

# Introduction to

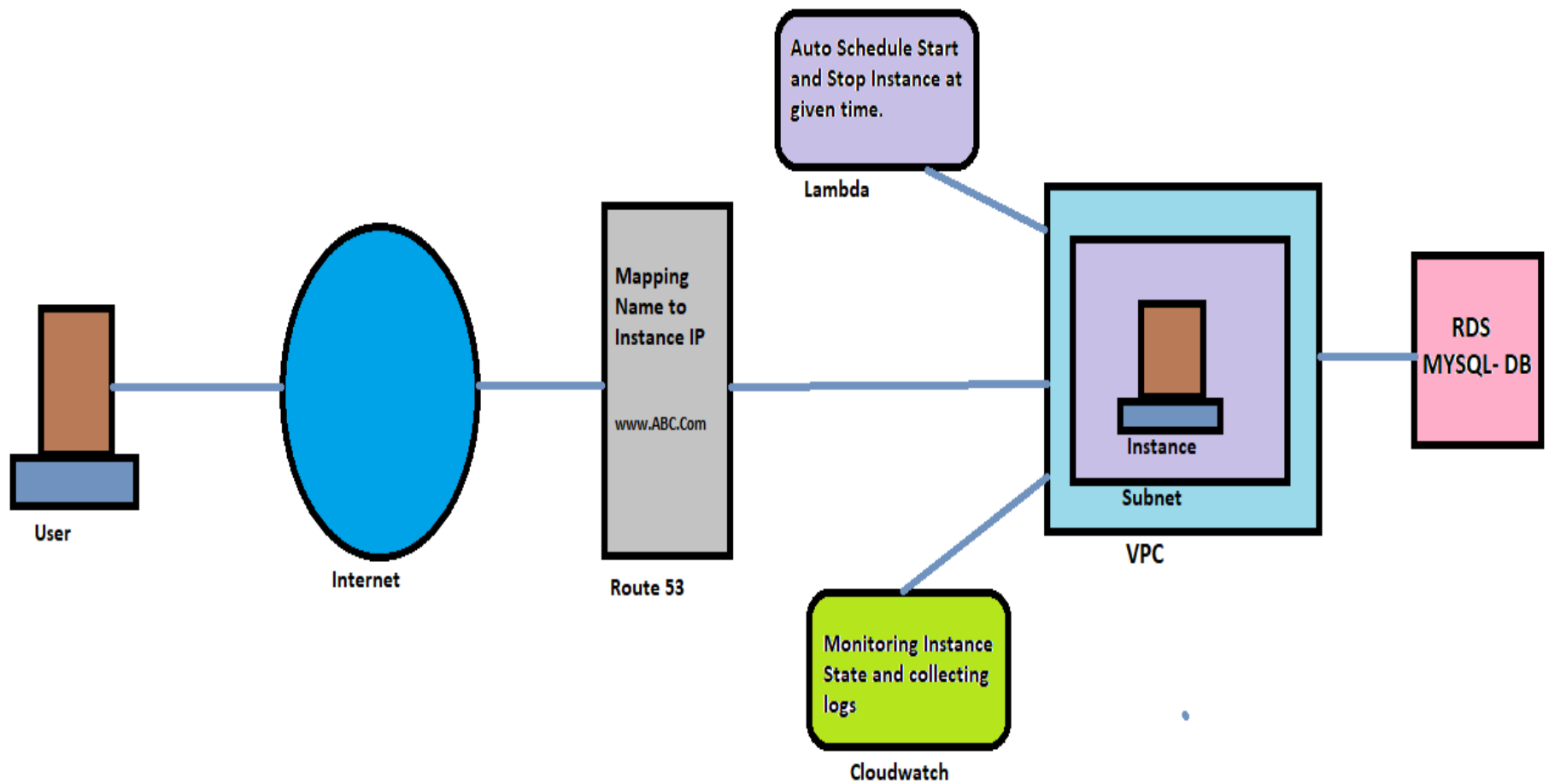


HashiCorp

# Terraform

# Infrastructure as Code





# DevOps Concepts: Pets vs. Cattle



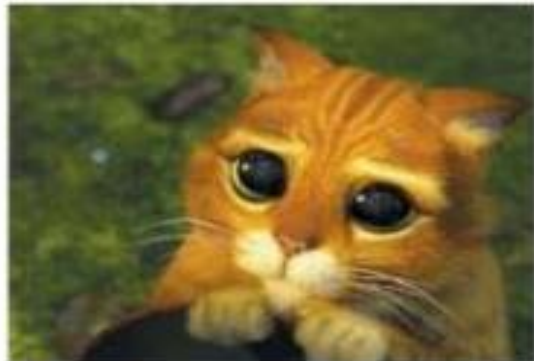
**pets**

**vs**



**cattle**

## Service Model



- Pets are given names like `pussinboots.cern.ch`
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health



- Cattle are given numbers like `vm0042.cern.ch`
  - They are almost identical to other cattle
  - When they get ill, you get another one
- 
- Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

# Mutable or immutable

- ✓ Traditional server environments are mutable, in that they are changed after they are installed. Administrators are always making tweaks or adding code.
- ✓ **Mutable- Repair concept**
- ✓ An immutable infrastructure is one in which servers are never modified after they're deployed. If something needs to be updated or changed, new servers are built afresh from a common template with the desired changes.
- ✓ **Immutable – Replacement concept**

## Differences based on Procedural and Declarative

- ✓ Chef and Ansible use a **procedural** style language where you write code that **specifies, step-by-step, how to achieve the desired end state.**
- ✓ The onus is on the user to determine the optimal deployment process.
- ✓ Procedural languages are more familiar to system admins who have backgrounds in scripting.
- ✓ Terraform, SaltStack, and Puppet use a **declarative** style where you write code that specifies the desired end state.
- ✓ The IaC tool itself then **determines how to achieve that state in the most efficient way possible.**
- ✓ Declarative tools are more familiar to users with a programming background.
- ✓ **Procedural – What to do and How to do**
- ✓ **Declarative - What to do**

# What is Terraform?

- Terraform is an open source “Infrastructure as Code” tool, created by HashiCorp.
- A *declarative* coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application. It then generates a plan for reaching that end-state and executes the plan to provision the infrastructure.
- Because Terraform uses a simple syntax, can provision infrastructure across multiple cloud and on-premises data centers, and can safely and efficiently re-provision infrastructure in response to configuration changes, it is currently one of the most popular infrastructure automation tools available. If your organization plans to deploy a hybrid cloud or multicolor environment, you’ll likely want or need to get to know Terraform.



# Infrastructure as Code (IaC)

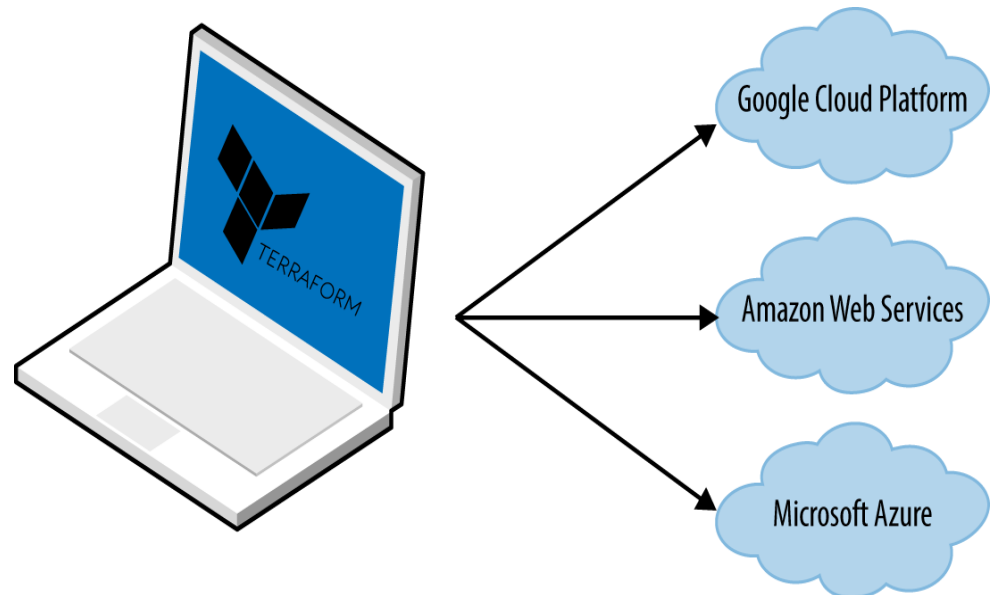
- IaC allows developers to codify infrastructure in a way that makes provisioning automated, faster, and repeatable. It's a key component of Agile and DevOps practices such as version control, continuous integration, and continuous deployment.
- Infrastructure as code can help with the following:
  - ✓ Improve speed:
  - ✓ Improve reliability:
  - ✓ Prevent configuration drift:
  - ✓ Support experimentation, testing, and optimization:

# Why Terraform?

**Open source:** Terraform is backed by large communities of contributors who build plugins to the platform.

**Platform agnostic:** Meaning you can use it with *any* cloud services provider. Most other IaC tools are designed to work with single cloud provider.

**Immutable infrastructure:** Terraform provisions *immutable infrastructure*, which means that with each change to the environment, the current configuration is replaced with a new one that accounts for the change, and the infrastructure is reprovisioned.



# Terraform modules

- Terraform *modules* are small, reusable Terraform configurations for multiple infrastructure resources that are used together. Terraform modules are useful because they allow complex resources to be automated with re-usable, configurable constructs.
- Writing even a very simple Terraform file results in a module. A module can call other modules—called *child modules*—which can make assembling configuration faster and more concise.
- Modules can also be called multiple times, either within the same configuration or in separate configurations.

# Terraform vs. Ansible

Terraform and Ansible are both Infrastructure as Code tools, but there are a couple significant differences between the two:

- While Terraform is purely a declarative tool, Ansible combines both declarative and *procedural* configuration.
- In procedural configuration, you specify the steps, or the precise manner, in which you want to provision infrastructure to the desired state.  
Procedural configuration is more work but it provides more control.
- Terraform is open source; Ansible is developed and sold by Red Hat.

## Difference Ansible and Terraform?



Which one to use? 🤔



**Both:** Infrastructure as a Code

### Mainly a configuration tool

- configure that infrastructure
- deploy apps
- install/update software

Mainly infrastructure  
provisioning tool

relatively new

more advanced in  
orchestration

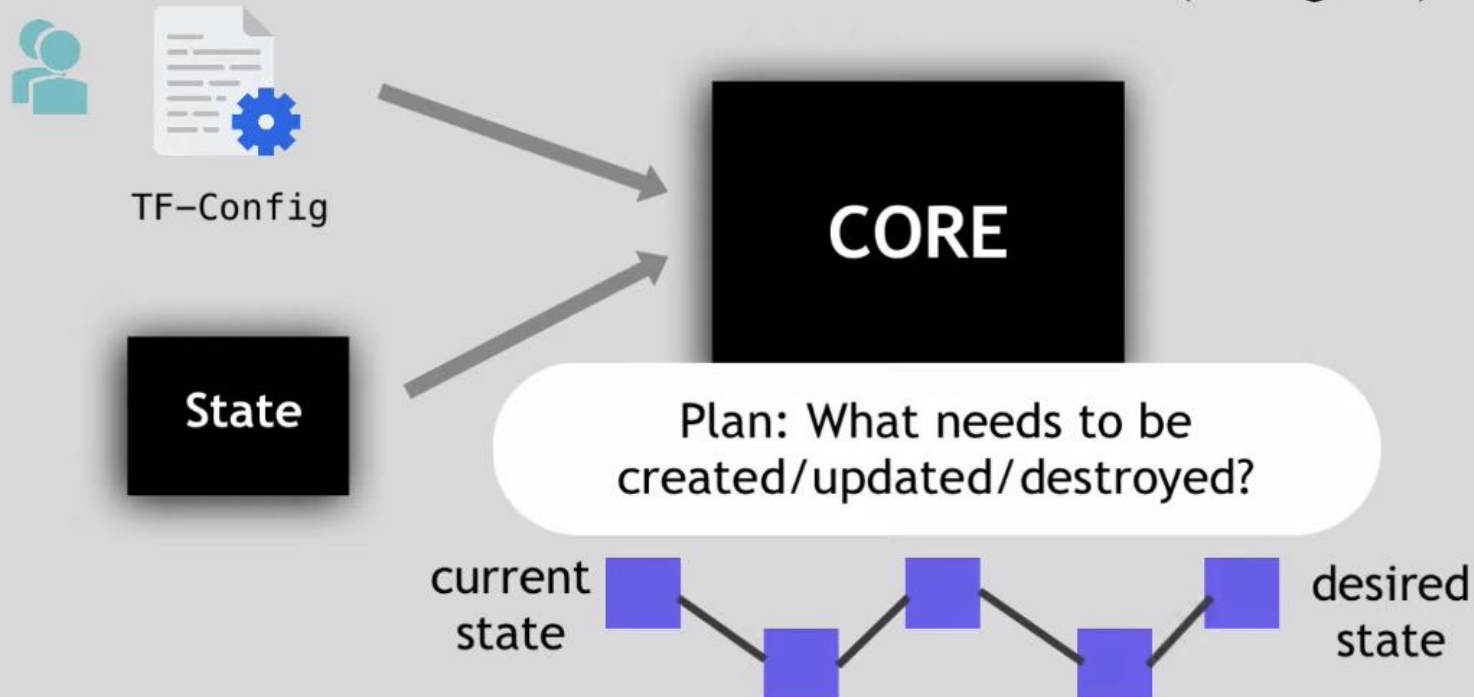
# Terraform vs. Kubernetes

- Kubernetes is an open source container orchestration system that lets developers schedule deployments onto nodes in a compute cluster and actively manages containerized workloads to ensure that their state matches the users' intentions.
- Terraform, on the other hand, is an Infrastructure as Code tool with a much broader reach, letting developers automate complete infrastructure that spans multiple public clouds and private clouds.
- Terraform can automate and manage Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or even Software-as-a-Service (SaaS) level capabilities and build all these resources across all those providers in parallel.

# Terraform Architecture

2 main components


► 2 input sources:



# Terraform Architecture

2 main components

## PROVIDERS:

 AWS | Azure | [IaaS]

Kubernetes | [PaaS]

Fastly | [SaaS]



TF-Config

State

CORE

AWS

K8s

Service



# Terraform Installation

Step 1- Download Binary:

```
# curl -O https://releases.hashicorp.com/terraform/0.12.13/terraform\_0.12.13\_linux\_amd64.zip
```

```
# echo $PATH
```

Step 2- Unzip the binary to /usr/bin

```
#unzip terraform_0.12.13_linux_amd64.zip -d /usr/bin/
```

Step 3- Check Terraform Version

```
# terraform -v
```

```
# terraform -help
```

Now create an IAM role to give administrator access and attach to the instance

## Creation of EC2 Instance Using Terraform:

Deploying AWS EC2 instances with Terraform is an excellent way to build infrastructure as code, and automate the provisioning, deployment and maintenance of resources to EC2 as well as custom solutions.

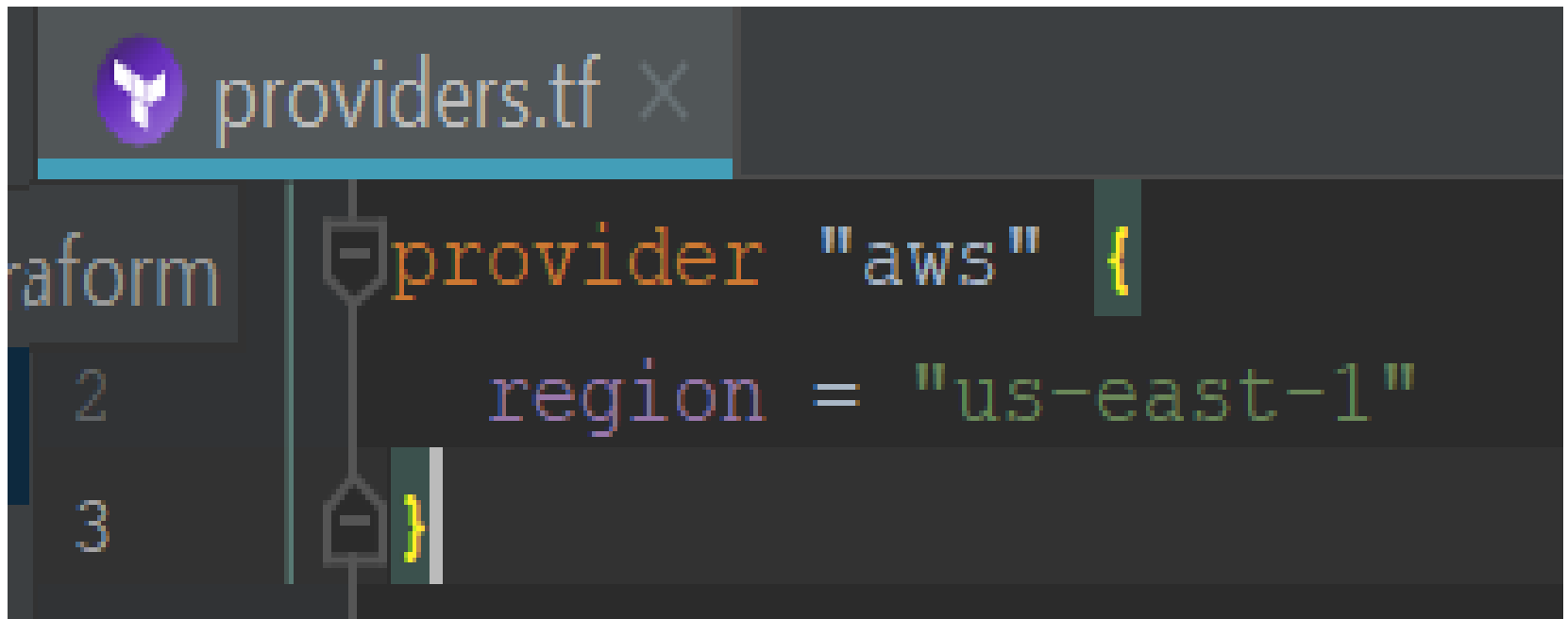
There are three actions covered here:

1. Configure
2. Initialize
3. Apply

## Creation of EC2 Instance Using Terraform:

The first step to using Terraform is typically to configure the providers (We're using AWS here) you want to use. Create a file called `providers.tf` & provide the provider information.

Provider can be any as per you: aws, azure, gcp.

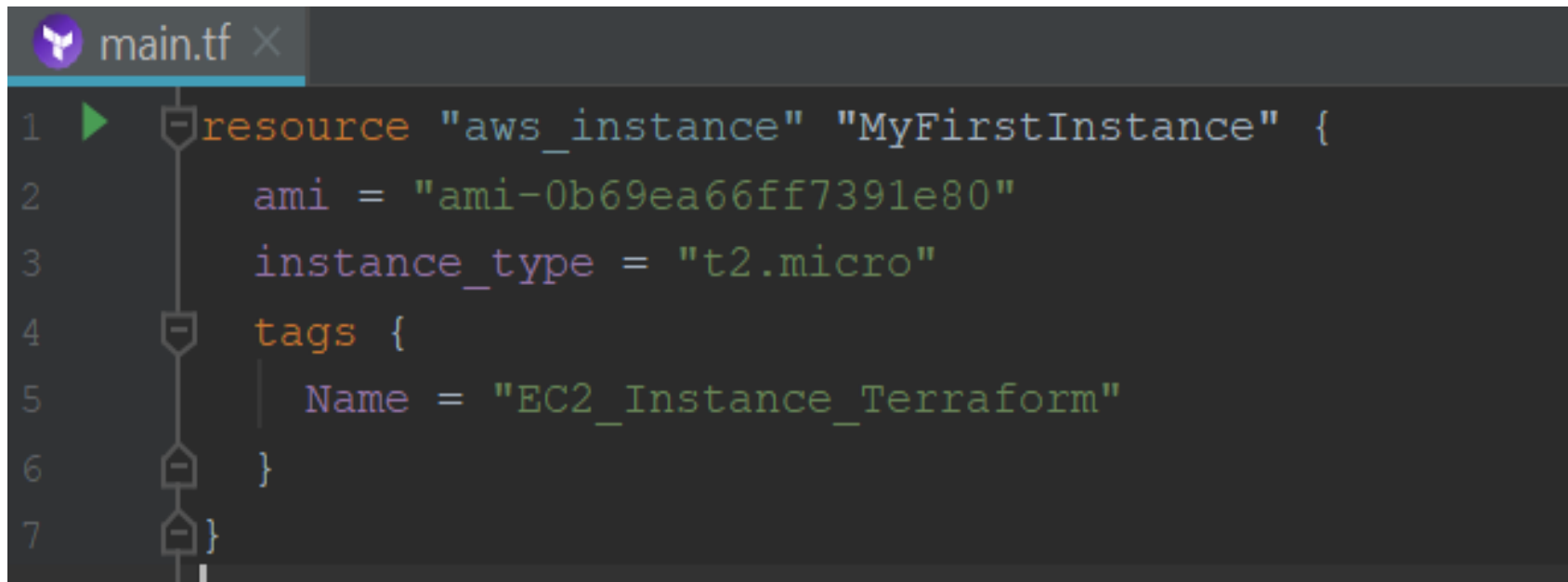


```
providers.tf x
provider "aws" {
  region = "us-east-1"
}
```

The screenshot shows a code editor with a tab titled 'providers.tf'. The code defines the AWS provider with the region 'us-east-1'. The code is as follows:

## Creation of EC2 Instance Using Terraform:

- Configuration basically consists of one or more *arguments* that are specific to that resource. We'll create one more file called `main.tf`
- So for creating the EC2 Instance, we need basically `ami`, `instance_type` & `tags`.
- Now we will go to terminal, go to that path where you created *main.tf* and run the **terraform init** command.
- To actually create the Instance, run the **terraform apply** command.



```
main.tf x
1  ► resource "aws_instance" "MyFirstInstance" {
2      ami = "ami-0b69ea66ff7391e80"
3      instance_type = "t2.micro"
4      tags {
5          Name = "EC2_Instance_Terraform"
6      }
7  }
```

# Important files and command

## Files

- 1) main.tf
- 2) variable.tf
- 3) terraform.tfstate
- 4) main.plan

## Commands

- 1) terraform init
- 2) terraform plan
- 3) terraform apply
- 4) Follow step 2 and 3 or directly type "terraform apply"
- 4) terraform show
- 5) terraform destroy

## Terraform in AWS Linux2 AMI

Create an IAM role to give administrator access and attach to the instance

```
# curl -O https://releases.hashicorp.com/terraform/0.12.13/terraform\_0.12.13\_linux\_amd64.zip
```

```
# echo $PATH
```

```
# unzip terraform_0.12.13_linux_amd64.zip -d /usr/bin/
```

```
# terraform -v
```

```
# mkdir terraform-vpc – cd terraform-vpc
```

Keep vpc-main.tf and vpc-variables.tf files here

```
# terraform init
```

```
# terraform plan
```

```
# terraform apply
```

# Installing Terraform on Windows



- To install Terraform, find the appropriate package for your system and download it.
- Terraform is packaged as a zip archive, So, after downloading Terraform, unzip the package.
- Terraform runs as a single binary named terraform.
- The final step is to make sure that the **terraform** binary is available on the PATH.
- Finally verify the installation of terraform with following command.
  - terraform
  - terraform --version

# Terraform in Windows

## Installation

- 1) Download terraform for windows.
- 2) Copy downloaded file –unzip it and keep in some folder (eg – c:\terraform)
- 3) Go to cmd – c:\terraform – type command –>  
> terraform --version
- 4) This PC – R.C. -- properties –Advanced system setting---Advanced – Environment variables –System variable - Path –Edit – New – just paste the path here(c:\terraform) –ok -- ok



## Terraform in Windows—Create VPC in AWS Cloud

- 1) In windows install AWS CLI
- 2) Open IAM –create user – administrator access –close –open –security credential – create access key – copy access key and secret key -
- 3) In windows open cmd – aws configure –provide credential
- 4) CMD - mkdir terraform-vpc – cd terraform-vpc
- 5) Keep vpc-main.tf and vpc-variables.tf files here
- 6) > terraform init
- 7) > terraform plan
- 8) > terraform apply