Training Course

Terraform

Course Overview

- **This course helps you to understand about Terraform and how to build a IaC by Terraform**
- **Prerequisites**
  - ✓ Some Linux system administration experience
  - ✓ Basic familiarity with Linux command line & windows powershell
  - ✓ Knowledge of popular development and scripting languages
  - ✓ Understand how to create your own network system on the AWS

Course Overview

- Module 1: Introduction to Infrastructure as Code
- Module 2: Terraform
- Module 3: Terraform with AWS

Module 1: Introduction IaC

What is Infrastructure as Code?

- Infrastructure as code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface.
- IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.
- It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure lifecycle.

Before have IaC

Prepare servers

- Setup servers
- Configure networking
- Create routable
- Install software

- Configure Software
- Install DB

Before have IaC

Prepare servers

- High human resources cost
- More effort time
- More human errors possible

Before have IaC

- Prepare servers
- Maintenance
- For multiple environments

- Update versions
- Deploy new release
- DB backups/updates
- Recover app
- Add/recover servers

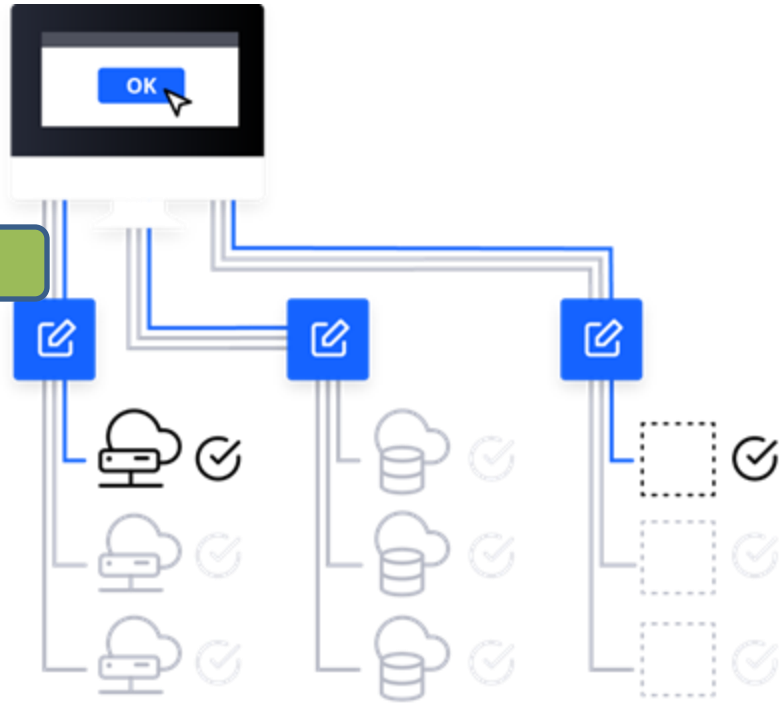
Before have IaC

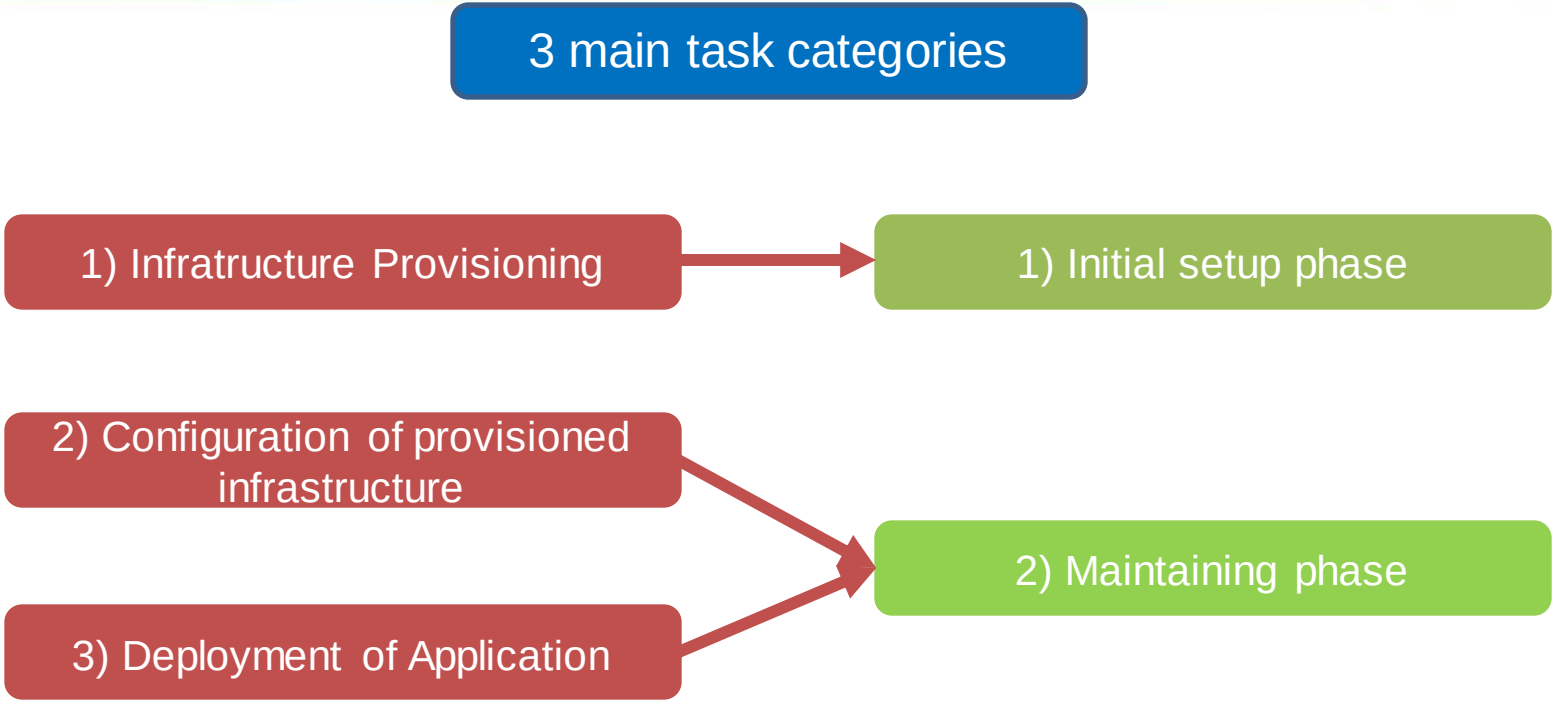
- Prepare servers
- Maintenance
- For multiple environments

After have IaC

Automate complete process with

Infrastructure as Code (IaC)




3 main task categories
1) Infratructure Provisioning
2) Configuration of provisioned infrastructure
3) Deployment of Application
1) Initial setup phase
2) Maintaining phase

Distinction of phases

1) Initial setup phase

- ✓ Provision infrastructure
- ✓ Configure infrastructure

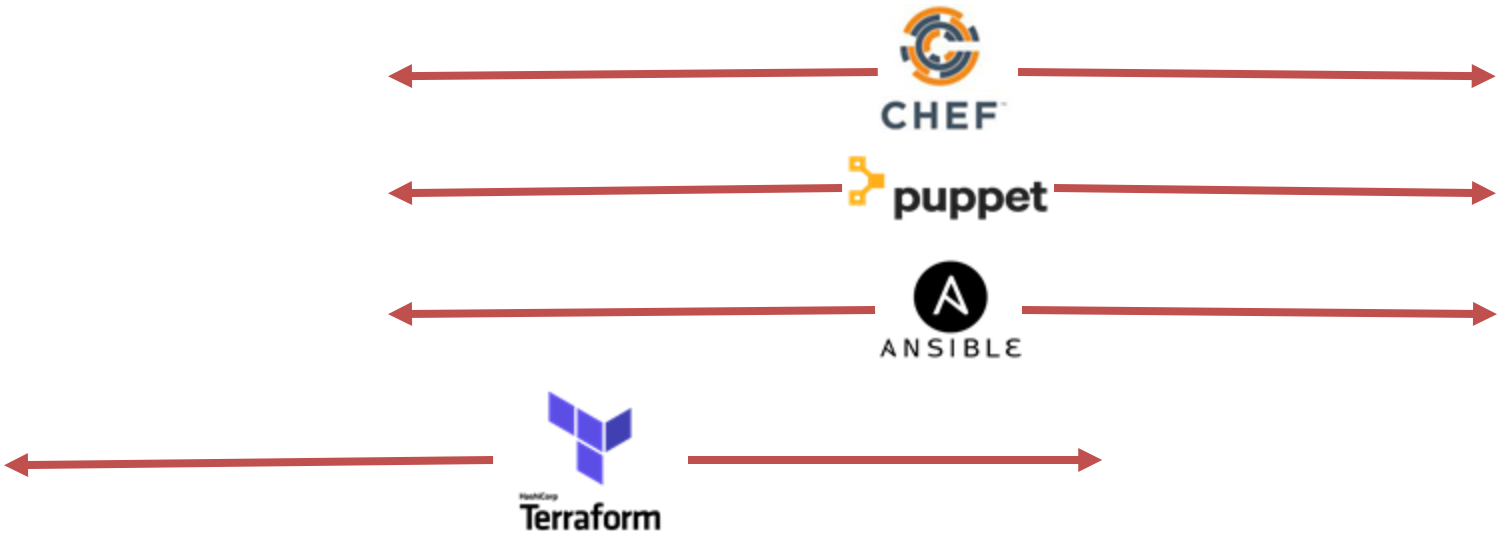
- ✓ Initial installation of software
- ✓ Initial configuration of software

2) Maintianing phase

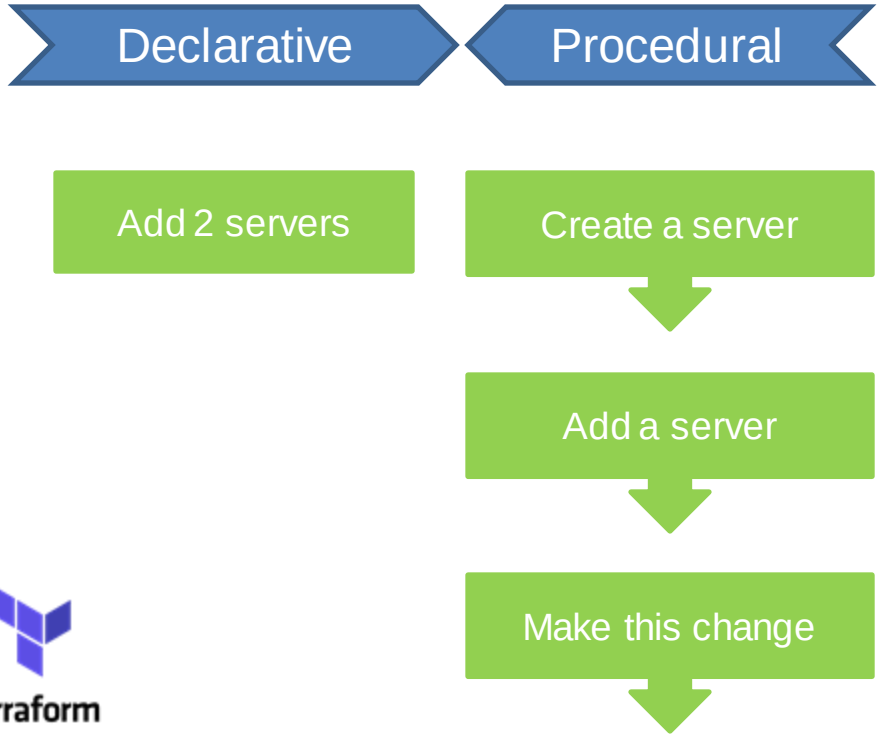
- ✓ Adjustment to infrastructure
- ✓ Add and remove servers

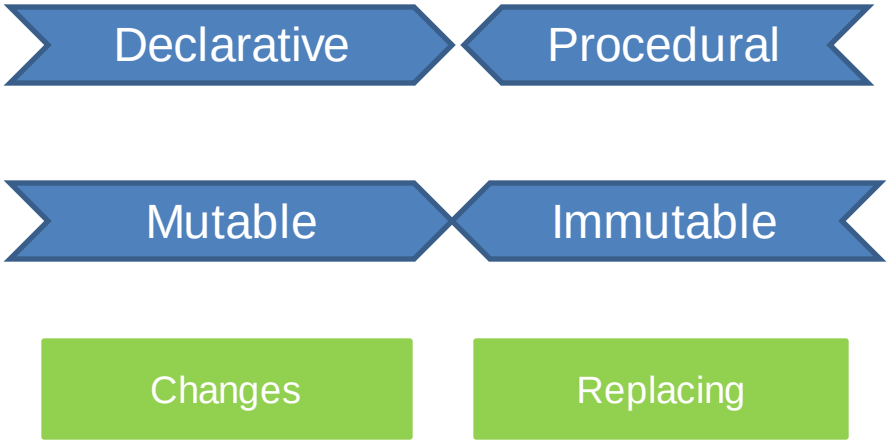
- ✓ Update software
- ✓ Re-configuration of software

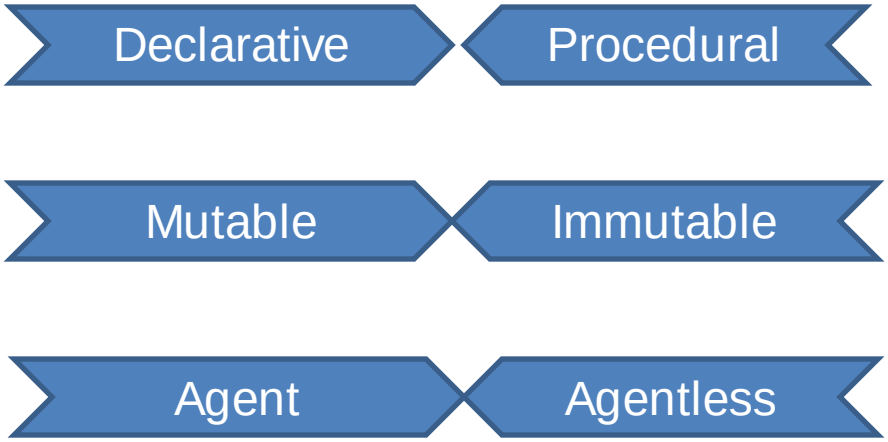
IaC tools



Initial Infrastructure Setup → Manage Infrastructure → Initial Application Setup → Manage Applications


Declarative
Procedural
Add 2 servers
Create a server
Add a server
Make this change


Declarative
Procedural
Mutable
Immutable
Changes
Replacing



CHEF

puppet

Terraform

ANSIBLE

Module 2: Terraform

HashiCorp Configuration Language

- The HashiCorp Configuration Language (HCL) is a configuration language authored by HashiCorp.
- HCL is used with HashiCorp cloud infrastructure automation tools, such as Terraform.
- The language was created with the goal of being both human and machine friendly.
- It is JSON compatible, which means it is interoperable with other systems outside of the Terraform product line.

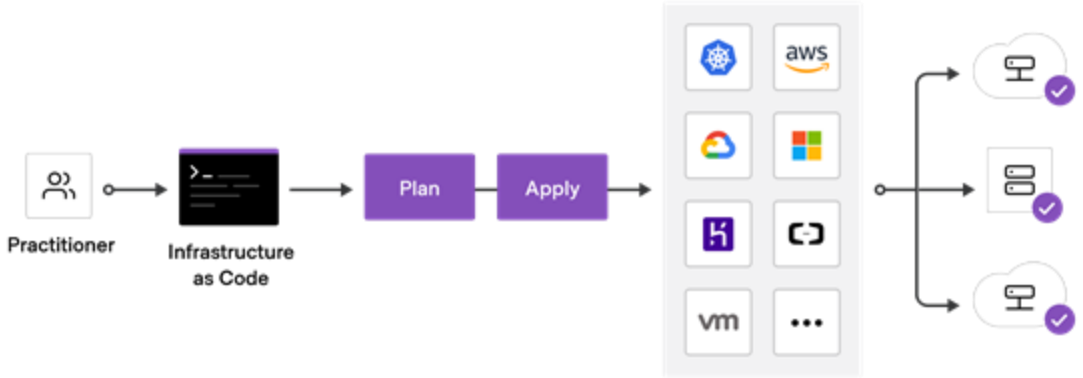
```
resource "aws_instance" "bastion_host" {
  ami             = "ami-0b215afe809665ae5"
  instance_type   = "t3.small"
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.public.id]
  key_name        = var.key_name
  tags = {
    Name = "Bastion_Host"
    Environment = var.ENV
  }
}
resource "aws_instance" "master" {
  ami             = "ami-0b215afe809665ae5"
  instance_type   = "t3.small"
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.private.id]
  key_name        = var.key_name
  tags = {
    Name = "Master"
    Environment = var.ENV
  }
  ebs_block_device {
    device_name = "/dev/sda1"
    volume_size = 20
  }
}
```

What is Terraform?

- Automate and manage your infrastructure
- Your platform
- And services that run on that platform

- ✓ Open source
- ✓ Declarative

>> Define what end result you want



Difference Ansible vs Terraform?

Both: Infrastructure as Code

Both automate:
Provisioning, configuring, managing the infrastructure

Mainly a configuration tool
- ✓ Configure that infrastructure
- ✓ Deploy apps
- ✓ Install/update software

Better: for configuring that infrastructure

- ✓ **Mainly infrastructure provisioning tool**
- ✓ **Can deploy apps**
- ✓ **More advanced in orchestration**

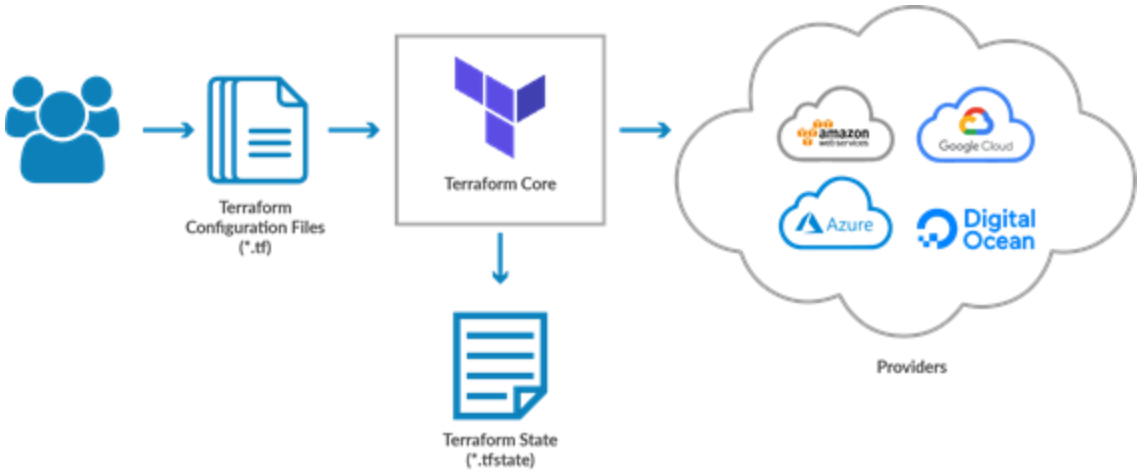
Better: for infrastructure

Easy with Terraform

- ✓ Create infrastructure
- ✓ Changes to infrastructure

Replicating infrastructure

- ✓ Dev
- ✓ Test
- ✓ Prod



Over 100 provider

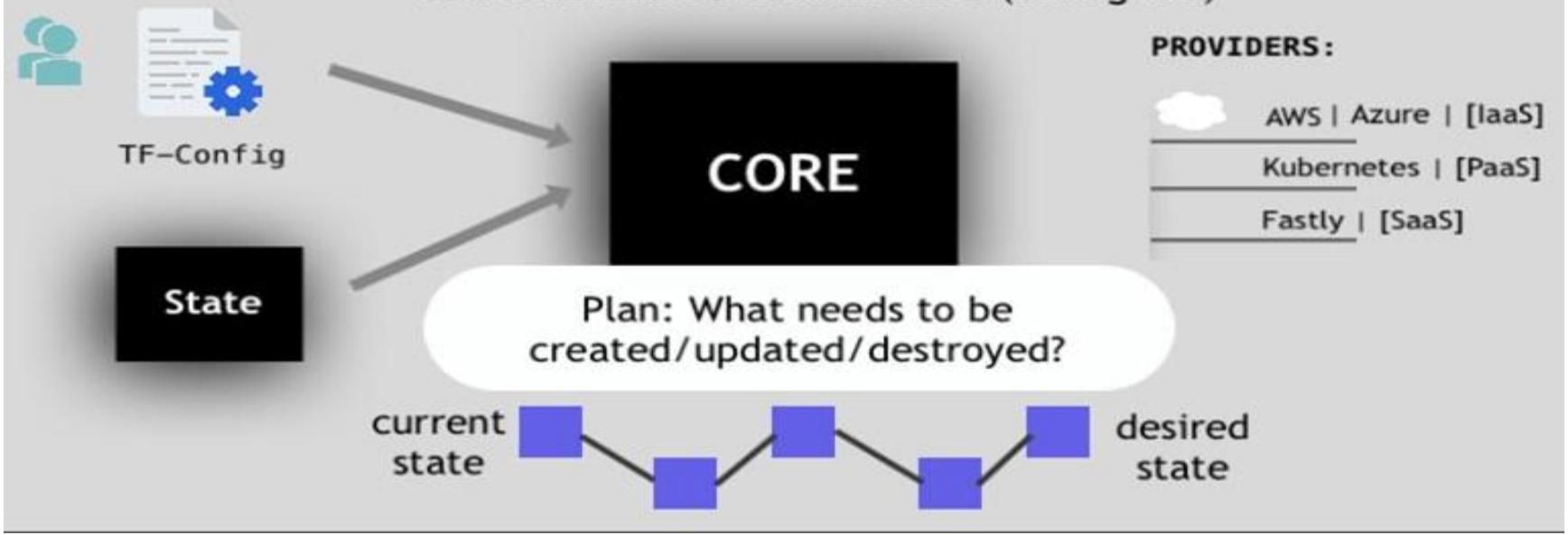
To over 1000 resources

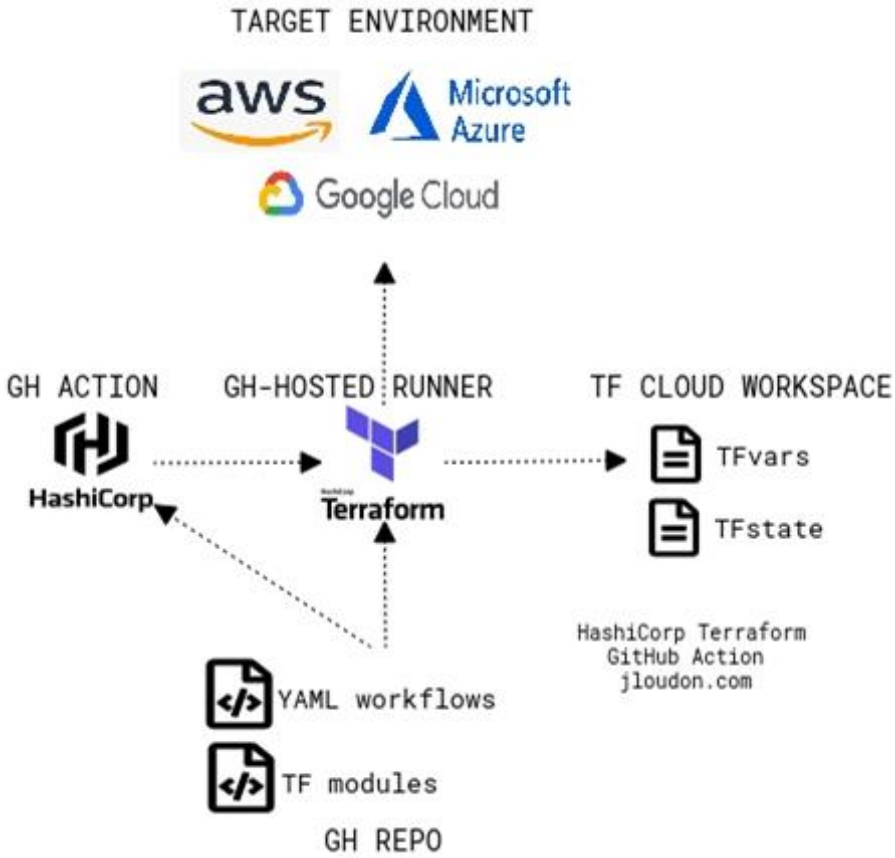
Terraform Architecture

2 main components

- 2 input sources:

current state VS desired state (config file)




TARGET ENVIRONMENT
aws
Microsoft Azure
Google Cloud
GH ACTION
GH-HOSTED RUNNER
TF CLOUD WORKSPACE
HashiCorp
Terraform
TFvars
TFstate
HashiCorp Terraform
GitHub Action
jloudon.com
YAML workflows
TF modules
GH REPO

Example Configuration Files

Define provider and resource

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "~> 3.27"
        }
    }
}
provider "aws" {
    profile = "default"
    region  = "ap-east-1"
}
```

```
resource "aws_instance" "node_1" {
  ami             = "ami-0b215afe809665ae5"
  instance_type   = "t3.small"
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.private.id]
  key_name        = var.key_name
  tags = {
    Name = "node_1"
    Environment = var.ENV
  }
}
```

Terraform Command

terraform init	Collect provider and installs
terraform fresh	Query infrastructure provider to get current state
terraform plan	Create an execution plan
terraform apply	Execute the plan
terraform output	Show resource info
terraform destroy	Destroy the resources/infrastructure

Variables

var.tf

```
variable "ENV" {
  default = "test"
}
```

instance.tf

```
resource "aws_instance" "node_1" {
  ami             = "ami-0b215afe809665ae5"
  instance_type   = "t3.small"
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.private.id]
  key_name        = var.key_name
  tags = {
    Name = "node_1"
    Environment = var.ENV
  }
}
```

Provision software

There are 2 way to provision software on your instances:

- You can build your own custom AMI and bundle your software with the image
  - ✓ Packer is a great tool to do this
- Another way to boot standardized AMIs, and then install the software in it you need
  - ✓ Using file uploads
  - ✓ Using automation tools like chef, puppet, ansible
  - ✓ Using remote exec

```
provisioner "file" {
    source = "script.sh"
    destination = "/opt/script.sh"
}
provisioner "remote-exec" {
      inline = [
        "chmod +x /opt/script.sh",
        "/opt/script.sh arguments"
      ]
}
```

output

- Terraform keeps attributes of all the resources you create

Ex: the aws_instance resource has the attribute public_ip

- Those attributes can be queried and outputted
- This can be useful just to output valuable information

or to feed information to external software

```
resource "aws_instance" "node_1" {
  key_name        = var.key_name
  tags = {
    Name = "node_1"
    Environment = var.ENV
  }
}
output "public_node_1" {
  value = aws_instance.node_1.public_ip
}
```

```
terraform output public_node_1
```

Remote State

- Terraform keeps the remote state of infrastructure
- It stores it in a file called terraform.tfstate
- There is also a backup of the previous state in terraform.tfstate.backup
- When you execute terraform apply, a new terraform.tfstate and backup is written
- This is how terraform keeps track of the remote state
- You can keep the terraform.tfstate in version control like git

Datasources

- For certain providers (like AWS), terraform providers datasources
- Datasources provide you with dynamic information
  - ✓ A lot of data is available by AWS in a structured format using their API
  - ✓ Terraform also exposes this information using data sources
- Examples:
  - ✓ List of AMIs
  - ✓ List of availability Zones

```
data "aws_ami" "amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

resource "aws_instance" "app" {
  ami = data.aws_ami.amazon_linux.id

  instance_type = var.instance_type
  # ...
}
```

Template

- The template provider can help creating customized configuration files
- You can build templates based on variables from terraform resource attributes
- The result is a string that can be used as a variable in terraform

init.tpl

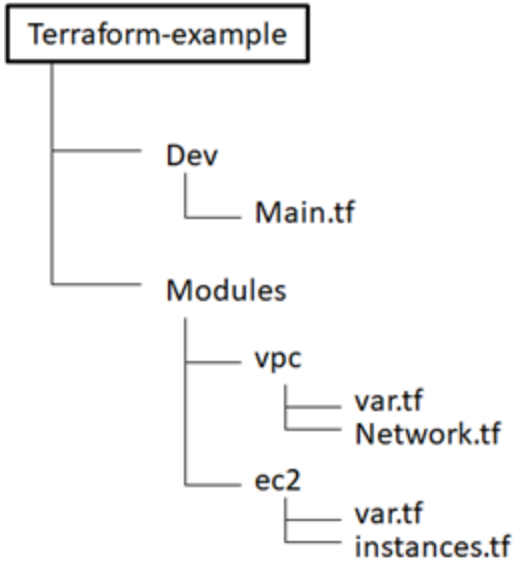
```
#!/bin/bash

echo "CONSUL_ADDRESS = ${consul_address}" > /tmp/iplist
```

```
data "template_file" "init" {
  template = "${file("${path.module}/init.tpl")}"
  vars = {
    consul_address = "${aws_instance.consul.private_ip}"
  }
}
```

Modules

- You can use modules to make your terraform more organized
  - ✓ Use third party modules
- Reuse parts of your code
- Modules from github



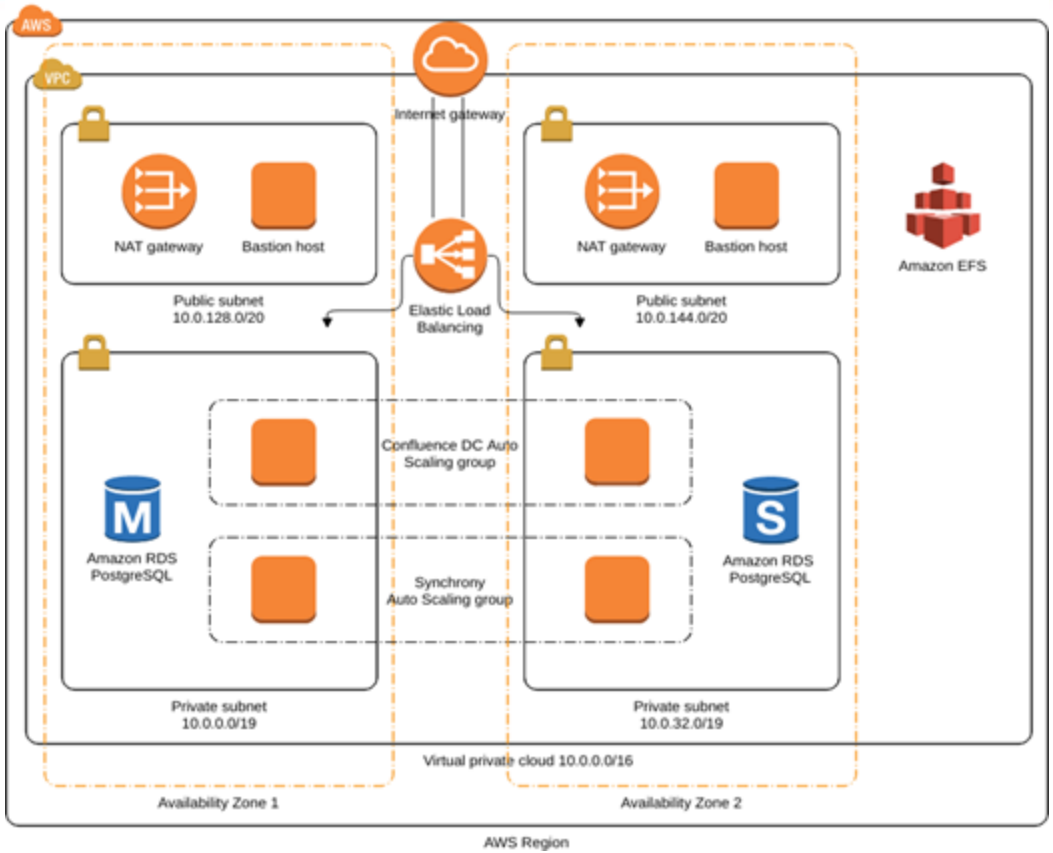

```
module "example" {
    source = "github.com/abc/example"
}
```

Module 3:
Terraform with AWS

Setup

- Install Terraform
- Install AWS CLI
- Add AWS Configure

AWS Architecture



VPC & subnet

```
resource "aws_vpc" "VLAN" {
  cidr_block           = "192.168.0.0/16"
  instance_tenancy     = "default"
  enable_dns_support   = "true"
  enable_dns_hostnames = "true"
  enable_classiclink   = "false"
  tags = {
    Name = "Training"
    Environment = var.ENV
  }
}
```

```
resource "aws_subnet" "public_subnet" {
    vpc_id                  = aws_vpc.VLAN.id
    cidr_block              = "192.168.1.0/24"
    map_public_ip_on_launch = "true"
    availability_zone       = "ap-east-1a"
    tags = {
        Name = "public_subnet"
    }
}
resource "aws_subnet" "private_subnet" {
    vpc_id                  = aws_vpc.VLAN.id
    cidr_block              = "192.168.2.0/24"
    map_public_ip_on_launch = "false"
    availability_zone       = "ap-east-1c"
    tags = {
        Name = "private_subnet"
    }
}
```

Internet Gateway & Route Table

```
resource "aws_internet_gateway" "internet_gateway" {
  vpc_id = aws_vpc.VLAN.id
  tags = {
    Name = "internet_gateway"
    Environment = var.ENV
  }
}
resource "aws_route_table" "route_table" {
  vpc_id = aws_vpc.VLAN.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.internet_gateway.id
  }
  tags = {
    Name = "route_table"
    Environment = var.ENV
  }
}
resource "aws_route_table_association" "route_table_association" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.route_table.id
}
```

NAT

```
resource "aws_eip" "nat" {
    vpc = true
}
resource "aws_nat_gateway" "nat_gateway" {
    allocation_id = aws_eip.nat.id
    subnet_id     = aws_subnet.public_subnet.id
    depends_on    = [aws_internet_gateway.internet_gateway]
}
resource "aws_route_table" "main_private" {
    vpc_id = aws_vpc.VLAN.id
    route {
        cidr_block      = "0.0.0.0/0"
        nat_gateway_id  = aws_nat_gateway.nat_gateway.id
    }
    tags = {
        Name = "main_private"
    }
}
resource "aws_route_table_association" "route_table_association_1" {
    subnet_id       = aws_subnet.private_subnet.id
    route_table_id  = aws_route_table.main_private.id
}
```

Security Group

```
resource "aws_security_group" "public" {
  vpc_id = aws_vpc.VLAN.id
  name   = "allow-all"
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "Training"
  }
}
```

Instance

```
resource "aws_instance" "test" {
    ami           = "ami-0b215afe809665ae5"
    instance_type = "t3.micro"

    subnet_id      = aws_subnet.public_subnet.id
    security_groups = [aws_security_group.security.id]
    key_name        = aws_key_pair.key.key_name
    tags = {
        Name = "instance"
    }
    ebs_block_device {
        device_name = "/dev/sda1"
        volume_size = 20
    }
}
```

```
resource "aws_ebs_volume" "ebs_volume" {
    availability_zone = "ap-east-1a"
    size              = 20
    type              = "gp2"
    tags = {
        Name = "volume"
    }
}
resource "aws_volume_attachment" "ebs_volume_attachment" {
    device_name = "/dev/xvdh"
    volume_id   = aws_ebs_volume.ebs_volume.id
    instance_id = aws_instance.test.id
}
```

Practice

Build a system on AWS using Terraform include:

- VPC, 2 subnet (private, public)
- NAT, IGW, Route Table
- Security Group
- Instance
- EBS volume attach to instance

Thank you