aws\_key* )

Here is the output along with a screenshot my terminal-

Generating public/private RSA key pair.

Enter file in which to save the key (/home/cloud/.ssh/id\_rsa): /home/cloud/ajay/keys/aws/aws\_key

Enter passphrase (empty for no passphrase):

Enter the same passphrase again:

Your identification has been saved in /home/cloud/ajay/keys/aws/aws\_key

Your public key has been saved in /home/cloud/ajay/keys/aws/aws\_key.pub

The key fingerprint is:

SHA256:sAOjXyvJc2gnMrvxXA+qiaU9pUEvwl5ZG9Y2kZqRf5M rahul@rahul-HP-ZBook-15-G2

The key's randomart image is:

+---[RSA 2048]----+

| . . |

| o o |

| o B . . |

| .. O B E |

|....+ B S . |

|..o=o= o |

|..\*=X \* |

| \*oX O o |

|o \*++ . |

+----[SHA256]-----+

Terrafrom ssh into ec2 instance generate keys using ssh-keygen

#### 1.2 Verify the generated public key and private key

In the previous step, we have generated the key-pair which we are going to use for provisioning the EC2 instance. But let us take a look at the keys and how it looks.

If you remember in the previous step we have generated the keys at path /home/cloud/madhu/keys/aws we should see two key files over there -

1. aws\_key (private key)
2. aws\_key.pub (public key)

Terrafrom ssh into ec2 instance generate keys using ssh-keygen

We are going to use public key aws\_key.pub inside the terraform file to provision/start the ec2 instance.

###### 1.2.1 public key **aws\_key.pub**

Here is the content of the public key aws\_key.pub(you can open the file in any editor of your choice) -

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDbvRN/gvQBhFe+dE8p3Q865T/xTKgjqTjj56p1IIKbq8SDyOybE8ia0rMPcBLAKds+wjePIYpTtRxT9UsUbZJTgF+SGSG2dC6+ohCQpi6F3xM7ryL9fy3BNCT5aPrwbR862jcOIfv7R1xVfH8OS0WZa8DpVy5kTeutsuH5FMAmEgba4KhYLTzIdhM7UKJvNoUMRBaxAqIAThqH9Vt/iR1WpXgazoPw6dyPssa7ye6tUPRipmPTZukfpxcPlsqytXWlXm7R89xAY9OXkdPPVsrQA0XFQnY8aFb9XaZP8cm7EOVRdxMsA1DyWMVZOTjhBwCHfEIGoePAS3jFMqQjGWQd ajay@cloud-HP-ZBook-15-G2

*BASH*

###### 1.2.2 private key **aws\_key**

Here is the content of the private key aws\_key(you can open the file in any editor of your choice) -

-----BEGIN OPENSSH PRIVATE KEY-----

b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABFwAAAAdzc2gtcn

NhAAAAAwEAAQAAAQEA270Tf4L0AYRXvnRPKd0POuU/8UyoI6k44+eqdSCCm6vEg8jsmxPI

mtKzD3ASwCnbPsI3jyGKU7UcU/VLFG2SU4BfkhkhtnQuvqIQkKYuhd8TO68i/X8twTQk+W

j68G0fOto3DiH7+0dcVXx/DktFmWvA6VcuZE3rrbLh+RTAJhIG2uCoWC08yHYTO1CibzaF

DEQWsQKiAE4ah/Vbf4kdVqV4Gs6D8Oncj7LGu8nurVD0YqZj02bpH6cXD5bKsrV1pV5u0f

PcQGPTl5HTz1bK0ANFxUJ2PGhW/V2mT/HJuxDlUXcTLANQ8ljFWTk44QcAh3xCBqHjwEt4

xTKkIxlkHQAAA9CsiPUKrIj1CgAAAAdzc2gtcnNhAAABAQDbvRN/gvQBhFe+dE8p3Q865T

/xTKgjqTjj56p1IIKbq8SDyOybE8ia0rMPcBLAKds+wjePIYpTtRxT9UsUbZJTgF+SGSG2

dC6+ohCQpi6F3xM7ryL9fy3BNCT5aPrwbR862jcOIfv7R1xVfH8OS0WZa8DpVy5kTeutsu

H5FMAmEgba4KhYLTzIdhM7UKJvNoUMRBaxAqIAThqH9Vt/iR1WpXgazoPw6dyPssa7ye6t

UPRipmPTZukfpxcPlsqytXWlXm7R89xAY9OXkdPPVsrQA0XFQnY8aFb9XaZP8cm7EOVRdx

MsA1DyWMVZOTjhBwCHfEIGoePAS3jFMqQjGWQdAAAAAwEAAQAAAQB83JvPwSHWGtWhK4Yw

S6Tz2oDTJLQGT4o8Ns/tbmPJAXnRSMLp+/vpvgBxrUV6XE5xAvt/IZfwqOFH9AKNwRV8zV

2BLzaw7qQBPyYai9Ozzmana4V+dl4RgwffkX/GTruIPac7KKR+zLXy/aNVBACwhUJBVYDP

Dlf1g8hUOS5WcqpRpdzf3PWjAosC2sZUNtoU9wmMTOQMb96haWgdqQrdulzhBxMb7/hJQ5

q11gGdTboFM2l8yKMbRsba1OG/0JR2cmSNoK2PDL8GxskolwveGjvzsYPUS0iQsvjqAWQX

F3cW7lBq0eL23qkLFxAJxFQU4mB9yH5KCunU0K9am18hAAAAgAxkrreETLmI0snIX3uX+T

ySTNTb0G9kya0G3TGpkjvZT8Ump/L8ysleidqK2VYVEPre6vfTECqzWhBL1BVT/howuacG

5LeEkgS/F8oQJ19iEKT4LXxTumlFJqIj7+9jjnD5Z+gxi/ttJVTWvN/L4Ho9ZbZ6SoUBIB

hnb+tDP0+mAAAAgQDwFBx6E7dARqwOeN8byrmzsYmnr9a2jx4eXR1iDX7ZzPLjYoOCFiJP

3ahHhXSzPx5pJK4k8MRYUFNTjtl2K+cKgCZC75Dj5mfI/8l0pvmUrZ2e2GN1+ATUcQO7Dy

0j8124Bke0OkItBTfwQrsIOSyYzySOll88odoISNB2BdZsewAAAIEA6k+ljxLX1sfn8wOg

l9zgXqoT64tiQIJ720WSHuc5xoslgHVdjeunVy1eAaZMDURn8sbedlGLaebaGdVZwidR04

gpodkAZdkBm+tju2NRIxIhuDU02ddJFEGy/8lp+XqEm+YfbUpHrCSNOjYBDkdAp6umQFVq

jQi2RBtpIsNFikcAAAAacmFodWxAcmFodWwtSFAtWkJvb2stMTUtRzIB

-----END OPENSSH PRIVATE KEY-----

*BASH*

## 2. Use public key to start EC2 instance

Alright now we have the public key and the private key with us, let us create our terraform configuration file using the public key .i.e. aws\_key.pub

(\*Note - If this is your first time with terraform and you have not worked before then I would recommend you to go through an article on [*How to install terraform*](https://jhooq.com/install-terrafrom/) and [*How to create EC2 instance on AWS*](https://jhooq.com/terraform-ec2-instance-setup/))

Here is the main.tf -

provider "aws" {

region = "ap-south-1"

access\_key = ""

secret\_key = ""

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-01a4f99c4ac11b03c"

instance\_type = "t2.micro"

key\_name= "vcube-book"

vpc\_security\_group\_ids = [aws\_security\_group.main.id]

resource "aws\_security\_group" "main" {

egress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 0

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "-1"

security\_groups = []

self = false

to\_port = 0

}

]

ingress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 22

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "tcp"

security\_groups = []

self = false

to\_port = 22

}

]

}

resource "aws\_key\_pair" "deployer" {

key\_name = "vcube-book"

public\_key = " ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCihDAhkioSWqfTDzJSKag2lqQCPNh/hayPl+3TTogfK2+F8WiWIn3wXeP8F1xT1VzZY/s+nGMvT+zGGtAfNk8WWdw7orLY7LRYP3zYNzlDSc8U3bg+CA3B4POaQvK6ypUAc+SW2zaRMyLYaQpMsF/ZD3h7G6Ptr/7+A8xabEk2Lm4aHgXRLoqDOBnK99W1ri9i8Qc7HK3hgYdD3Bnc917NkNsKh/qaOpKmpslKkRWICrDIR6wFnZYVWTkizr85KAjuC7HKPilCNkntYoYA6HDFhPPPZSb53+E8pFnwxSQjJzks9q1B+viZ0BUUbLSBMYnlR9CFrkAS2JI5BvrtGodV admin@DESKTOP-0S3CU0K"

}

*BASH*

You can first verify the terraform configuration using the terraform plan and then finally you can apply it using terraform apply.

After applying the configuration you can verify the instance by going into the AWS console -

AWS terraform instance running after adding the public and private key

## 3. Use private key 'aws\_key' to SSH into EC2 instance

In the previous step, we have started the EC2 instance, now we need to connect to EC2 instance using the private key.

You can find the connect command from the aws console -

ssh commend to connect with ec2 machine using private key

Here is the SSH connect command for your reference-

(By default in the command you will see *.pem* extension in the private key file name but since we have created private with the name *aws\_key* so we need to remove *.pem* extension from the file)

ssh -i "aws\_key" ubuntu@ec2-18-185-22-181.eu-central-1.compute.amazonaws.com

*BASH*

after connecting to ec2 instance using the private key

## 4. Generate pem file from AWS console and use the file to SSH into EC2

In this step, we are going to AWS's ***key pair*** utility to generate the keys for us.

Login to your AWS console and in the search bar type is key pair

AWS key pair

Click on the Create Key pair

AWS create key pair

Now you need to supply the key name. You can choose the private key format .pem

Enter the key pair name for creating the aws key

As you will click on ***Create key pair*** you will be able to download the key and save it somewhere onto your disk.

Now you can simply use the key aws\_key.pem by specifying the key name inside your terraform file.

Here is an example of terraform script -

provider "aws" {

region = "ap-south-1"

access\_key = "xxxxxxxNXB2BYDxxxxx"

secret\_key = "xxxxxxxxxxGewdbOhnacm2QIMgcBxxxxx"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0767046d1677be5a0"

instance\_type = "t2.micro"

key\_name= "aws\_key"

vpc\_security\_group\_ids = [aws\_security\_group.main.id]

}

resource "aws\_security\_group" "main" {

egress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 0

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "-1"

security\_groups = []

self = false

to\_port = 0

}

]

ingress = [

{

cidr\_blocks = [ "0.0.0.0/0", ]

description = ""

from\_port = 22

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "tcp"

security\_groups = []

self = false

to\_port = 22

}

]

}

*BASH*

**Few points to pay attention for -**

1. You need to open port 22 for SSH
2. For opening port 22 you need to create a security group and attach the security group to aws instance

**Apply the Terraform configuration** You can simply run the following commands in the same sequence one after another -

1. terraform init
2. terraform plan
3. terraform apply

###### 4.1 Verify the EC2 instance

After provisioning/starting the EC2 instance, we need to verify two things -

1. EC2 instance running state
2. SSH into EC2 instance using the aws\_key.pem key

**1. Verify EC2 instance running state**

Go to your AWS console and look for the number of EC2 instances running. If you successfully started your EC2 instance then you will following under the EC2 dashboard of AWS -

Successfully running EC2 instance

**2. SSH into EC2 instance using the**aws\_key.pem**key**

So till now we have created the key as well started the AWS ec2 instance which is also up and running.

Now we need to ssh into the ec2 instance so that we can verify that the keys which we have generated are working fine.

Let's first goto the AWS console and select the EC2 instance which we have created. Then click on connect.

Connect the Ec2 instance

After you click on the connection you will be prompted with different options to connect -

1. EC2 Instance connect
2. Session Manager
3. SSH Client
4. EC2 Serial Console

We are going to choose the option ***SSH Client***

SSH Client connect option with EC2

Open your terminal and go to the location where you have saved aws\_key.pem key file. Because we need that private key file to connect with the EC2 instance.

Run the following command to change the permission of the file -

chmod 400 aws\_key.pem

*BASH*

Now use the following command to connect with the EC2 instance -

ssh -i "aws\_key.pem" ubuntu@ec2-52-58-111-83.eu-central-1.compute.amazonaws.com

*BASH*

After making the successful SSH connection you should see the following on you terminal -

Connect with EC2 instance using aws\_key