Terraform create EC2 Instance on AWS

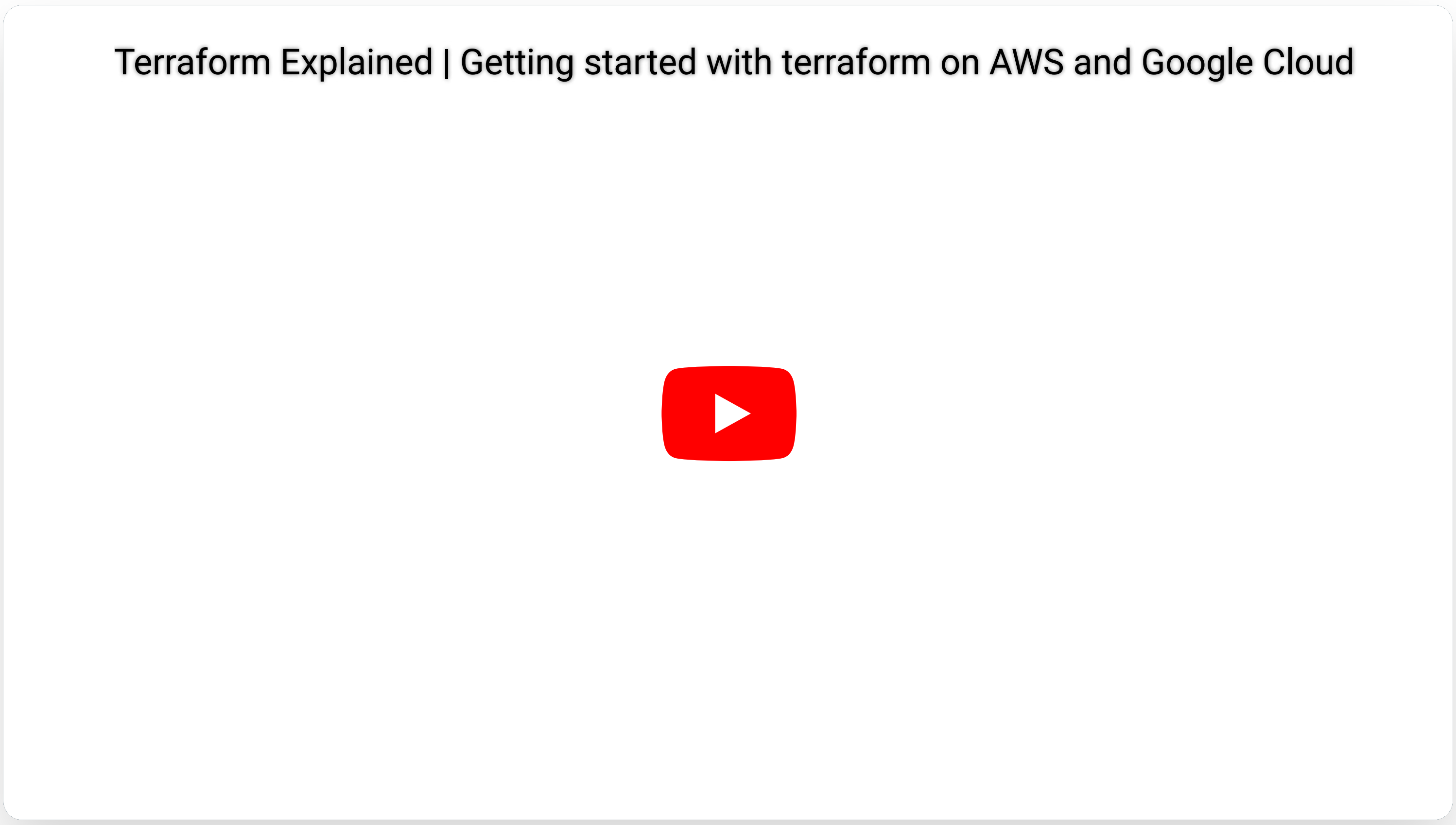
Mar 19, 2021 · 12 min read · Share on:

This guide will help you to **Create your first AWS EC2 using terraform**.

When it comes to IAC(Infrastructure As Code) Terraform is always the first choice of DevOps although there are many alternatives available in the market such as Ansible, Chef, Puppet, SaltStack, CloudFormation but due the fact that -

1. Terraform is really easy install
2. Terraform has very good API documentation
3. It is widely adopted in the DevOps community
4. Great support for a popular cloud service provider such as Google Cloud Platform, AWS.

(To know more on How to setup Virtual Machine on Google Cloud - Click Here)


Terraform Explained | Getting started with terraform on AWS and Google Cloud

1. The only prerequisite is - You must install Terraform before jumping to Google Cloud Setup.

Table of Content

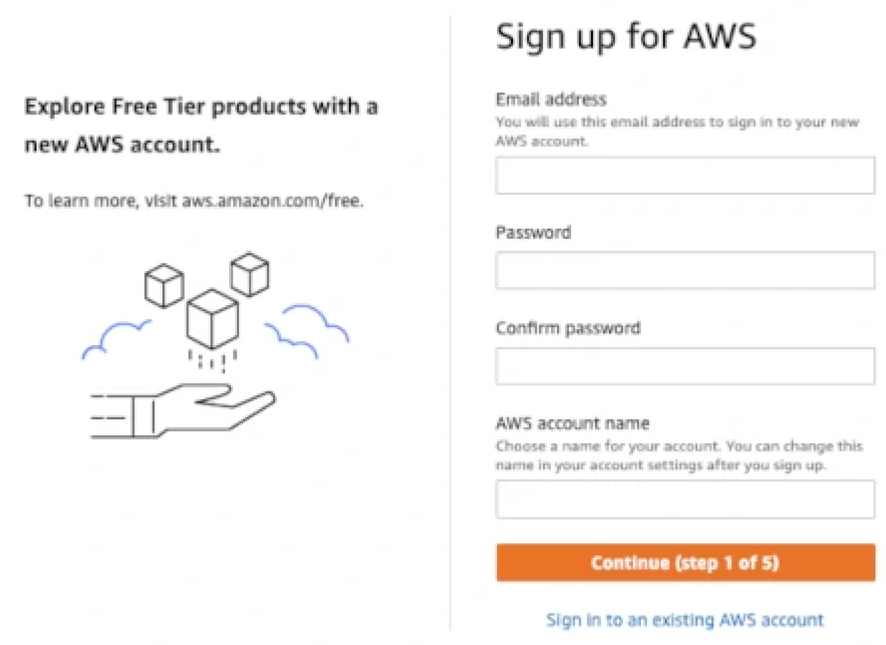
1. Setup AWS Account

As our aim of this article to setup an AWS EC2 instance the first step would be to create an AWS account.

If you are a beginner and want to learn the Terraform then AWS provides you free tier - *12 months or 750 Hours/month*, where you can experiment.

1.1 Sign up for AWS account

Goto https://aws.amazon.com/ and click on Complete sign-up


Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.

Sign up for AWS

Email address
You will use this email address to sign in to your new AWS account.

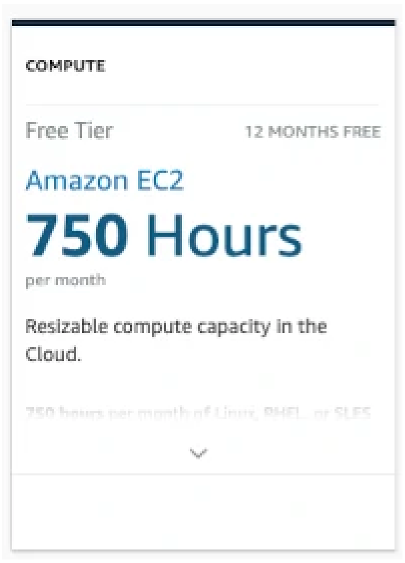
Password

Confirm password

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

AWS Sing up page

You need to choose **AMAZON EC2 Free Tier**. This tier is sufficient enough for learning purposes.


COMPUTE

Free Tier 12 MONTHS FREE

Amazon EC2

750 Hours
per month

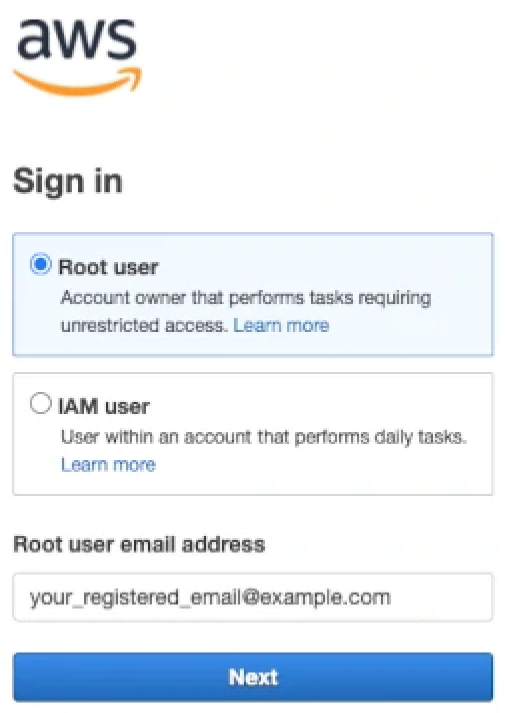
Resizable compute capacity in the Cloud.

750 hours per month of Linux, RHEL, or SLES

AWS EC2 Free tier

Amazon will ask for your Credit card number to complete the sign-up process, AWS will debit around 1$ so that they can verify your card details. Amazon will refund the amount after the authorization.

1.2 After Signup LogIn as ROOT user


Sign in

Root user
Account owner that performs tasks requiring unrestricted access. Learn more

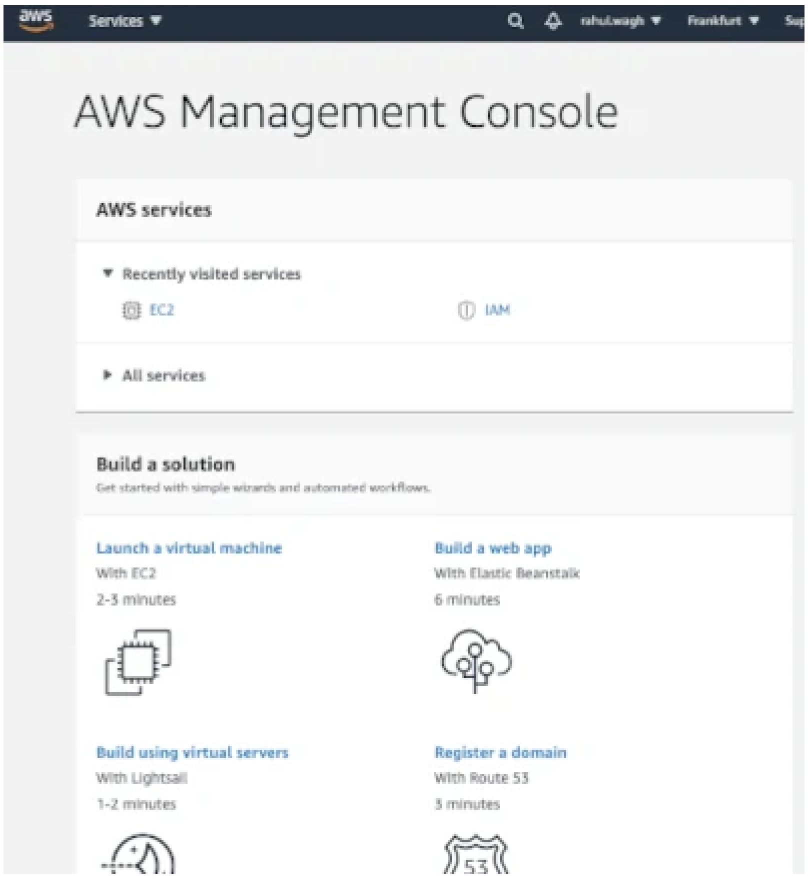
IAM user
User within an account that performs daily tasks. Learn more

Root user email address

your_registered_email@example.com

AWS console login as root user

Once you login into your AWS account you should see a dashboard, I know the dashboard can be little overwhelming for the first time user but do not worry we are gonna take one item at a time.


AWS Management Console

AWS services

Recently visited services
EC2 IAM

All services

Build a solution
Get started with simple wizards and automated workflows.

Launch a virtual machine
With EC2
2-3 minutes

Build a web app
With Elastic Beanstalk
6 minutes

Build using virtual servers
With Lightsail
1-2 minutes

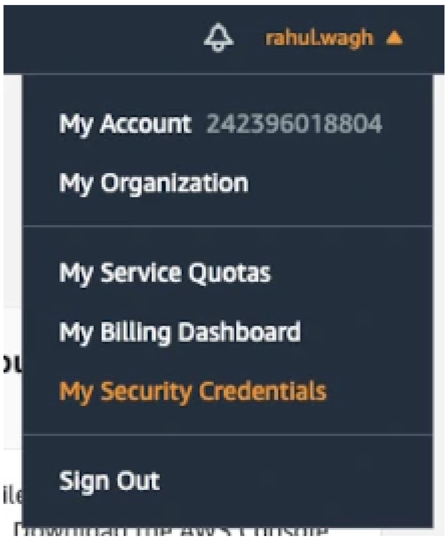
Register a domain
With Route 53
3 minutes

2.Generate Access keys (access key ID and secret access key)

Terraform installed on your Desktop/Laptop needs to communicate with AWS and to make this communication terraform needs to be authenticated.

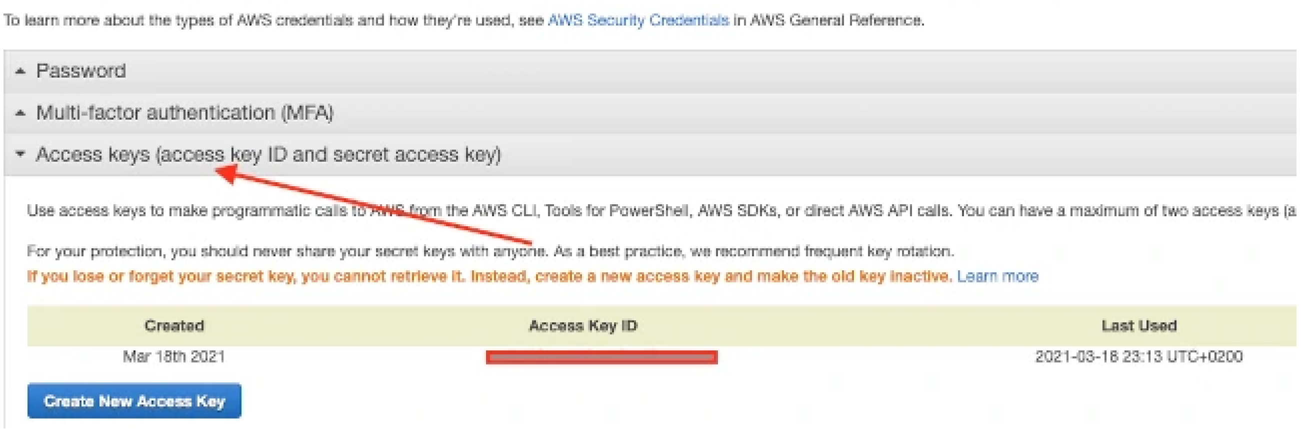
For authentication, we need to generate **Access Keys (access key ID and secret access key)**. These access keys can be used for making - programmatic calls to AWS from the AWS CLI, Tools for PowerShell, AWS SDKs, or direct AWS API calls.

1. Goto *My Security Credentials*



AWS security credentials

2. On *Your Security Credentials* page click on *Access keys (access key ID and secret access key)*


To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.

Password

Multi-factor authentication (MFA)

Access keys (access key ID and secret access key)

Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (a

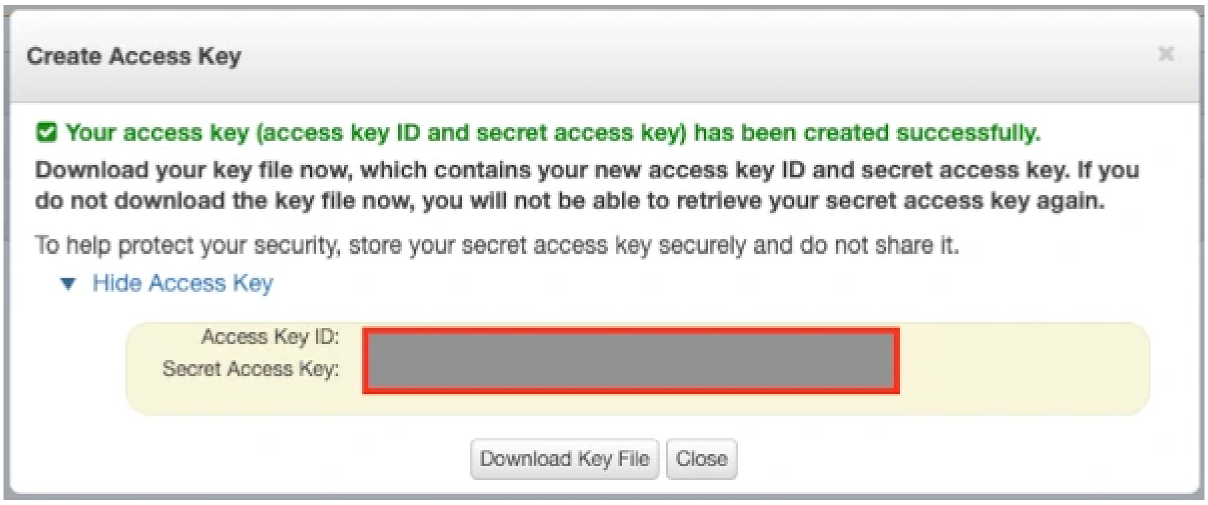
For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation.
If you lose or forget your secret key, you cannot retrieve it. Instead, create a new access key and make the old key inactive. Learn more

Created	Access Key ID	Last Used
Mar 18th 2021	[redacted]	2021-03-18 23:13 UTC+0200

AWS access key create new access key

3. Click on **Create New Access key**

4. Copy the *Access Key ID* and *Secret Access Key* (*Note:- You can view the Secret Access Key only once, so make sure to copy it.*)


Create Access Key

Your access key (access key ID and secret access key) has been created successfully.
Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.

To help protect your security, store your secret access key securely and do not share it.

Access Key ID: [redacted]
Secret Access Key:

AWS access key id and secret access key generated

3. Create your first Terraform infrastructure (main.tf)

Before we start writing terraform script, the first thing to learn over here is - *"You need to save your configuration with .tf extension"*

We will start by creating an empty `main.tf` file.

3.1 Provider

The first line of code in which we are going to write is *provider*.

We need to tell terraform which cloud provider we are going to connect .e.g - AWS, Google, or Azure

As this article is focused on **AWS**, so we are going to mention **AWS** as our provider.

Here is the basic syntax for the provider

```
YAML
resource "<PROVIDER>_<TYPE>" "<NAME>" {
 [CONFIG …]
}
```

1. "PROVIDER _ TYPE" - aws, google
2. "NAME" - You can define your name.

This is how our final `main.tf` will look like for AWS -

```
YAML
provider "aws" {
  region     = "eu-central-1"
  access_key = "XXXXXXXXXXXXXXXXXXXX"
  secret_key = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

Note: You can copy access_key and secret_key from Step-2

3.2 resource - "aws_instance"

So what do you mean by resource?

Resource - It is something that we are going to provision/start on AWS.

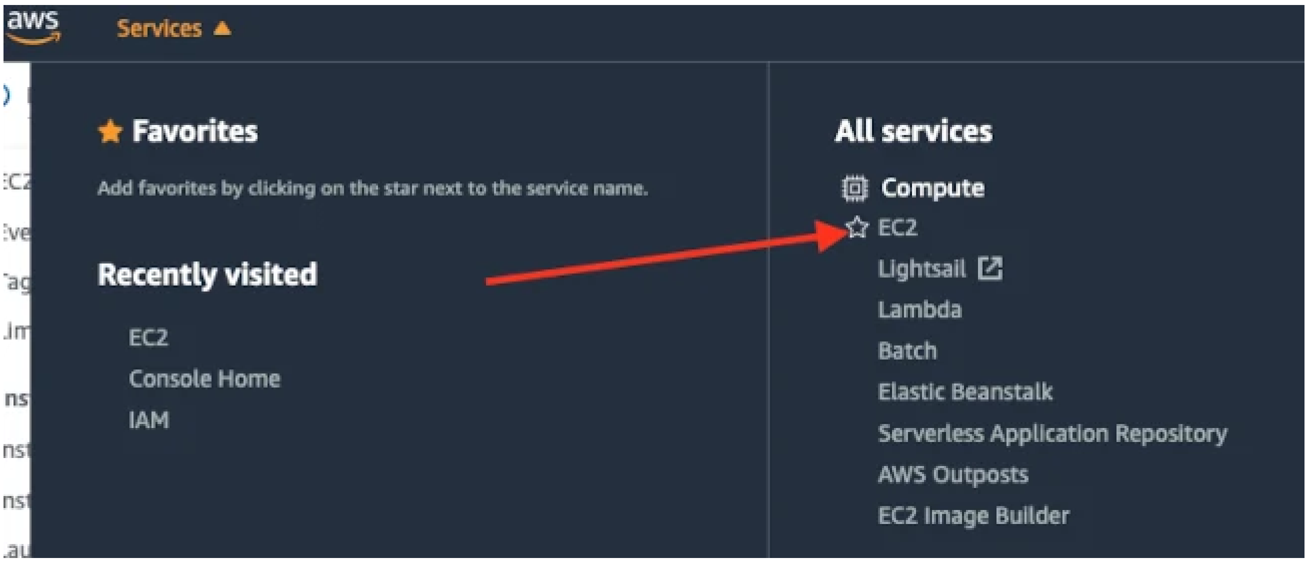
Now for this article, we are going to provision EC2 instance on AWS.

But before we provision the EC2 instance, we need to gather few points -

1. **ami** = you need to tell Terraform which AMI(Amazon Machine Image) you are going to use. Is it going to be Ubuntu, CentOS or something else
2. **instance_type** = Also based on your need you have to choose the instance_type and it can be t2.nano, t2.micro, t2. small etc.

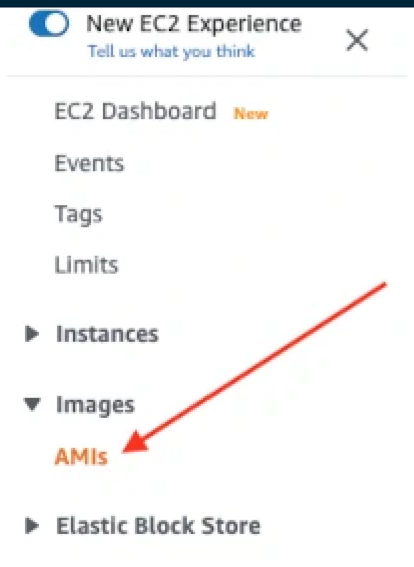
3.3 How to find ami(Amazon Machine Image)

1. To find the correct *ami* you need to Goto: `Services -> EC2`



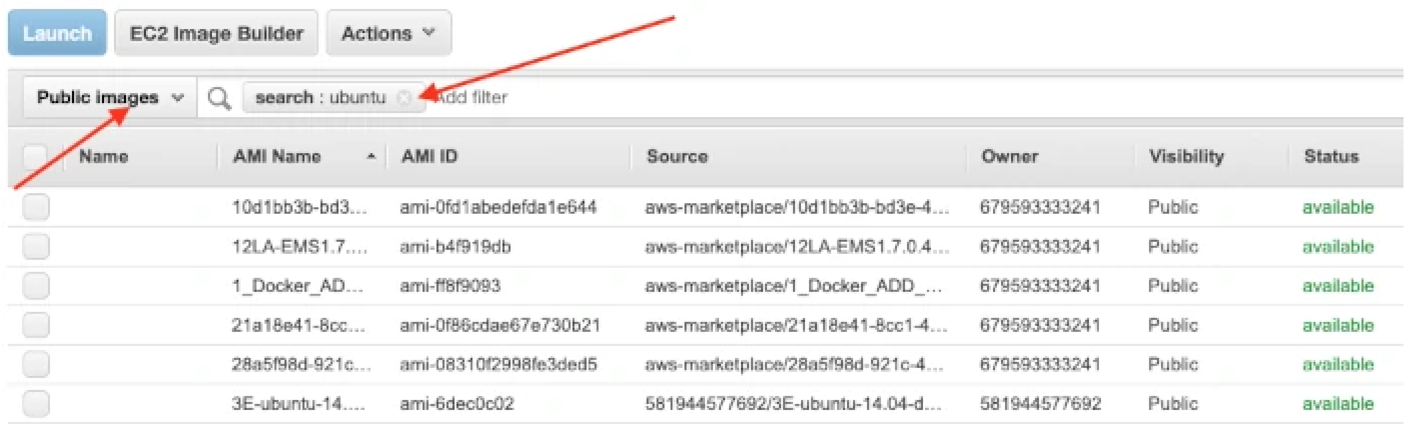
AWS services EC2

2. In the left Navigation you will find `Images -> AMIs`



AWS AMIs option in the left navigation

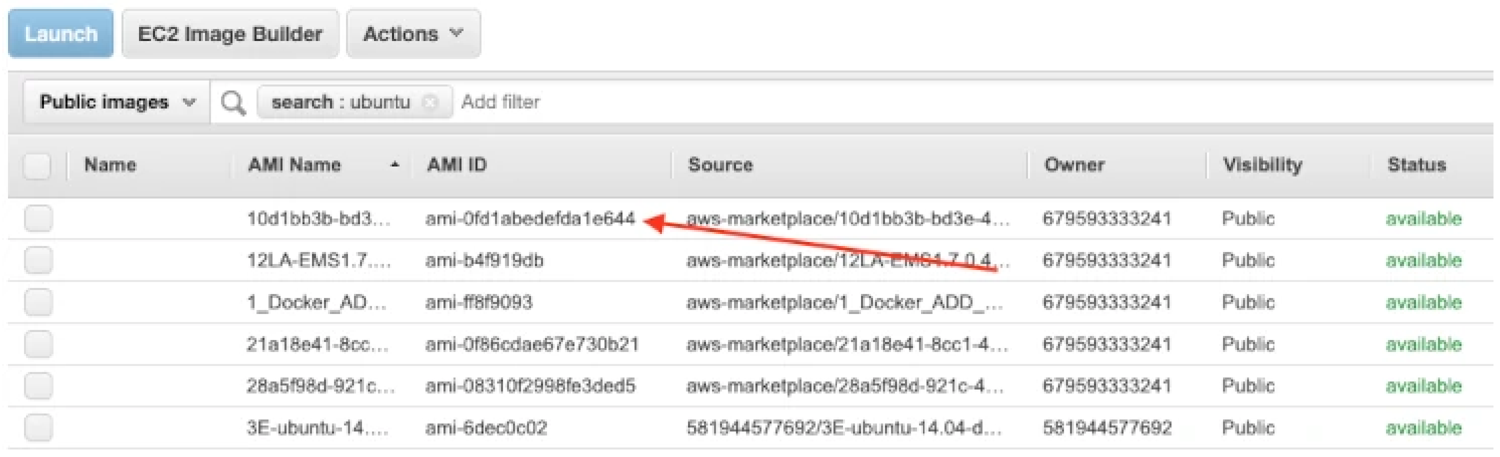
3. On the search menu click on `public images` and then type `Ubuntu`.



Name	AMI Name	AMI ID	Source	Owner	Visibility	Status
	10d1bb3b-bd3...	ami-0fd1abedefda1e644	aws-marketplace/10d1bb3b-bd3e-4...	679593333241	Public	available
	12LA-EMS1.7....	ami-b4f919db	aws-marketplace/12LA-EMS1.7.0.4...	679593333241	Public	available
	1_Docker_AD...	ami-ff8f9093	aws-marketplace/1_Docker_ADD_...	679593333241	Public	available
	21a18e41-8cc...	ami-0f86cdae67e730b21	aws-marketplace/21a18e41-8cc1-4...	679593333241	Public	available
	28a5f98d-921c...	ami-08310f2998fe3ded5	aws-marketplace/28a5f98d-921c-4...	679593333241	Public	available
	3E-ubuntu-14....	ami-6dec0c02	581944577692/3E-ubuntu-14.04-d...	581944577692	Public	available

AWS access key id and secret access key generated

4. Copy the AMI ID from the search result.



Name	AMI Name	AMI ID	Source	Owner	Visibility	Status
	10d1bb3b-bd3...	ami-0fd1abedefda1e644	aws-marketplace/10d1bb3b-bd3e-4...	679593333241	Public	available
	12LA-EMS1.7....	ami-b4f919db	aws-marketplace/12LA-EMS1.7.0.4...	679593333241	Public	available
	1_Docker_AD...	ami-ff8f9093	aws-marketplace/1_Docker_ADD_...	679593333241	Public	available
	21a18e41-8cc...	ami-0f86cdae67e730b21	aws-marketplace/21a18e41-8cc1-4...	679593333241	Public	available
	28a5f98d-921c...	ami-08310f2998fe3ded5	aws-marketplace/28a5f98d-921c-4...	679593333241	Public	available
	3E-ubuntu-14....	ami-6dec0c02	581944577692/3E-ubuntu-14.04-d...	581944577692	Public	available

AWS access key id and secret access key generated

3.4 How to find correct instance_type

You can find the correct `instance_type` by visiting this page.

Since I am looking for a very basic `instance_type` not production level instance, so I choose `t2.micro`

Here is the `aws_instance` configuration -

```bash
resource "aws_instance" "ec2_example" {
    ami = "ami-0767046d1677be5a0"
    instance_type = "t2.micro"
    tags = {
        Name = "Terraform EC2"
    }
}
```

4. terraform commands

Alright, now we have completed all the pre-requisites for provisioning our first `ec2` instance on the AWS.

4.1 terraform plan

The first command which we are going to run is -

```bash
terraform init
```

```
Initializing provider plugins...
- Reusing the previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v3.32.0...
- Installed hashicorp/aws v3.32.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary
```

The terraform init command is responsible for downloading all the dependencies which are required for the provider `AWS`.

Once you issue the `terraform init` command it will download all the provider's dependencies on your local machine.

4.2 terraform plan

This command will help you to understand how many resources you are gonna add or delete.

Here is the command -

```bash
terraform plan
```

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.ec2_example will be created
```

```
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
      + private_dns                  = (known after apply)
      + private_ip                   = (known after apply)
      + public_dns                   = (known after apply)
      + public_ip                    = (known after apply)
      + secondary_private_ips        = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tags                         = {
          + "Name" = "Terraform EC2"
        }
      + tenancy                      = (known after apply)
      + vpc_security_group_ids       = (known after apply)

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
...
```

As you can see the output of `terraform plan`, at the end it will show all the resources adde
and deleted.

4.3 terraform apply

This command will do some real stuff on AWS. Once you will issue this command, it will be going to connect to AWS and then finally going to provision AWS instance.

Here is the command -

```bash
terraform apply
```

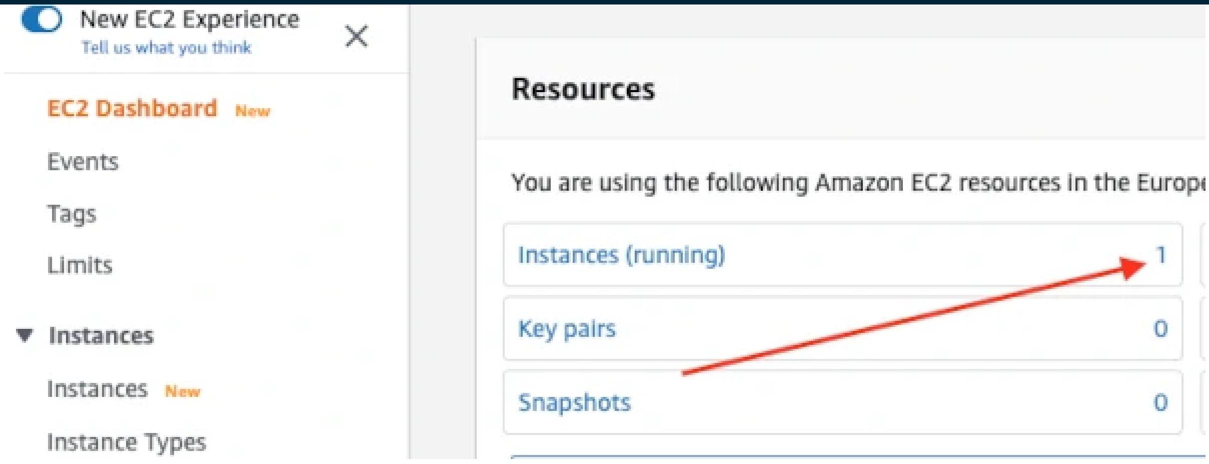
```bash
Plan: 1 to add, 0 to change, 0 to destroy.

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

  Enter a value: yes

aws_instance.ec2_example: Creating...
aws_instance.ec2_example: Still creating... [10s elapsed]
aws_instance.ec2_example: Still creating... [20s elapsed]
aws_instance.ec2_example: Still creating... [30s elapsed]
aws_instance.ec2_example: Creation complete after 33s [id=i-0a948ac635a2010f1]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

As you can see the log output has created `t2.micro` instance.

4.4 Verify the EC2 setup

Let's verify the setup by going back to AWS console.

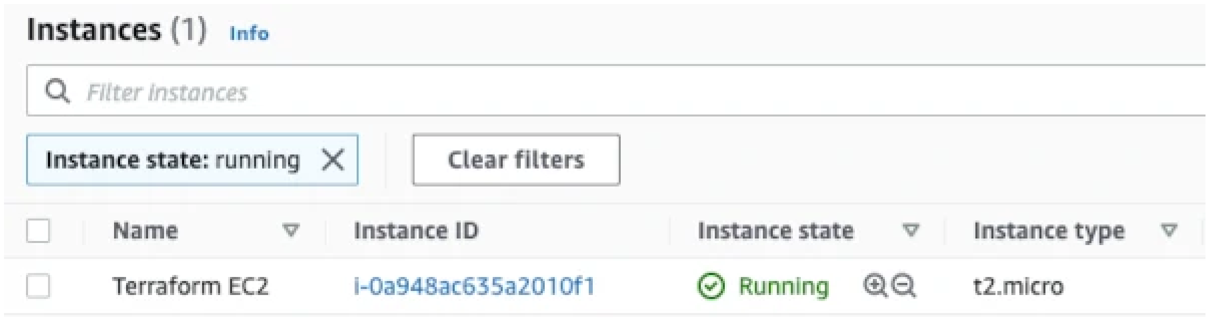
`Goto -> Services -> EC2` you should see 1 instance running.



You are using the following Amazon EC2 resources in the Europe

Instances (running)	1
Key pairs	0
Snapshots	0

AWS ec2 running instance



Name	Instance ID	Instance state	Instance type
Terraform EC2	i-0a948ac635a2010f1	Running	t2.micro

AWS ec2 running instance with more details

You can also see the **Tag name - Terraform EC2** which we mentioned in the terraform script.

4.4 terraform destroy

Now we have seen how to write your terraform script and how to provision your EC2 instance.

Let see how to remove or delete everything from AWS.

We are going to use the command -

```bash
terraform destroy
```

```
Do you want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.ec2_example: Destroying... [id=i-0a948ac635a2010f1]
aws_instance.ec2_example: Still destroying... [id=i-0a948ac635a2010f1, 10s elapsed]
aws_instance.ec2_example: Still destroying... [id=i-0a948ac635a2010f1, 20s elapsed]
aws_instance.ec2_example: Still destroying... [id=i-0a948ac635a2010f1, 30s elapsed]
aws_instance.ec2_example: Still destroying... [id=i-0a948ac635a2010f1, 40s elapsed]
aws_instance.ec2_example: Destruction complete after 40s

Destroy complete! Resources: 1 destroyed.
```

It will remove all the running EC2 Instances.

> The previous steps were very basic Terraform examples for setting up an EC2 instance. But in the actual project I think you need to do a little more than just basic ***EC2 Instance*** setup. The remaining part of this blog post will help you to do some advanced setup with your ***EC2 Instance*** using Terraform.
>
> **For example -**
>
> Setup up a custom startup script for an Amazon Elastic Compute Cloud (EC2) using Terraform
>
> Copy and Execute script from local machine to remote EC2 instance using file and remote-exec provisioner

5. Setup up a custom startup script for an Amazon Elastic Compute Cloud (EC2) using Terraform

To set up a custom startup script for an Amazon Elastic Compute Cloud (EC2) instance using Terraform, you can use the user_data attribute of the aws_instance resource. The user_data

Here is an example of how to use the `user_data` attribute to specify a custom startup script for an EC2 instance in Terraform. In this example -

1. First we will setup the EC2 instance
2. Secondly we are going to install Apache Web Server
3. At last we are going to set up a very basic HTML page and deploy it on the Apache web server

```bash
provider "aws" {
   region     = "eu-central-1"
   access_key = "<INSERT_YOUR_ACCESS_KEY>"
   secret_key = "<INSERT_YOUR_SECRET_KEY>"
}

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]

  user_data = <<-EOF
      #!/bin/sh
      sudo apt-get update
      sudo apt install -y apache2
      sudo systemctl status apache2
      sudo systemctl start apache2
      sudo chown -R $USER:$USER /var/www/html
      sudo echo "<html><body><h1>Hello this custom page built with Terraform User Data<
      EOF
}
```

6. Copy and Execute the script from local machine to remote EC2 instance using file and remote-exec provisioner

In case of long and complex scripts, you should use `file` as well as `remote-exec` provisioner. *(For more in-depth tutorial please refer to this blog post on - What is terraform provisioner)*

file provisioner- First we will use `file` provisioner to copy the file to the remote EC2 instance.

remote-exec provisioner- To execute the script copied using `file` provisioner

Here is an example where you can also use the `file provisioner` in Terraform to copy a script file from the local machine to the instance and execute it.

```
#main.tf

provider "aws" {
   region     = "eu-central-1"
   access_key = "<INSERT_YOUR_ACCESS_KEY>"
   secret_key = "<INSERT_YOUR_SECRET_KEY>"
}

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]

  # file provisioner -
  # It will copy the "startup.sh" to remote machine
  provisioner "file" {
    source      = "/home/rahul/Jhooq/startup.sh"
    destination = "/home/ubuntu/startup.sh"
  }

  # connection -
  # This block will be used for ssh connection to initiate the copy
  connection {
      type        = "ssh"
      host        = self.public_ip
      user        = "ubuntu"
      private_key = file("/home/rahul/Jhooq/keys/aws/aws_key")
      timeout     = "4m"
   }
}
```

```
    provisioner "remote-exec" {
     inline = [
       "chmod +x /home/ubuntu/startup.sh",
       "/home/ubuntu/startup.sh"
     ]
    }

resource "aws_security_group" "main" {
  egress = [
    {
      cidr_blocks      = [ "0.0.0.0/0", ]
      description      = ""
      from_port        = 0
      ipv6_cidr_blocks = []
...
```

Read More - **Terragrunt** -

1. How to use Terragrunt?

Posts in this Series

- Managing strings in Terraform: A comprehensive guide
- How to use terraform depends_on meta argument?
- What is user_data in Terraform?
- Why you should not store terraform state file(.tfstate) inside Git Repository?
- How to import existing resource using terraform import comand?
- Terraform - A detailed guide on setting up ALB(Application Load Balancer) and SSL?
- Testing Infrastructure as Code with Terraform?
- How to remove a resource from Terraform state?
- What is Terraform null Resource?
- In terraform how to skip creation of resource if the resource already exist?
- How to setup Virtual machine on Google Cloud Platform

- How to generate SSH key in Terraform using tls_private_key?
- How to fix-Terraform Error acquiring the state lock ConditionalCheckFiledException?
- Terraform Template - A complete guide?
- How to use Terragrunt?
- Terraform and AWS Multi account Setup?
- Terraform and AWS credentials handling?
- How to fix-error configuring S3 Backend no valid credential sources for S3 Backend found?
- Terraform state locking using DynamoDB (aws_dynamodb_table)?
- Managing Terraform states?
- Securing AWS secrets using HashiCorp Vault with Terraform?
- How to use Workspaces in Terraform?
- How to run specific terraform resource, module, target?
- How Terraform modules works?
- Terraform how to do SSH in AWS EC2 instance?
- What is terraform provisioner?
- Is terraform destroy needed before terraform apply?
- How to fix terraform error Your query returned no results. Please change your search criteria and try again?
- How to use Terraform Data sources?
- How to use Terraform resource meta arguments?
- How to use Terraform Dynamic blocks?
- Terraform - How to nuke AWS resources and save additional AWS infrastructure cost?
- Understanding terraform count, for_each and for loop?
- How to use Terraform output values?
- How to fix error configuring Terraform AWS Provider error validating provider credentials error calling sts GetCallerIdentity SignatureDoesNotMatch?
- How to fix Invalid function argument on line in provider credentials file google Invalid value for path parameter no file exists
- How to fix error value for undeclared variable a variable named was assigned on the command line?
- What is variable.tf and terraform.tfvars?
- How to use Terraform Variables - Locals,Input,Output
- Terraform create EC2 Instance on AWS

- Install terraform on Ubuntu 20.04, CentOS 8, MacOS, Windows 10, Fedora 33, Red hat 8 and Solaris 11

Search...

Categories

TERRAFORM 46 KUBERNETES 26 DOCKER 24 HELM-CHART 11
ANSIBLE 8 AWS 7 BLOGGING 6 SPRING-BOOT 5 SSL 5
QUARKUS 4 GITHUB 3 KUBESPRAY 3 PROMETHEUS-GRAFANA 3
VAGRANT 3

ALL CATEGORIES

Series

TERRAFORM 44 KUBERNETES 27 DOCKER 24 HELM-CHART 11
ANSIBLE 8 AWS 2 LINUX-COMMANDS 1

Tags

KUBERNETES 18 HELM-CHART 10 BLOGGING 4 QUARKUS 4
DOCKER 3 GITHUB 3 SSL 3 KUBESPRAY 2 SPRING-BOOT 2

Recent Posts

- 4 Ways to copy file from localhost to docker container
- Multiple commands execution in Docker Compose?
- Clone Public and Private Git Repositories with Ansible
- How to Limit Ansible playbook on Hosts, Groups and multiple hosts?
- Easy Fix for 'zsh command not found ansible' Error After Installing Ansible with Pip
- Demystifying Hosts, Inventory Roles, and Tasks
- Fixing-Unable to start service apache2 Job for apache2.service failed because the control process exited with error code?
- Why Ansible is the Ultimate Tool for DevOps Teams - A Beginner's Guide?

Rahul Wagh

Its all about Open Source and DevOps, here I talk about Kubernetes, Docker, Java, Spring boot and practices.

READ MORE