HashiCorp Certified Terraform Associate 2024

https://www.linkedin.com/in/lokeshkumar-aws-devops

https://www.linkedin.com/in/lokeshkumar-aws-devops

IAC Tools

DevOps = Developers

Exploring Toolsets

There are various types of tools that can allow you to deploy infrastructure as code :

- **Terraform**
- **CloudFormation**
- **Heat**
- **Ansible**
- **SaltStack**
- **Chef, Puppet and others**

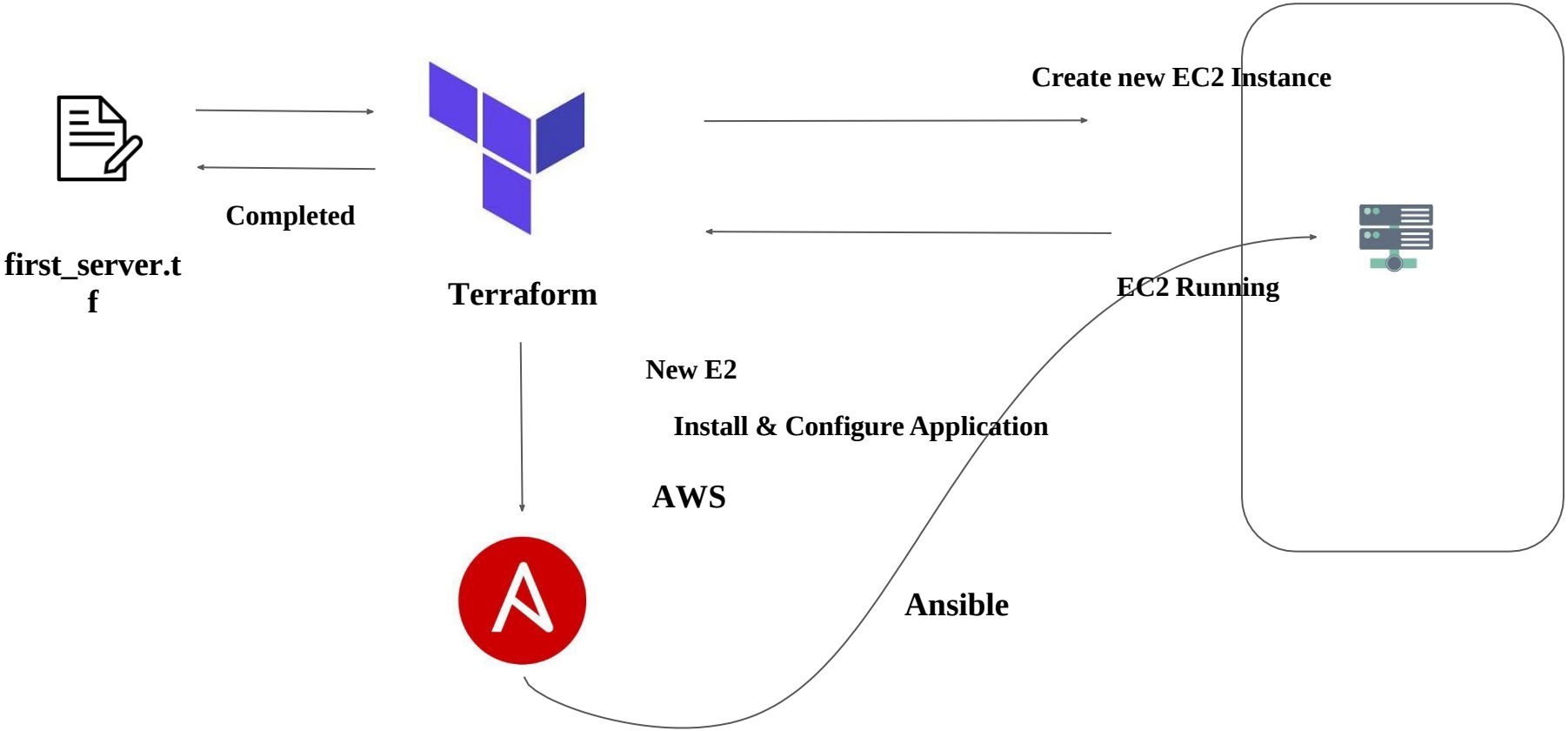
Configuration Management vs Infrastructure Orchestration

Ansible, Chef, Puppet are configuration management tools which means that they are primarily designed to install and manage software on existing servers.

Terraform, CloudFormation are the infrastructure orchestration tools which basicallymeans they can provision the servers and infrastructure by themselves.

Configuration Management tools can do some degree of infrastructure provisioning, butthe focus here is that some tools are going to be better fit for certain type of tasks.

IAC & Configuration Management = Friends



Which tool to choose ?

Question remains on how to choose right IAC tool for the organization

i) Is your infrastructure going to be vendor specific in longer term ? Example AWS.

ii) Are you planning to have multi-cloud / hybrid cloud based infrastructure ?

iii) How well does it integrate with configuration management tools ?

iv) Price and Support

Terraform

i) Supports multiple platforms, has hundreds of providers.

ii) Simple configuration language and faster learning curve.

iii) Easy integration with configuration management tools like Ansible.

iv) Easily extensible with the help of plugins.

v) Free !!!

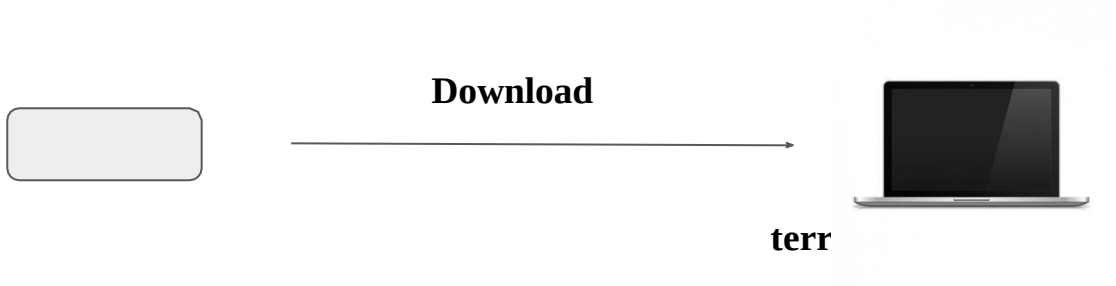
Installing Terraform

Terraform in detail

Overview of Installation Process

Terraform installation is very simple.

You have a single binary file, download and use it.



Supported Platforms

Terraform works on multiple platforms, these includes:

- **Windows**
- **macOS**
- **Linux**
- **FreeBSD**
- **OpenBSD**
- **Solaris**

Terraform Installation - Mac & Linux

There are two primary steps required to install terraform in Mac and Linux

1) Download the Terraform Binary File.

2) Move it in the right path.

Choosing IDE For Terraform

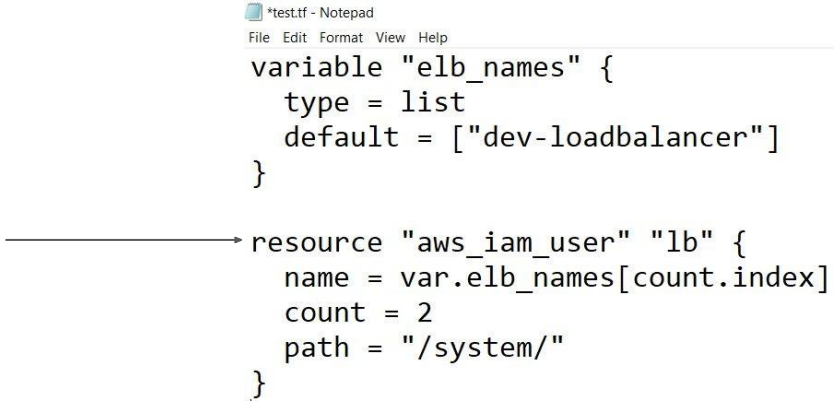
Terraform in detail

Terraform Code in NotePad!

You can write Terraform code in Notepad and it will not have any impact.

Downsides:

- Slower Development
- Limited Features



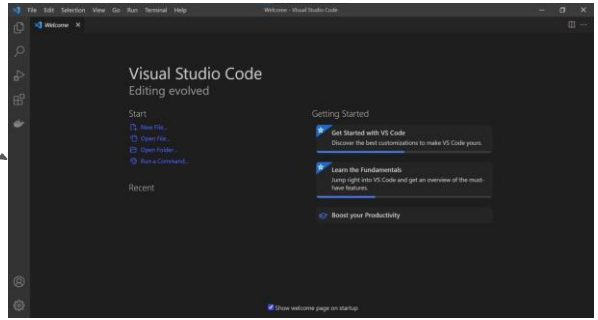
Need of a Better Software

There is a need of a better application that allows us to develop code faster.

What are the Options!

There are many popular source code editors available in the market.

Source Code Editors

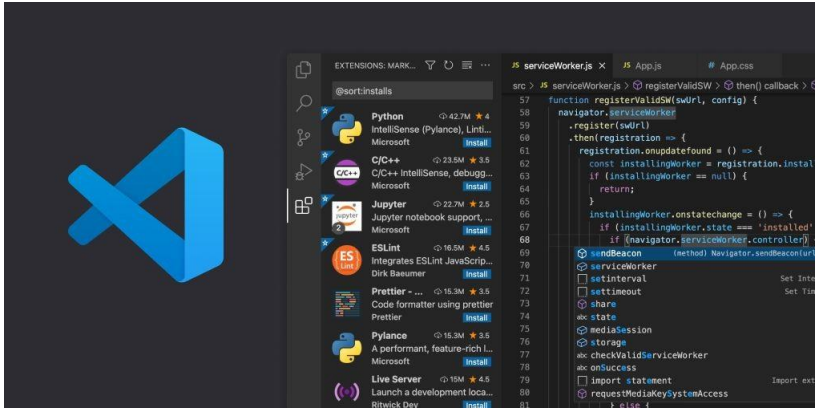


Editor for This Course

We are going to make use of Visual Studio Code as primary editor in this course.

Advantages:

1. Supports Windows, Mac, Linux
2. Supports Wide variety of programming languages.
3. Many Extensions.



Visual Studio Code Extensions

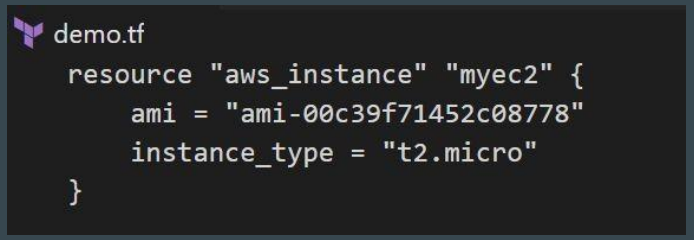
Understanding the Basics

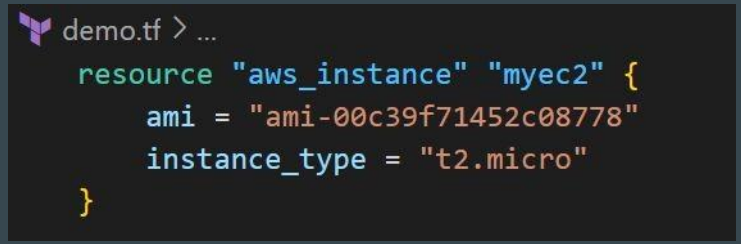
Extensions are add-ons that allow you to customize and enhance your experience in Visual Studio by adding new features or integrating existing tools

They offer wide range of functionality related to colors, auto-complete, report spelling errors etc.

Terraform Extension

HashiCorp also provides extension for Terraform for Visual Studio Code.





Setting up the Lab

Let’s start Rolling !

Let’s Start

i) Create a new AWS Account.

ii) Begin the course

Registering an AWS Account

Authentication and Authorization

Understanding the Basics

Before we start working on managing environments through Terraform, the first important step is related to Authentication and Authorization.

Basics of Authentication and Authorization

Authentication is the process of verifying who a user is.

Authorization is the process of verifying what they have access to

Example:

Alice is a user in AWS with no access to any service.

Learning for Todays' Video

Terraform needs access credentials with relevant permissions to create and manage the environments.

username	password
Bob	pwd928#

Access Credentials

Depending on the provider, the type of access credentials would change.

Provider	Access Credentials
AWS	Access Keys and Secret Keys
GitHub	Tokens
Kubernetes	Kubeconfig file, Credentials Config
Digital Ocean	Tokens

First Virtual Machine Through Terraform

Revising the Basics of EC2

EC2 stands for Elastic Compute Cloud.

In-short, it's a name for a virtual server that you launch in AWS.

VM **EC2 Instance**

Available Regions

Cloud providers offers multiple regions in which we can create our resource.

You need to decide the region in which Terraform would create the resource.

Virtual Machine Configuration

A Virtual Machine would have it’s own set of configurations.

- **CPU**
- **Memory**
- **Storage**
- **Operating System**

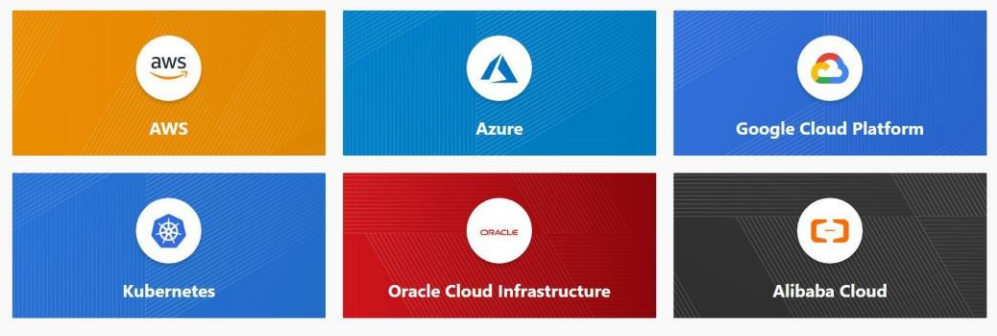
While creating VM through Terraform, you will need to define these.

Providers and Resources

Basics of Providers

Terraform supports multiple providers.

Depending on what type of infrastructure we want to launch, we have to useappropriate providers accordingly.



Learning 1 - Provider Plugins

A provider is a plugin that lets Terraform manage an external API.

When we run terraform init, plugins required for the provider are automatically downloaded and saved locally to a .terraform directory.

```
C:\Users\zealv\Desktop\kplabs-terraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v4.60.0...
- Installed hashicorp/aws v4.60.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

Learning 2 - Resource

Resource block describes one or more infrastructure objects

Example:

- **resource aws_instance**
- **resource aws_alb**
- **resource iam_user**
- **resource digitalocean_droplet**

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

Learning 3 - Resource Blocks

A resource block declares a resource of a given type ("aws_instance") with a given local name ("myec2").

Resource type and Name together serve as an identifier for a given resourceand so must be unique.

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

EC2 Instance Number 1

```
resource "aws_instance" "web" {
  ami           = ami-123
  instance_type = "t2.micro"
}
```

EC2 Instance Number 2

Point to Note

You can only use the resource that are supported by a specific provider.

In the below example, provider of Azure is used with resource of aws_instance

```
provider "azurerm" {}

resource "aws_instance" "web" {
  ami           = ami-123
  instance_type = "t2.micro"

}
```

Important Question

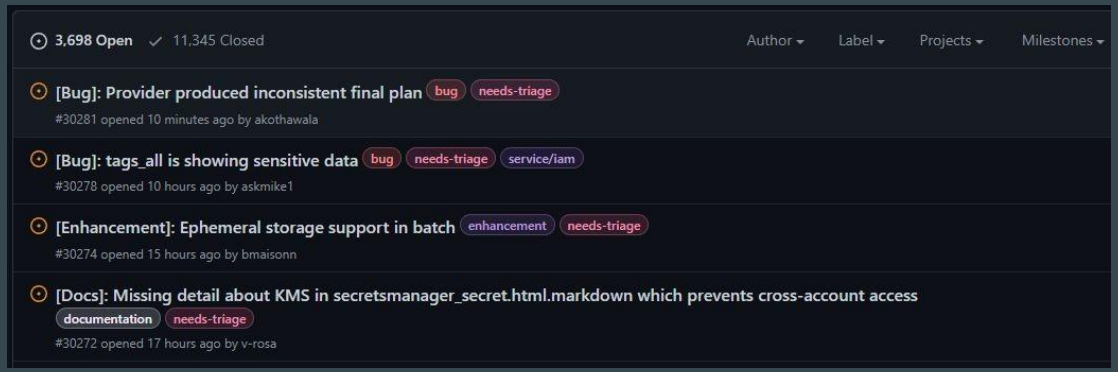
The core concepts, standard syntax remains similar across all providers.

If you learn the basics, you should be able to work with all providers easily.

Issues and Bugs with Providers

A provider that is maintained by HashiCorp does not mean it has no bugs.

It can happen that there are inconsistencies from your output and things mentioned in documentation. You can raise issue at Provider page.



Provider Tiers

Provider Maintainers

There are 3 primary type of provider tiers in Terraform.

Provider Tiers	Description
Official	Owned and Maintained by HashiCorp.
Partner	Owned and Maintained by Technology Company that maintains direct partnership with HashiCorp.
Community	Owned and Maintained by Individual Contributors.

Provider Namespace

Namespaces are used to help users identify the organization or publisher responsible for the integration

Tier	Description
Official	hashicorp
Partner	Third-party organization e.g. mongodb/mongodbatlas
Community	Maintainer's individual or organization account, e.g. DeviaVir/gsuite

Important Learning

Terraform requires explicit source information for any providers that are not HashiCorp-maintained, using a new syntax in the required_providers nested block inside the terraform configuration block

```
provider "aws" {
  region     = "us-west-2"
  access_key = "PUT-YOUR-ACCESS-KEY-HERE"
  secret_key = "PUT-YOUR-SECRET-KEY-HERE"
}
```

HashiCorp Maintained

```
terraform {
  required_providers {
    digitalocean = {
      source = "digitalocean/digitalocean"
    }
  }
}

provider "digitalocean" {
  token = "PUT-YOUR-TOKEN-HERE"
}
```

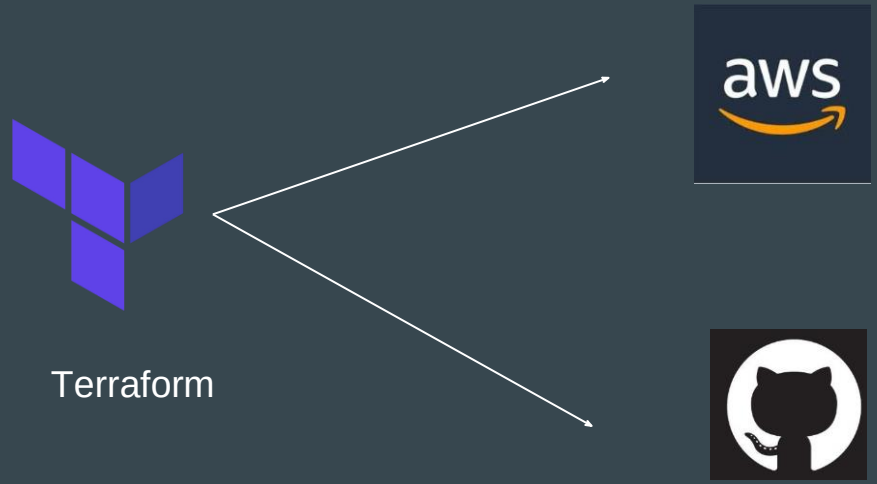
Non-HashiCorp Maintained

Terraform Destroy

Learning to Destroy Resources

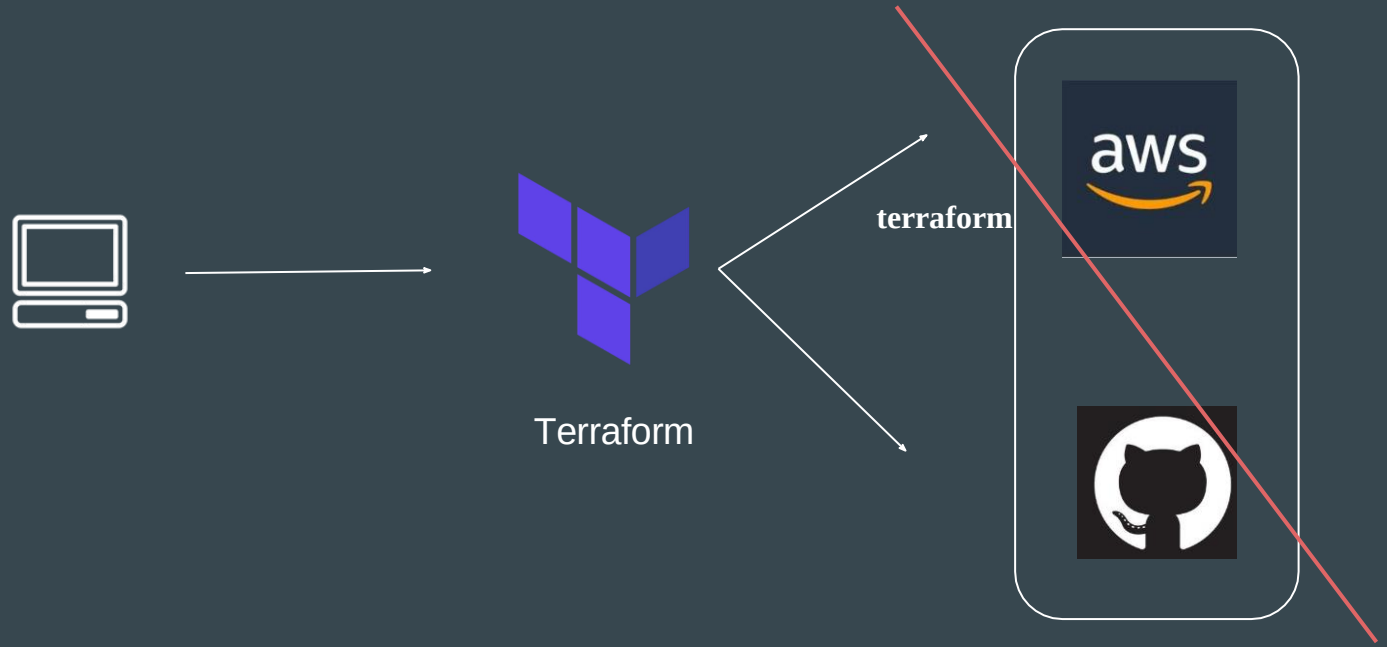
If you keep the infrastructure running, you will get charged for it.

Hence it is important for us to also know on how we can delete the infrastructure resources created via terraform.



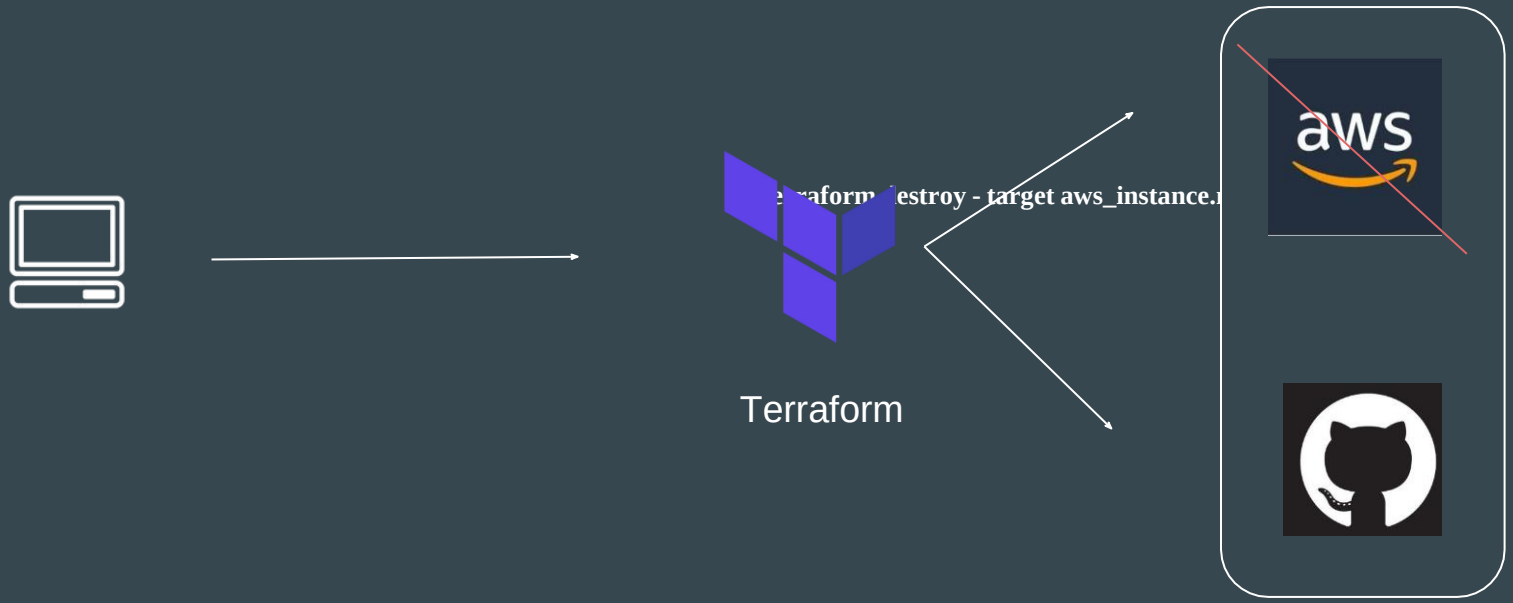
Approach 1 - Destroy ALL

terraform destroy allows us to destroy all the resource that are created within thefolder.



Approach 2 - Destroy Some

terraform destroy with -target flag allows us to destroy specific resource.



Terraform Destroy with Target

The -target option can be used to focus Terraform's attention on only a subset of resources.

Combination of : Resource Type + Local Resource Name

Resource Type	Local Resource Name
aws_instance	myec2
github_repository	example

```
resource "aws_instance" "myec2" {
    ami = "ami-00c39f71452c08778"
    instance_type = "t2.micro"
}
```

```
resource "github_repository" "example" {
  name        = "example"
  description = "My awesome codebase"

  visibility = "public"

}
```

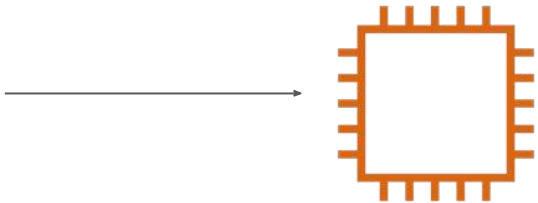
Desired & Current State

Terraform in detail

Desired State

Terraform's primary function is to create, modify, and destroy infrastructure resources tomatch the desired state described in a Terraform configuration

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```



EC2 - t2.micro

Current State

Current state is the actual state of a resource that is currently deployed.

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

t2.medium

Important Pointer

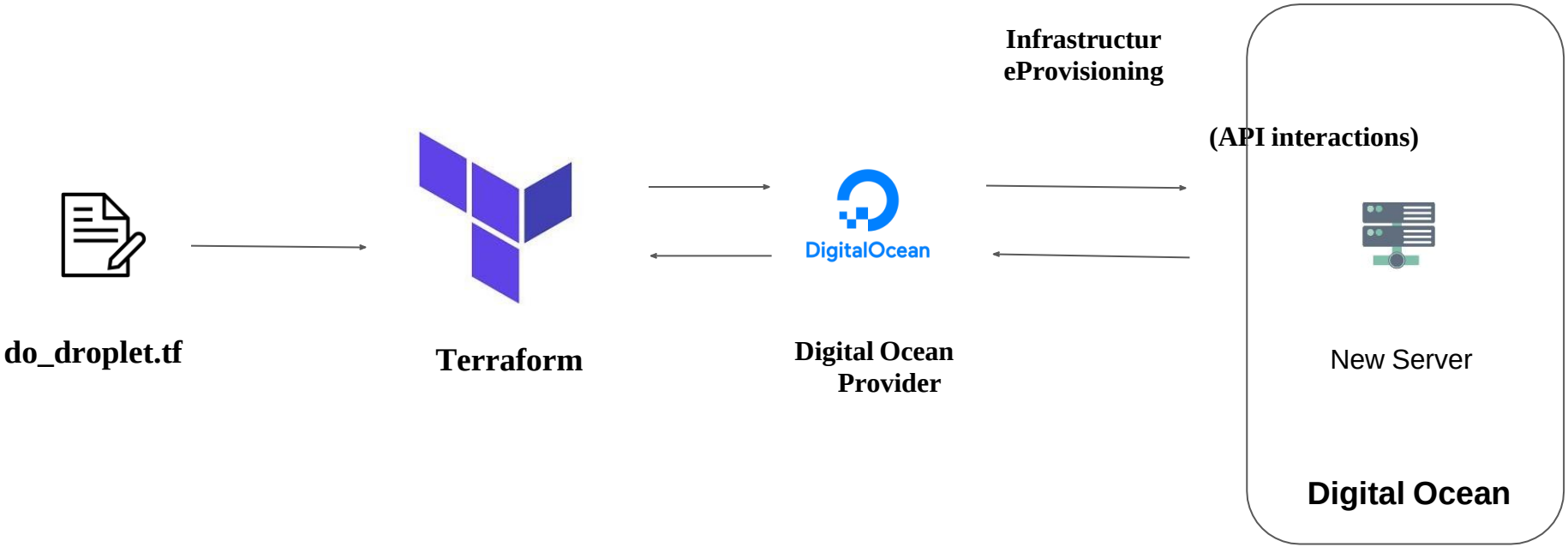
- **Terraform tries to ensure that the deployed infrastructure is based on the desired state.**

  - **If there is a difference between the two, terraform plan presents a description of thechanges necessary to achieve the desired state.**

Provider Versioning

Terraform in detail

Provider Architecture



Overview of Provider Versioning

Provider plugins are released separately from Terraform itself.

They have different set of version numbers.

Version 1

Version 2

Explicitly Setting Provider Version

During terraform init, if version argument is not specified, the most recent provider will be downloaded during initialization.

For production use, you should constrain the acceptable provider versions via configuration, toensure that new versions with breaking changes will not be automatically installed.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}
```

Arguments for Specifying provider

There are multiple ways for specifying the version of a provider.

Version Number Arguments	Description
>=1.0	Greater than equal to the version
<=1.0	Less than equal to the version
~>2.0	Any version in the 2.X range.
>=2.10,<=2.30	Any version between 2.10 and 2.30

Dependency Lock File

Terraform dependency lock file allows us to lock to a specific version of the provider.

If a particular provider already has a selection recorded in the lock file, Terraform will alwaysre-select that version for installation, even if a newer version has become available.

You can override that behavior by adding the -upgrade option when you run terraform init,

```
provider "registry.terraform.io/hashicorp/aws" {
  version     = "2.70.0"
  constraints = ">= 2.31.0, <= 2.70.0"
  hashes = [
    "h1:fx8tbGVwK1YIDI6UdHLnorC9PA1ZPSWEeW3V3aDCdWY=",
    "zh:01a5f351146434b418f9ff8d8cc956ddc801110f1cc8b139e01be2ff8c544605",
    "zh:1ec08abbaf09e3e0547511d48f77a1e2c89face2d55886b23f643011c76cb247",
    "zh:606d134fef7c1357c9d155aadbee6826bc22bc0115b6291d483bc1444291c3e1",
    "zh:67e31a71a5ecbbc96a1a6708c9cc300bbfe921c322320cdbb95b9002026387e1",
```

Terraform Refresh

Understanding the Challenge

Terraform can create an infrastructure based on configuration you specified.It

can happen that the infrastructure gets modified manually.

```
resource "aws_instance" "web" {
  ami           = ami-123
  instance_type = "t2.micro"
}
```

t2.micro

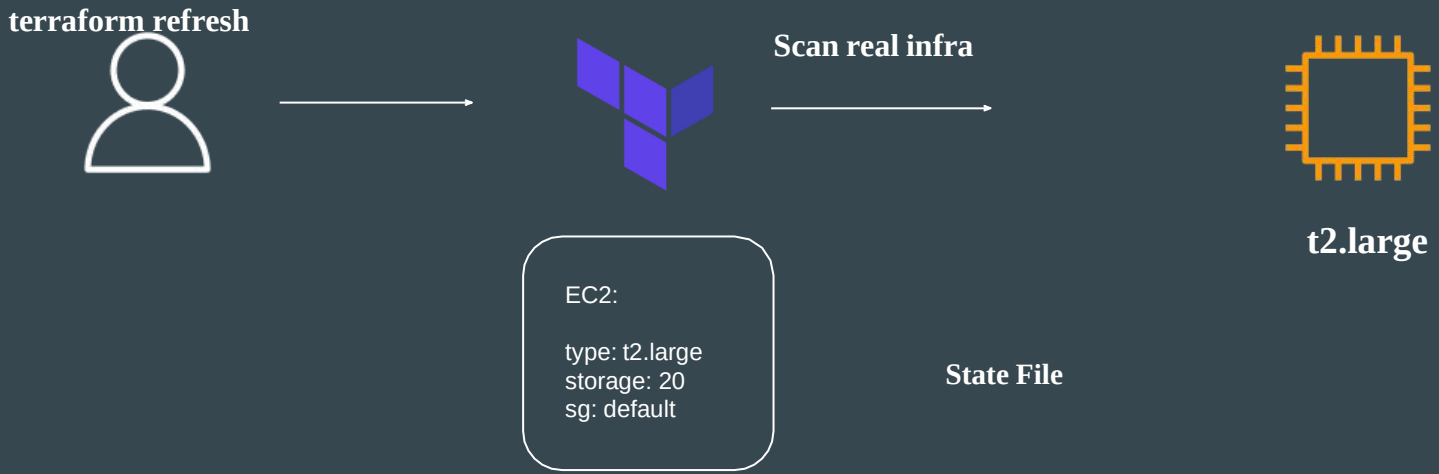
EC2:

type: t2.micro
storage: 20
sg: default

State File

Understanding the Challenge

The terraform refresh command will check the latest state of your infrastructure and update the state file accordingly.



Points to Note

You shouldn't typically need to use this command, because Terraform automatically performs the same refreshing actions as a part of creating a planin both the terraform plan and terraform apply commands.

Understanding the Usage

The terraform refresh command is deprecated in newer version of terraform.

The -refresh-only option for terraform plan and terraform apply was introduced in Terraform v0.15.4.

Lecture Format - Terraform Course

Terraform in detail

Overview of the Format

We tend to use a different folder for each practical that we do in the course. This allows us to be more systematic and allows easier revisit in-case required.

Lecture Name	Folder Names
Create First EC2 Instance	folder1
Tainting resource	folder2

Conditional Expression	folder3

Find the appropriate code from GitHub

Code in GitHub is arranged according to sections that are matched to the domains in the course.Every section in GitHub has easy Readme file for quick navigation.

Video-Document Mapper

Sr No	Document Link
1	Understanding Attributes and Output Values in Terraform
2	Referencing Cross-Account Resource Attributes
3	Terraform Variables
4	Approaches for Variable Assignment
5	Data Types for Variables

Destroy Resource After Practical

We know how to destroy resources by

now**terraform destroy**

After you have completed your practical, make sure you destroy the resource before moving tothe next practical.

This is easier if you are maintaining separate folder for each practical.

Attributes & Output Values

Terraform in detail

Understanding Attributes

Terraform has capability to output the attribute of a resource with the output values.

Example:

ec2_public_ip = 35.161.21.197

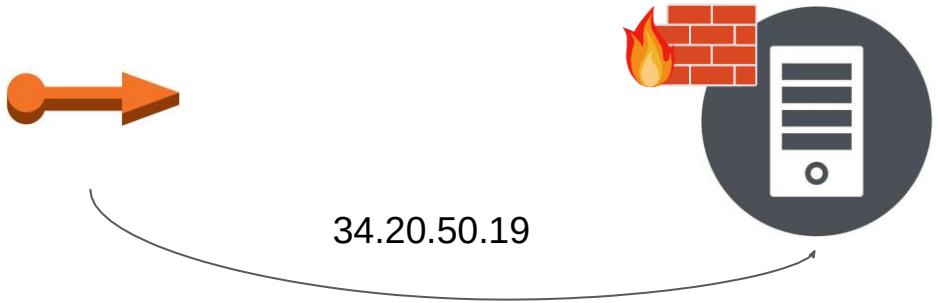
bucket_identifier = terraform-test-kplabs.s3.amazonaws.com

Attributes are important

An outputed attributes can not only be used for the user reference but it can also act as a input toother resources being created via terraform

Let’s understand this with an example:

After EIP gets created, it’s IP address should automatically get whitelisted in the security group.



Referencing Cross-Resource Attributes

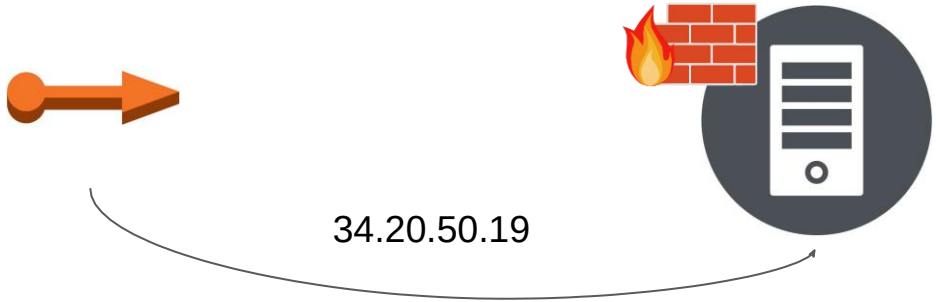
Terraform in detail

Attributes are important

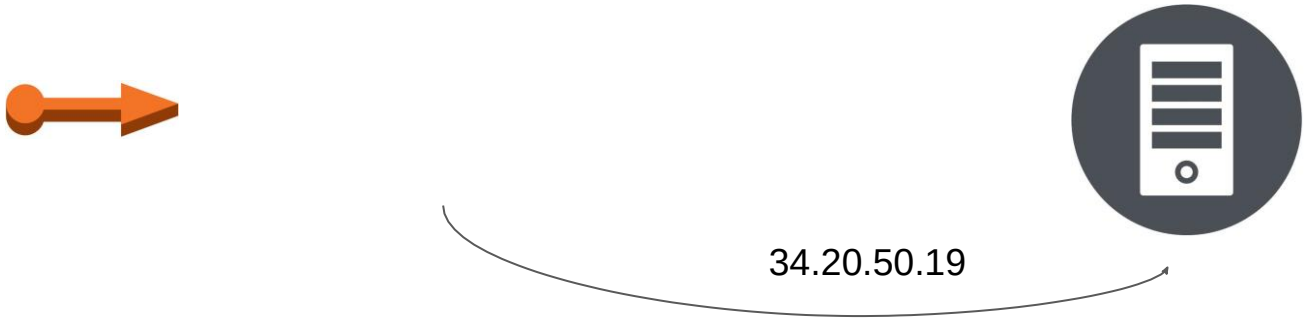
An outputed attributes can not only be used for the user reference but it can also act as a input toother resources being created via terraform

Let’s understand this with an example:

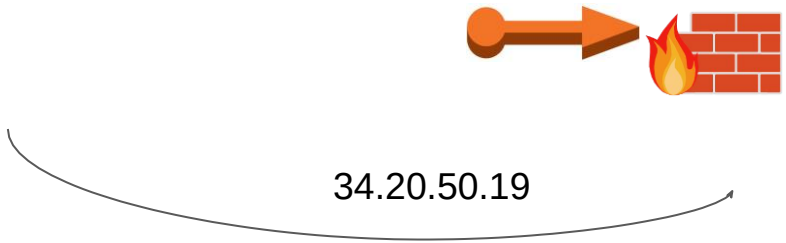
After EIP gets created, it’s IP address should automatically get whitelisted in the security group.



Example 1: EIP and EC2 Instance



Example 2: EIP and Security Group

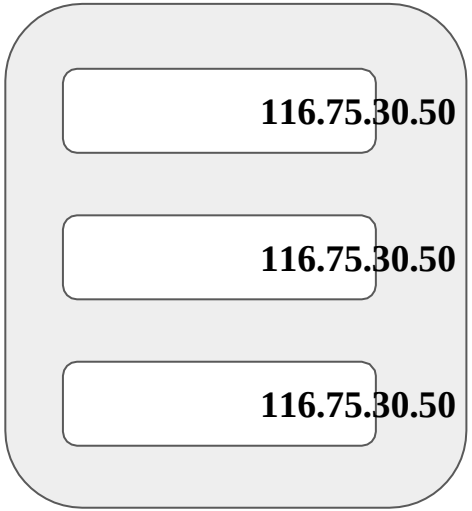


Terraform Variables

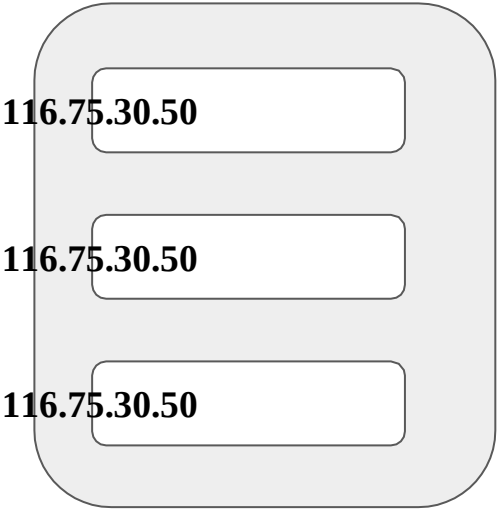
Terraform in detail

Static = Work

Repeated static values can create more work in the future.



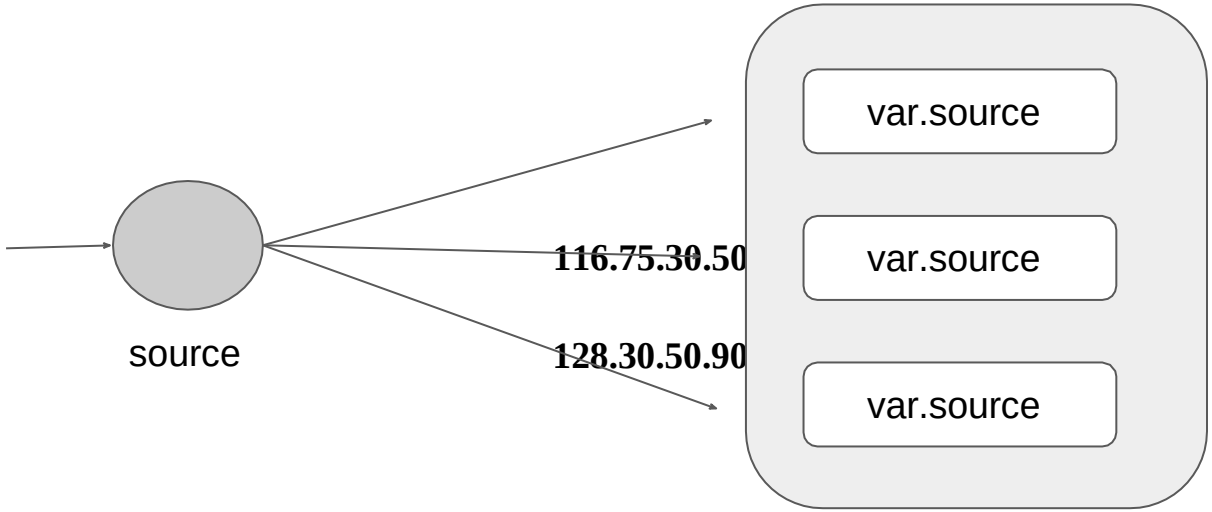
Project A



Project B

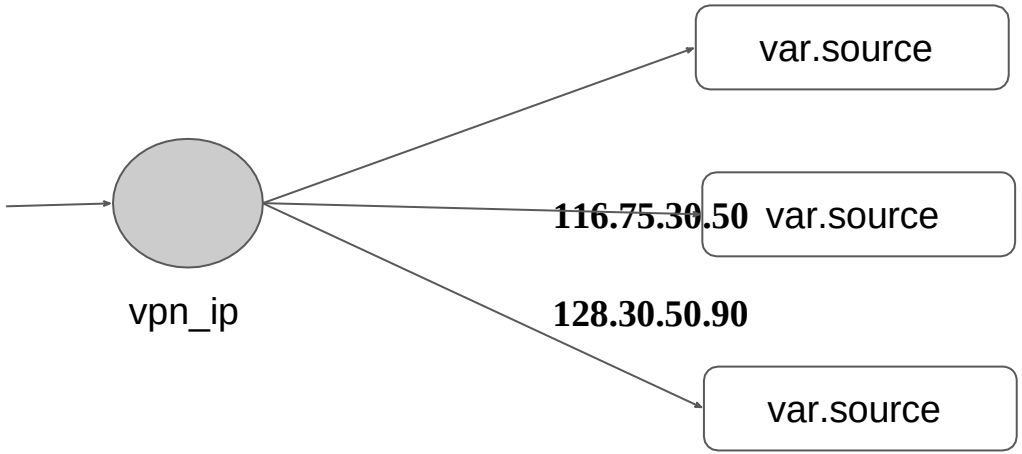
Variables are good

We can have a central source from which we can import the values from.



Variables are good

We can have a central source from which we can import the values from.



Approaches to Variable Assignment

Terraform in detail

Multiple Approaches to Variable Assignment

Variables in Terraform can be assigned values in multiple ways.

Some of these include:

- Environment variables
- Command Line Flags
- From a File
- Variable Defaults

Data Types for Variables

Terraform in detail

Overview of Type Constraints

The type argument in a variable block allows you to restrict the type of value that will be acceptedas the value for a variable

```
variable "image_id" {
type = string
}
```

If no type constraint is set then a value of any type is accepted.

Example Use-Case

Every employee in Medium Corp is assigned a Identification Number.

Any resource that employee creates should be created with the name of the identification numberonly.

variables.tf	terraform.tfvars
variable “instance_name” {}	instance_name=”john-123”

Example Use-Case

Every employee in Medium Corp is assigned a Identification Number.

Any EC2 instance that employee creates should be created using the identification number only.

variables.tf	terraform.tfvars

variable “instance_name” {
type=number
}
	instance_name=”john-123”

Overview of Data Types

Type Keywords	Description
string	Sequence of Unicode characters representing some text, like "hello".
list	Sequential list of values identified by their position. Starts with 0 [“mumbai” ,”singapore”, ”usa”]
map	a group of values identified by named labels, like {name = "Mabel", age = 52}.
number	Example: 200

Count Parameter

Terraform in detail

Overview of Count Parameter

The count parameter on resources can simplify configurations and let you scale resources bysimply incrementing a number.

Let’s assume, you need to create two EC2 instances. One of the common approach is to definetwo separate resource blocks for aws_instance.

```
resource "aws_instance" "instance-1" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

```
resource "aws_instance" "instance-2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

Overview of Count Parameter

With count parameter, we can simply specify the count value and the resource can be scaledaccordingly.

```
resource "aws_instance" "instance-1" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
   count = 5
}
```

Count Index

In resource blocks where count is set, an additional count object is available in expressions, so youcan modify the configuration of each instance.

This object has one attribute:

count.index — The distinct index number (starting with 0) corresponding to this instance.

Understanding Challenge with Count

With the below code, terraform will create 5 IAM users. But the problem is that all will have thesame name.

```
resource "aws_iam_user" "lb" {
  name = "loadbalancer"
  count = 5
  path = "/system/"
}
```

Understanding Challenge with Count

count.index allows us to fetch the index of each iteration in the loop.

```
resource "aws_iam_user" "lb" {
  name = "loadbalancer.${count.index}"
  count = 5
  path = "/system/"
}
```

Understanding Challenge with Default Count Index

Having a username like loadbalancer0, loadbalancer1 might not always be suitable.Better names like dev-loadbalancer, stage-loadbalancer, prod-loadbalancer is better. count.index can help in such scenario as well.

```
variable "elb_names" {
  type        = list
  default     = ["dev-loadbalancer", "stage-loadbalancer", "prod-loadbalancer"]
}
```

Conditional Expression

Terraform in detail

Overview of Conditional Expression

A conditional expression uses the value of a bool expression to select one of two values.Syntax of Conditional expression:

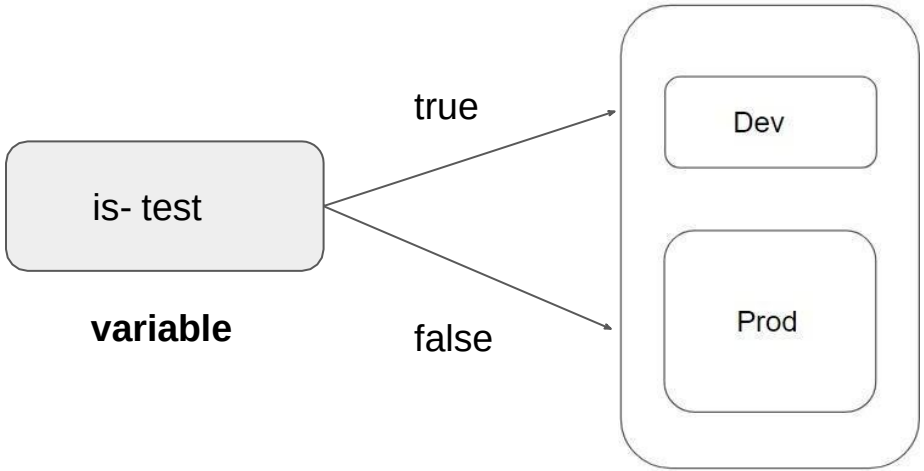
```
condition ? true_val : false_val
```

If condition is true then the result is true_val. If condition is false then the result is false_val.

Example of Conditional Expression

Let's assume that there are two resource blocks as part of terraform configuration.

Depending on the variable value, one of the resource blocks will run.

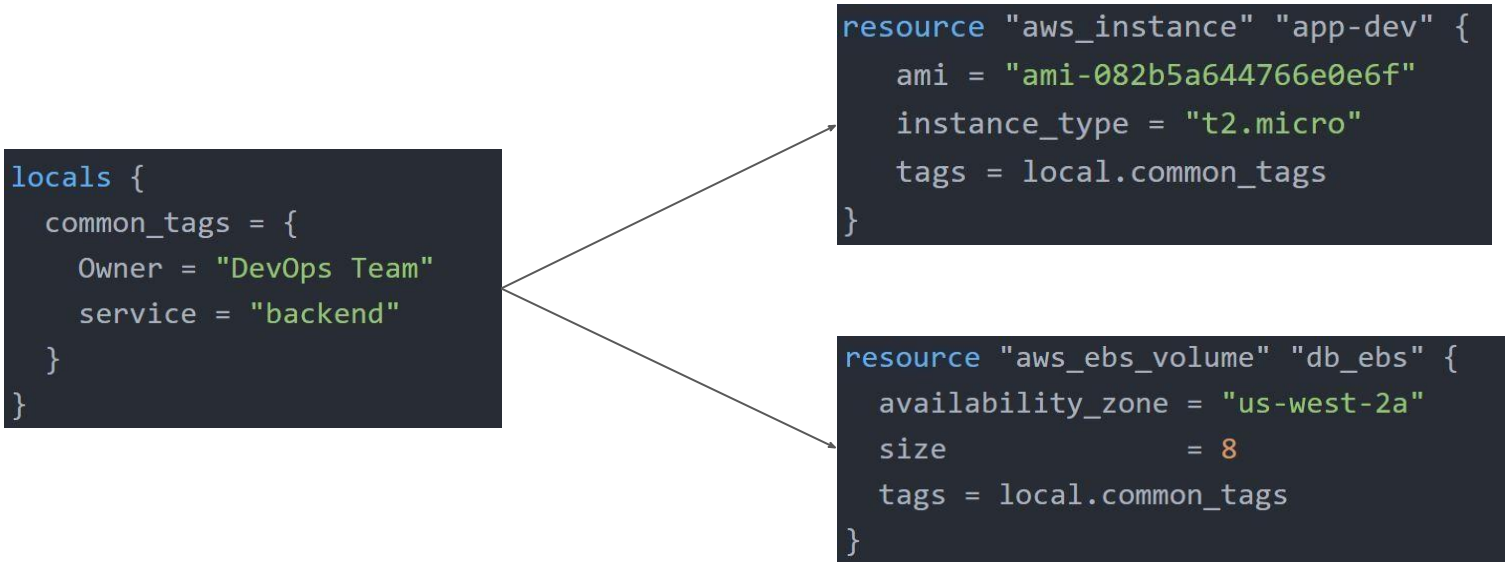


Local Values

Terraform in detail

Overview of Local Values

A local value assigns a name to an expression, allowing it to be used multiple times within amodule without repeating it.



Local Values Support for Expression

Local Values can be used for multiple different use-cases like having a conditional expression.

```
locals {
  name_prefix = "${var.name != "" ? var.name : var.default}"
}
```

Important Pointers for Local Values

Local values can be helpful to avoid repeating the same values or expressions multiple times in aconfiguration.

If overused they can also make a configuration hard to read by future maintainers by hiding theactual values used

Use local values only in moderation, in situations where a single value or result is used in manyplaces and that value is likely to be changed in future.

Terraform Functions

Terraform in detail

Overview of Terraform Functions

The Terraform language includes a number of built-in functions that you can use to transformand combine values.

The general syntax for function calls is a function name followed by comma-separated argumentsin parentheses:

function (argument1, argument2)

Example:

```
> max(5, 12, 9)
12
```

List of Available Functions

The Terraform language does not support user-defined functions, and so only the functions builtin to the language are available for use

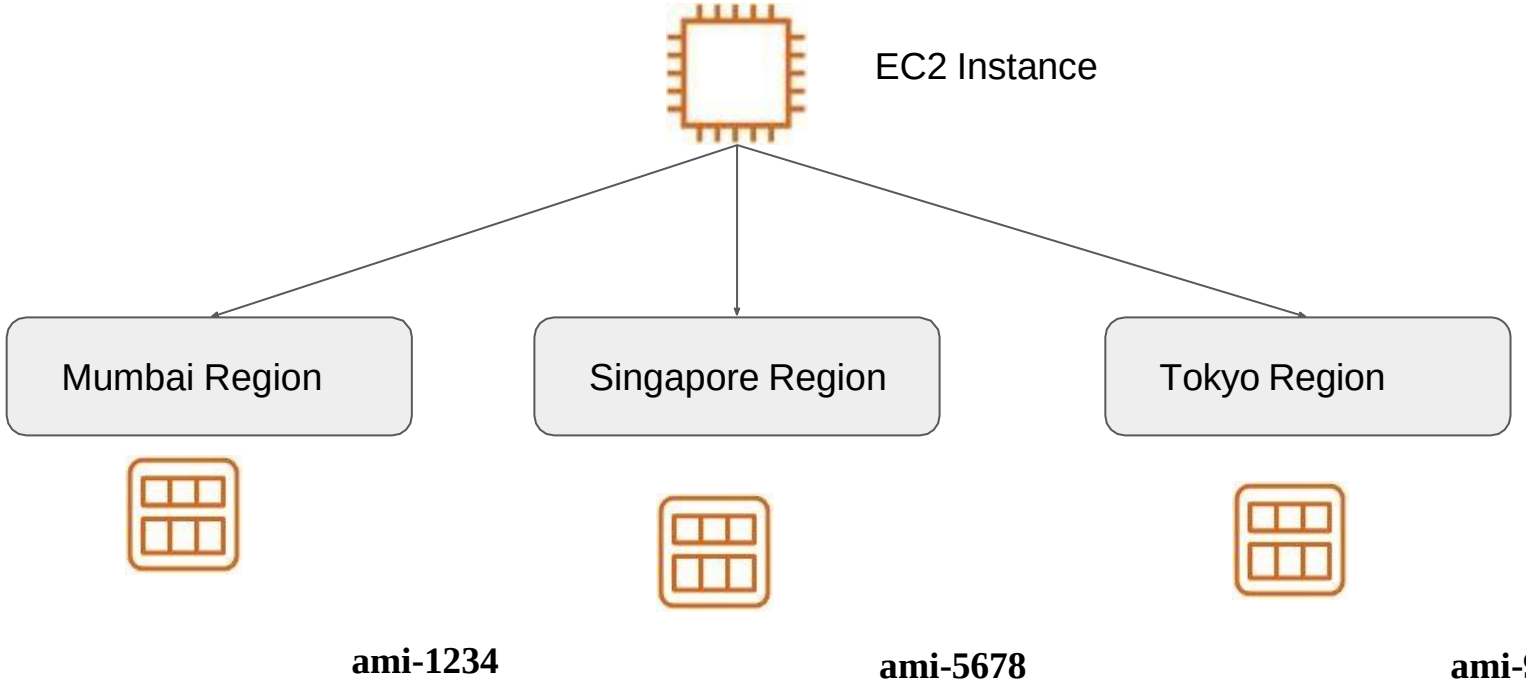
- **Numeric**
- **String**
- **Collection**
- **Encoding**
- **Filesystem**
- **Date and Time**
- **Hash and Crypto**
- **IP Network**
- **Type Conversion**

Data Sources

Terraform in detail

Overview of Data Sources

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.



Data Source Code

- Defined under the data block.
- Reads from a specific data source (aws_ami) and exports results under “app_ami”

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


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

```
resource "aws_instance" "instance-1" {
    ami = data.aws_ami.app_ami.id
   instance_type = "t2.micro"
}
```

Debugging Terraform

Terraform in detail

Overview of Debugging Terraform

Terraform has detailed logs which can be enabled by setting the TF_LOG environment variableto any value.

You can set TF_LOG to one of the log levels TRACE, DEBUG, INFO, WARN or ERROR to change the verbosity of the logs

```
bash-4.2# terraform plan
2020/04/22 13:45:31 [INFO] Terraform version: 0.12.24
2020/04/22 13:45:31 [INFO] Go runtime version: go1.12.13
2020/04/22 13:45:31 [INFO] CLI args: []string{"/usr/bin/terraform", "plan"}
2020/04/22 13:45:31 [DEBUG] Attempting to open CLI config file: /root/.terraformrc
2020/04/22 13:45:31 [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2020/04/22 13:45:31 [DEBUG] checking for credentials in "/root/.terraform.d/plugins"
2020/04/22 13:45:31 [INFO] CLI command args: []string{"plan"}
2020/04/22 13:45:31 [TRACE] Meta.Backend: built configuration for "s3" backend with hash value 789489680
2020/04/22 13:45:31 [TRACE] Meta.Backend: backend has not previously been initialized in this working directory
2020/04/22 13:45:31 [DEBUG] New state was assigned lineage "a10f92bf-686d-e6cf-3e9d-755be5c8a6a3"
2020/04/22 13:45:31 [TRACE] Meta.Backend: moving from default local state only to "s3" backend
```

Important Pointers

TRACE is the most verbose and it is the default if TF_LOG is set to something other than a loglevel name.

To persist logged output you can set TF_LOG_PATH in order to force the log to always beappended to a specific file when logging is enabled.

Lecture Format - Terraform Course

Terraform in detail

Overview of the Format

We tend to use a different folder for each practical that we do in the course. This allows us to be more systematic and allows easier revisit in-case required.

Lecture Name	Folder Names
Create First EC2 Instance	folder1
Tainting resource	folder2
Conditional Expression	folder3

Find the appropriate code from GitHub

Code in GitHub is arranged according to sections that are matched to the domains in the course.Every section in GitHub has easy Readme file for quick navigation.

Video-Document Mapper

Sr No	Document Link
1	Understanding Attributes and Output Values in Terraform
2	Referencing Cross-Account Resource Attributes
3	Terraform Variables
4	Approaches for Variable Assignment
5	Data Types for Variables

Destroy Resource After Practical

We know how to destroy resources by

now**terraform destroy**

After you have completed your practical, make sure you destroy the resource before moving tothe next practical.

This is easier if you are maintaining separate folder for each practical.

Terraform Format

Terraform in detail

Importance of Readability

Anyone who is into programming knows the importance of formatting the code for readability.

The terraform fmt command is used to rewrite Terraform configuration files to take care of theoverall formatting.

```
provider "aws" {
      region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key    = "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"
  version    = ">=2.10,<=2.30"
}
```

```
provider "aws" {
      region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key    = "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"
  version    = ">=2.10,<=2.30"
}
```

```
provider "aws" {
  region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key = "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"
  version    = ">=2.10,<=2.30"
}
```

Terraform Validate

Terraform in detail

Overview of Terraform Validate

Terraform Validate primarily checks whether a configuration is syntactically valid.

It can check various aspects including unsupported arguments, undeclared variables and others.

```
resource "aws_instance" "myec2" {
  ami           = "ami-082b5a644766e0e6f"
  instance_type = "t2.micro"
  sky = "blue"
}
```

```
bash-4.2# terraform validate

Error: Unsupported argument

  on validate.tf line 10, in resource "aws_instance" "myec2":
  10:   sky = "blue"

An argument named "sky" is not expected here.
```

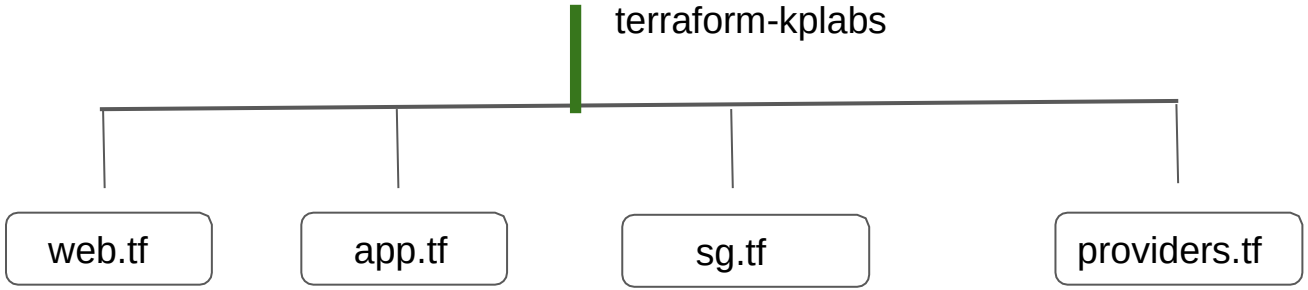
Load Order & Semantics

Terraform in detail

Understanding Semantics

Terraform generally loads all the configuration files within the directory specified inalphabetical order.

The files loaded must end in either .tf or .tf.json to specify the format that is in use.



Dynamic Block

Terraform In Depth

Understanding the Challenge

In many of the use-cases, there are repeatable nested blocks that needs to be defined.This can lead to a long code and it can be difficult to manage in a longer time.

```
ingress {
  from_port   = 9200
  to_port     = 9200
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

```
ingress {
  from_port   = 8300
  to_port     = 8300
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

Dynamic Blocks

Dynamic Block allows us to dynamically construct repeatable nested blocks which is supportedinside resource, data, provider, and provisioner blocks:

```
dynamic "ingress" {
  for_each = var.ingress_ports
  content {
    from_port   = ingress.value
    to_port     = ingress.value
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Iterators

The iterator argument (optional) sets the name of a temporary variable that represents thecurrent element of the complex value

If omitted, the name of the variable defaults to the label of the dynamic block ("ingress" in the example above).

```
dynamic "ingress" {
  for_each = var.ingress_ports
  content {
    from_port   = ingress.value
    to_port     = ingress.value
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```



```
dynamic "ingress" {
  for_each = var.ingress_ports
  iterator = port
  content {
    from_port   = port.value
    to_port     = port.value
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

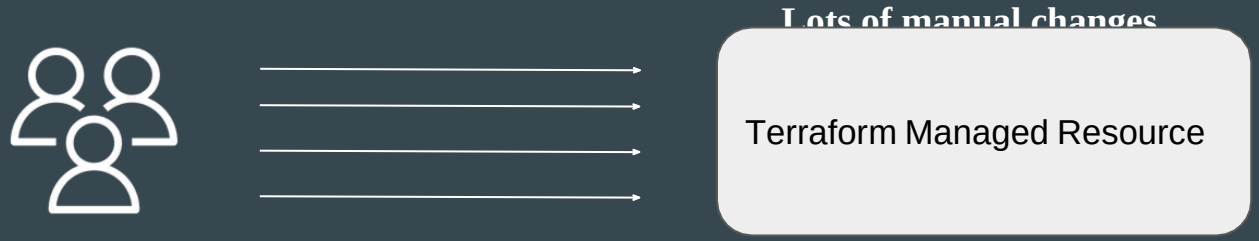
Terraform Taint

Understanding the Use-Case

You have created a new resource via Terraform.

Users have made a lot of manual changes (both infrastructure and inside the server)

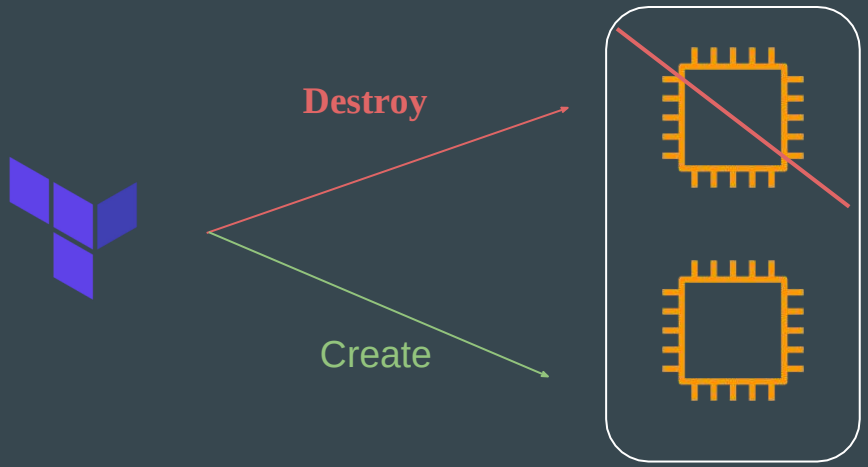
Two ways to deal with this: Import Changes to Terraform / Delete & Recreate the resource



Recreating the Resource

The -replace option with terraform apply to force Terraform to replace an object even though there are no configuration changes that would require it.

terraform apply -replace="aws_instance.web"



Points to Note

Similar kind of functionality was achieved using terraform taint command in older versions of Terraform.

For Terraform v0.15.2 and later, HashiCorp recommend using the -replace option with terraform apply

Splat Expression

Terraform Expressions

Overview of Spalat Expression

Splat Expression allows us to get a list of all the attributes.

```
resource "aws_iam_user" "lb" {
  name = "iamuser.${count.index}"
  count = 3
  path = "/system/"
}

output "arns" {
  value = aws_iam_user.lb[*].arn
}
```

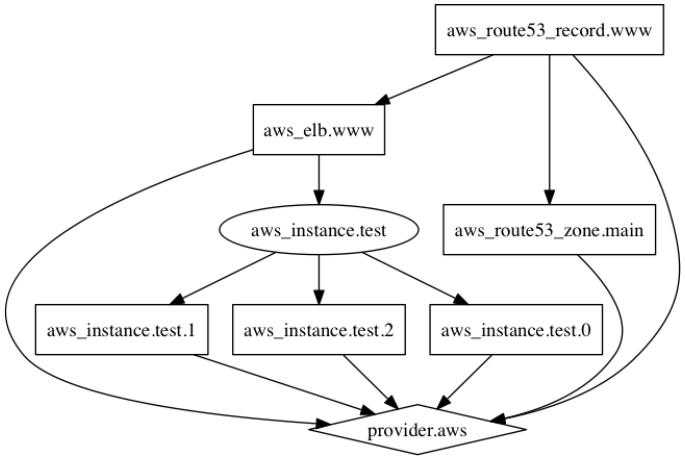
Terraform Graph

Terraform In Detail

Overview of Graph

The terraform graph command is used to generate a visual representation of either a configuration or execution plan

The output of terraform graph is in the DOT format, which can easily be converted toan image.



Saving Terraform Plan to a File

Terraform In Detail

Terraform Plan File

The generated terraform plan can be saved to a specific path.

This plan can then be used with terraform apply to be certain that only the changesshown in this plan are applied.

Example:

```
terraform plan -out=path
```

Terraform Output

Terraform in detail

Terraform Output

The terraform output command is used to extract the value of an output variable from the statefile.

```
C:\Users\Zeal Vora\Desktop\terraform\terraform output>terraform output iam_names
[
  "iamuser.0",
  "iamuser.1",
  "iamuser.2",
]
```

Terraform Settings

Terraform in detail

Overview of Terraform Settings

The special terraform configuration block type is used to configure some behaviors of Terraform itself, such as requiring a minimum Terraform version to apply your configuration.

Terraform settings are gathered together into terraform blocks:

```
terraform {
  # ...
}
```

Setting 1 - Terraform Version

The required_version setting accepts a version constraint string, which specifies which versionsof Terraform can be used with your configuration.

If the running version of Terraform doesn't match the constraints specified, Terraform willproduce an error and exit without taking any further actions.

```
terraform {
  required_version = "> 0.12.0"
}
```

Setting 2 - Provider Version

The required_providers block specifies all of the providers required by the current module,mapping each local provider name to a source address and a version constraint.

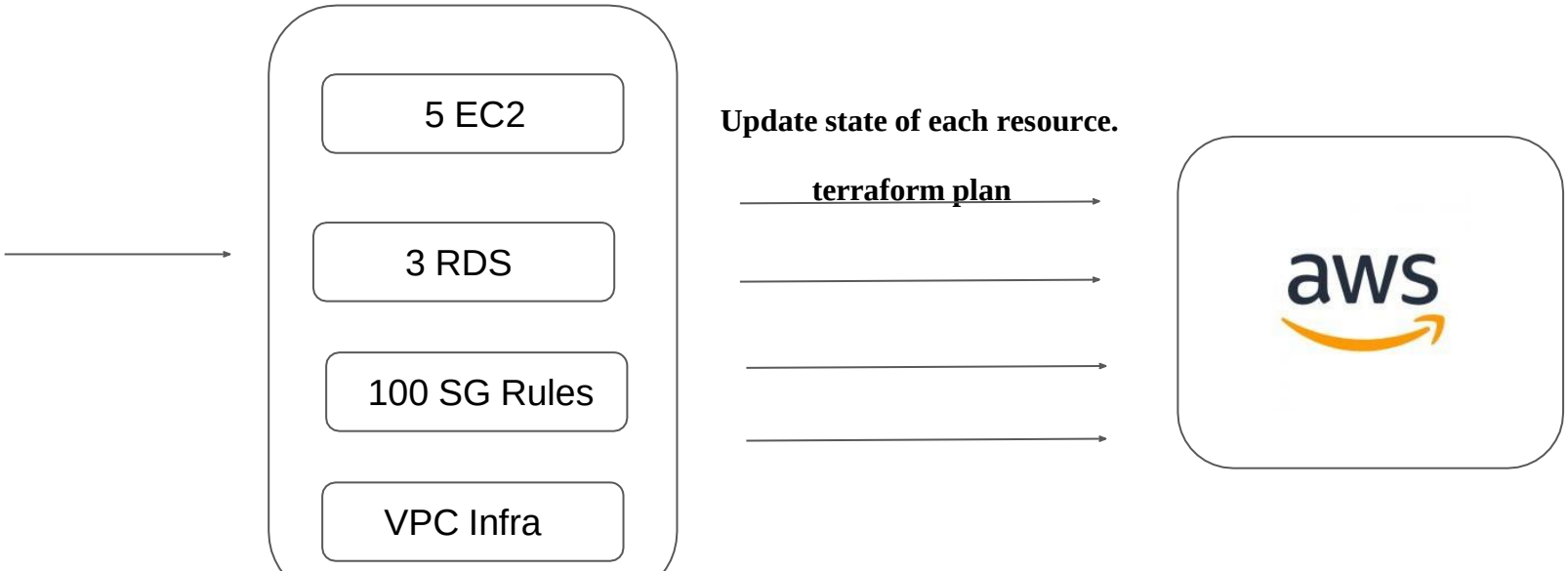
```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0"
    }
  }
}
```

Dealing with Larger Infrastructure

Terraform in detail

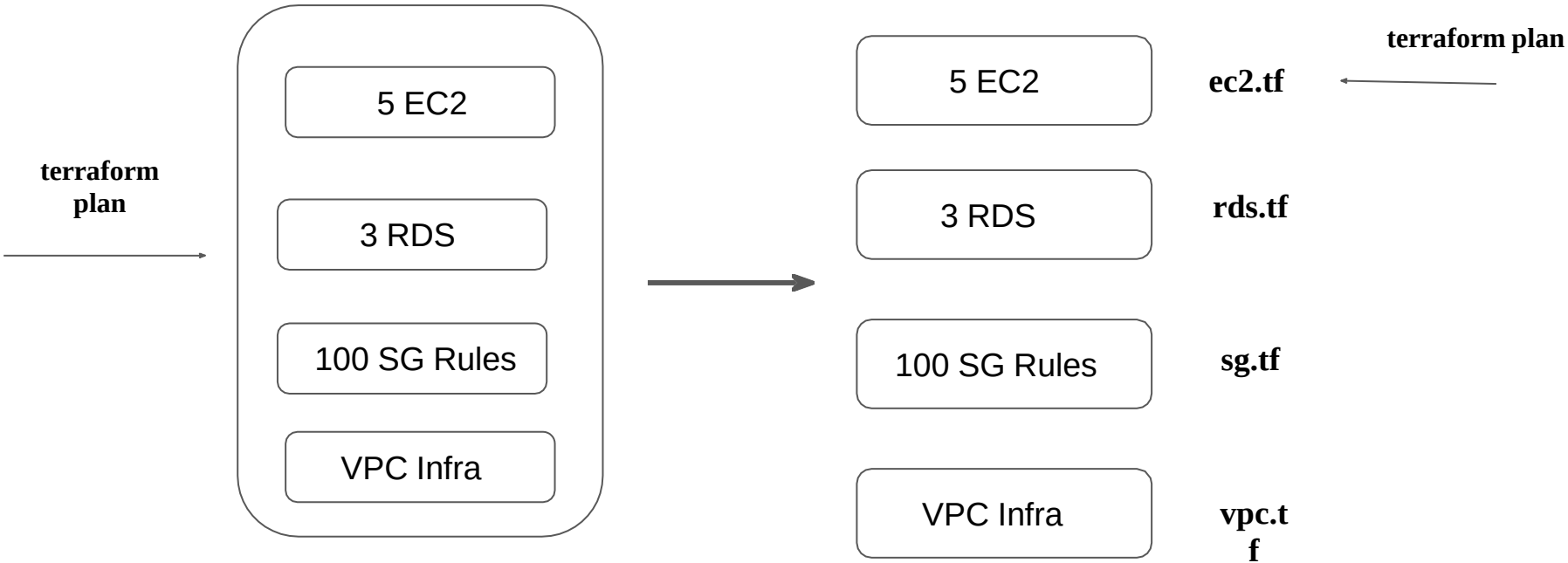
Challenges with Larger Infrastructure

When you have a larger infrastructure, you will face issue related to API limits for a provider.



Dealing With Larger Infrastructure

Switch to smaller configuration were each can be applied independently.

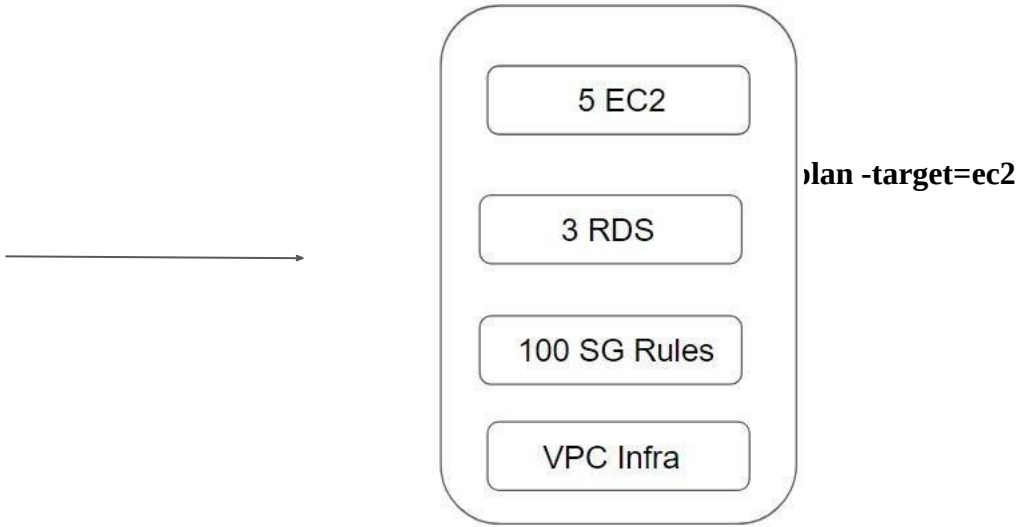


We can prevent terraform from querying the current state during operations like terraform plan.This can be achieved with the -refresh=false flag

Specify the Target

The -target=resource flag can be used to target a specific resource.

Generally used as a means to operate on isolated portions of very large configurations

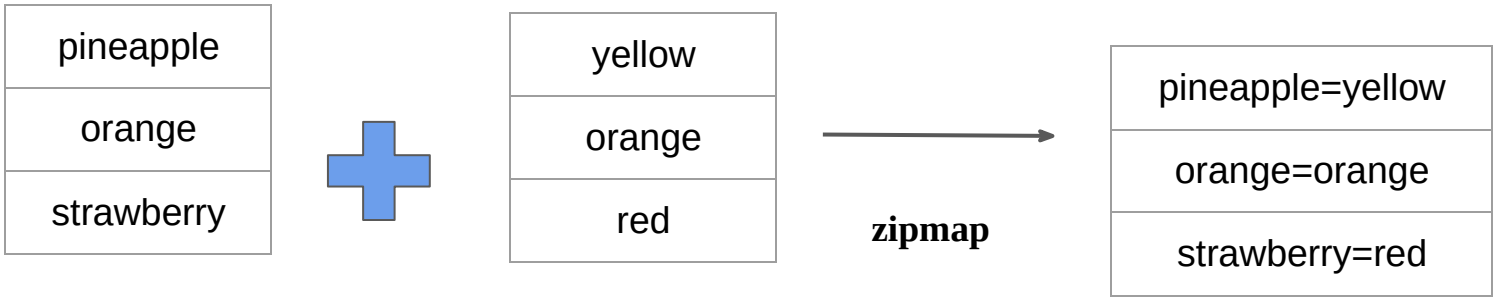


Zipmap

Terraform Function

Overview of Zipmap

The zipmap function constructs a map from a list of keys and a corresponding list of values.



Sample Output of Zipmap Function

```
←[J> zipmap(["pineapple","oranges","strawberry"], ["yellow","orange","red"])
{
  "oranges" = "orange"
  "pineapple" = "yellow"
  "strawberry" = "red"
}
```

Simple Use-Case

You are creating multiple IAM users.

You need output which contains direct mapping of IAM names and ARNs

```
zipmap = {
  "demo-user.0" = "arn:aws:iam::018721151861:user/system/demo-user.0"
  "demo-user.1" = "arn:aws:iam::018721151861:user/system/demo-user.1"
  "demo-user.2" = "arn:aws:iam::018721151861:user/system/demo-user.2"
}
```

Comments in Terraform Code

Commenting the Code!

Overview of Comments

A comment is a text note added to source code to provide explanatory information,usually about the function of the code

```
'''In this program, we check if the number is positive or
negative or zero and
display an appropriate message'''

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Comments in Terraform

The Terraform language supports three different syntaxes for comments:

Type	Description
#	begins a single-line comment, ending at the end of the line.
//	also begins a single-line comment, as an alternative to #.
/* and */	are start and end delimiters for a comment that might span over multiple lines.

Challenges with Count

Meta-Argument

Revising the Basics

Resource are identified by the index value from the list.

```
variable "iam_names" {
  type = list
  default = ["user-01","user-02","user-03"]
}

resource "aws_iam_user" "iam" {
  name = var.iam_names[count.index]
  count = 3
  path = "/system/"
}
```

Resource Address	Infrastructure
aws_iam_user.iam[0]	user-01
aws_iam_user.iam[1]	user-02
aws_iam_user.iam[2]	user-03

Challenge - 1

If the order of elements of index is changed, this can impact all of the other resources.

```
variable "iam_names" {
  type = list
  default = ["user-0","user-01","user-02","user-03"]
}

resource "aws_iam_user" "iam" {
  name = var.iam_names[count.index]
  count = 4
  path = "/system/"
}
```

Resource Address	Infrastructure
aws_iam_user.iam.[0]	user-01
aws_iam_user.iam.[1]	user-02
aws_iam_user.iam.[2]	user-03

Important Note

If your resources are almost identical, count is appropriate.

If distinctive values are needed in the arguments, usage of **for_each** is recommended.

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

Data Type - SET

Let’s Revise Programming

Basics of List

- Lists are used to store multiple items in a single variable.
- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

```
variable "iam_names" {
  type = list
  default = ["user-01","user-02","user-03"]
}
```

Understanding SET

- SET is used to store multiple items in a single variable.
- SET items are unordered and no duplicates allowed.

```
demoset = {"apple", "banana", "mango"}
```

```
demoset = {"apple", "banana", "mango", "apple"}
```

toset Function

toset function will convert the list of values to SET

```
> toset(["a", "b", "c","a"])
toset([
  "a",
  "b",
  "c",
])
```

for_each

Meta-Argument

Basics of For Each

for_each makes use of map/set as an index value of the created resource.

```
resource "aws_iam_user" "iam" {
  for_each = toset( ["user-01","user-02", "user-03"] )
  name     = each.key
}
```

Resource Address	Infrastructure
aws_iam_user.iam[user-01]	user-01
aws_iam_user.iam[user-02]	user-02
aws_iam_user.iam[user-03]	user-03

Replication Count Challenge

If a new element is added, it will not affect the other resources.

```
resource "aws_iam_user" "iam" {
  for_each = toset( ["user-0","user-01","user-02", "user-03"] )
  name     = each.key
}
```


Resource Address	Infrastructure
aws_iam_user.iam[user-01]	user-01
aws_iam_user.iam[user-02]	user-02
aws_iam_user.iam[user-03]	user-03
aws_iam_user.iam[user-0]	user-0

The each object

In blocks where for_each is set, an additional each object is available.

This object has two attributes:

Each object	Description
each.key	The map key (or set member) corresponding to this instance.
each.value	The map value corresponding to this instance

Provisioners

Interesting Part is here

Provisioners are interesting

Till now we have been working only on creation and destruction of infrastructure scenarios. **Let's take an example:**

We created a web-server EC2 instance with Terraform.

Problem: It is only an EC2 instance, it does not have any software installed.

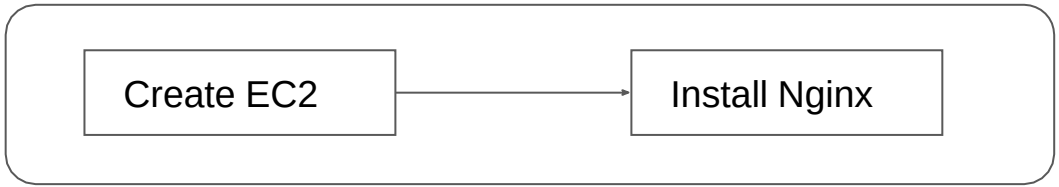
What if we want a complete end to end solution ?

Welcome to Terraform Provisioners

Provisioners are used to execute scripts on a local or remote machine as part of resource creationor destruction.

Let’s take an example:

On creation of Web-Server, execute a script which installs Nginx web-server.



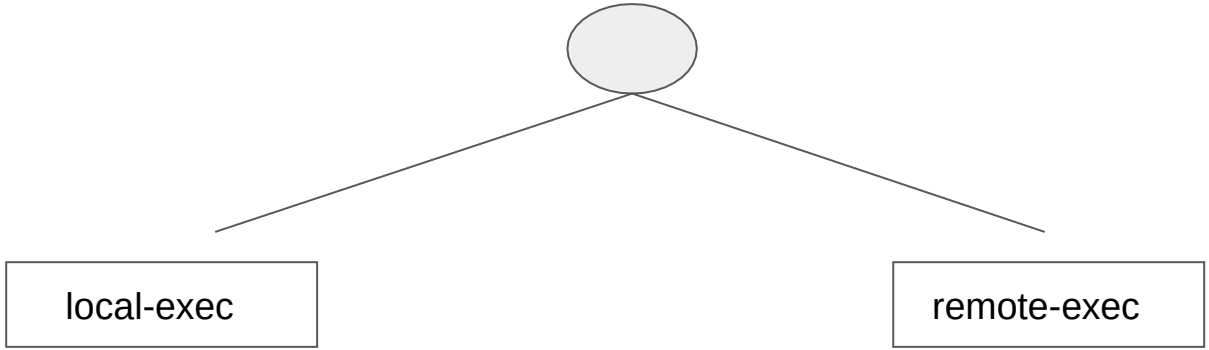
Types of Provisioners

Interesting Part is here

Provisioners are interesting

Terraform has capability to turn provisioners both at the time of resource creation as well as destruction.

There are two main types of provisioners:



Local Exec Provisioners

local-exec provisioners allow us to invoke local executable after resource is

created Let's take an example:

```
resource "aws_instance" "web" {
# ...

provisioner "local-exec" {
command = "echo ${aws_instance.web.private_ip} >> private_ips.txt"
}
}
```

Remote Exec Provisioners

Remote-exec provisioners allow to invoke scripts directly on the remote server.Let's take an example:

```
resource "aws_instance" "web" {
# ...

provisioner "remote-exec" {

……..

}
}
```

Provisioner Types

Terraform in detail

Overview of Provisioner Types

There are two primary types of provisioners:

Types of Provisioners	Description
Creation-Time Provisioner	Creation-time provisioners are only run during creation, not during updating or any other lifecycle If a creation-time provisioner fails, the resource is marked as tainted.
Destroy-Time Provisioner	Destroy provisioners are run before the resource is destroyed.

Destroy Time Provisioner

If when = destroy is specified, the provisioner will run when the resource it is defined within is destroyed.

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    when    = destroy
    command = "echo 'Destroy-time provisioner'"
  }
}
```

local-exec

Provisioners Time!

Provisioners are interesting

local-exec provisioners allows us to invoke a local executable after the resource is created.

One of the most used approach of local-exec is to run ansible-playbooks on the created serverafter the resource is created.

```
provisioner "local-exec" {
command = "echo ${aws_instance.web.private_ip} >> private_ips.txt"
}
```

Failure Behavior - Provisioners

Terraform in detail

Provisioner - Failure Behaviour

By default, provisioners that fail will also cause the terraform apply itself to fail.

on_failure setting can be used to change this. The allowed values are:

Allowed Values	Description
continue	Ignore the error and continue with creation or destruction.
fail	Raise an error and stop applying (the default behavior). If this is a creation provisioner, taint the resource.

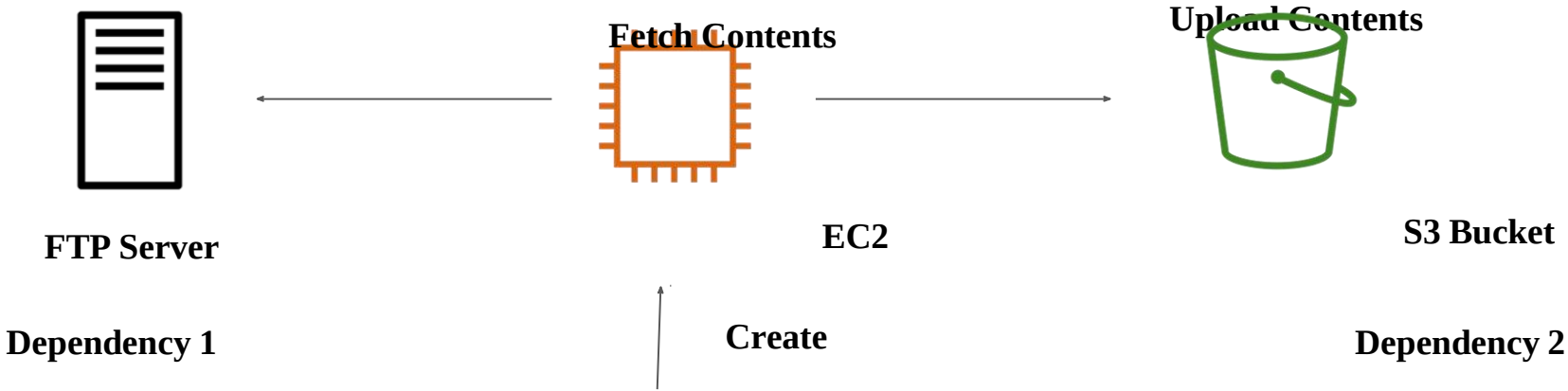
```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command    = "echo The server's IP address is ${self.private_ip}"
    on_failure = continue
  }
}
```

Null Resource

Terraform Function

Basics of Null Resource

The null_resource implements the standard resource lifecycle but takes no further action.



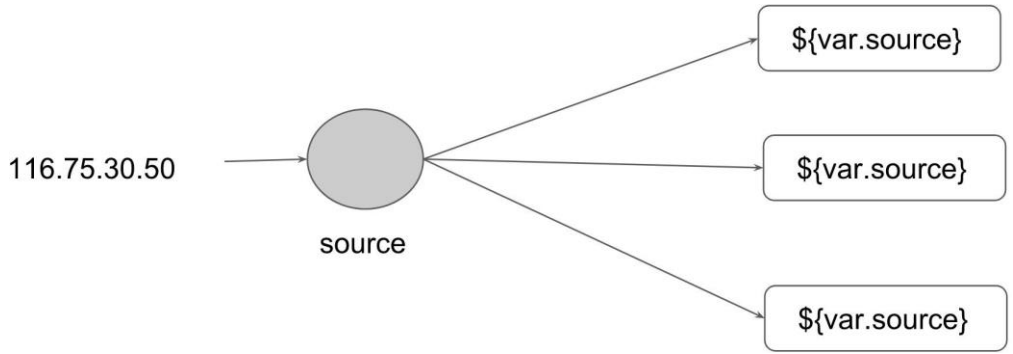
DRY Principle

Software Engineering

Understanding DRY Approach

In software engineering, don't repeat yourself (DRY) is a principle of software developmentaimed at reducing repetition of software patterns.

In the earlier lecture, we were making static content into variables so that there can be singlesource of information.



We are repeating resource code

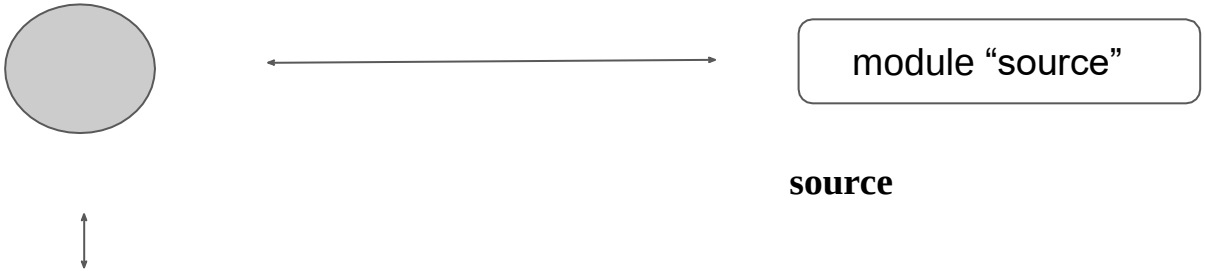
We do repeat multiple times various terraform resources for multiple projects.

Sample EC2 Resource

```
resource "aws_instance" "myweb" {
    ami = "ami-bf5540df"
    instance_type = "t2.micro"
    security_groups = ["default"]
  }
```

Centralized Structure

We can centralize the terraform resources and can call out from TF files whenever required.




```
resource "aws_instance" "myweb" {
    ami = "ami-bf5540df"
    instance_type = "t2.micro"
    security_groups = ["default"]
  }
```

Challenges with Modules

Software Engineering

Challenges

One common need on infrastructure management is to build environments like staging,production with similar setup but keeping environment variables different.

Staging

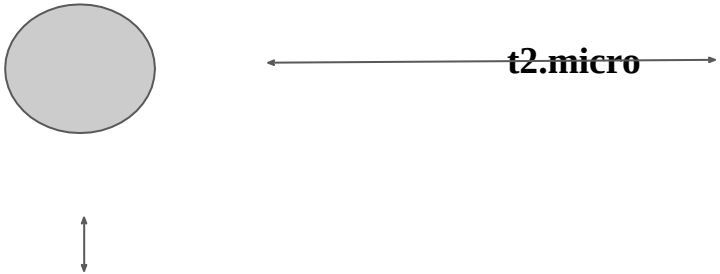
instance_type = t2.micro

Production

instance_type = m4.large

Challenges

When we use modules directly, the resources will be replica of code in the module.



t2.micro

Development

source

Staging

t2.small

Production

m4.large

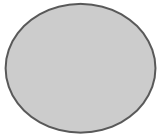
```
resource "aws_instance" "myweb" {
    ami = "ami-bf5540df"
    instance_type = "t2.micro"
    security_groups = ["default"]
}
```

Using Locals with Modules

Terraform Function

Understanding the Challenge

Using variables in Modules can also allow users to override the values which you mightnot want.



source

```
resource "aws_instance" "myweb" {

ami = "ami-bf5540df"

instance_type = var.instance_type

}
```

instance_type = m5.xlarge

Development

Staging

instance_type = m5.2xlarge

Production

instance_type = t2.medium

Setting the Context

There can be many repetitive values in modules and this can make your code difficult tomaintain.

You can centralize these using variables but users will be able to override it.

```
resource "aws_security_group" "elb-sg" {
  name        = "myelb-sg"


  ingress {
    description      = "Allow Inbound from Secret Application"
    from_port        = 8443
    to_port          = 8443
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }
```

Hardcoded Port


```
resource "aws_security_group" "elb-sg" {
  name        = "myelb-sg"


  ingress {
    description      = "Allow Inbound from Secret Application"
    from_port        = var.app_port
    to_port          = var.app_port
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }
```

Variable Port

Using Locals

Instead of variables, you can make use of locals to assign the values.

You can centralize these using variables but users will be able to override it.

```
resource "aws_security_group" "ec2-sg" {
  name        = "myec2-sg"

  ingress {
    description      = "Allow Inbound from Secret Application"
    from_port        = local.app_port
    to_port          = local.app_port
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

locals {
  app_port = 8443
}
```

Module Outputs

Output the Data

Revising Output Values

Output values make information about your infrastructure available on the commandline, and can expose information for other Terraform configurations to use.

```
output "instance_ip_addr" {
  value = aws_instance.server.private_ip
}
```

Accessing Child Module Outputs

In a parent module, outputs of child modules are available in expressions as module.<MODULE NAME>.<OUTPUT NAME>

```
resource "aws_security_group" "ec2-sg" {
  name        = "myec2-sg"

  ingress {
    description      = "Allow Inbound from Secret Application"
    from_port        = 8433
    to_port          = 8433
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

}

output "sg_id" {
  value = aws_security_group.ec2-sg.arn
}
```



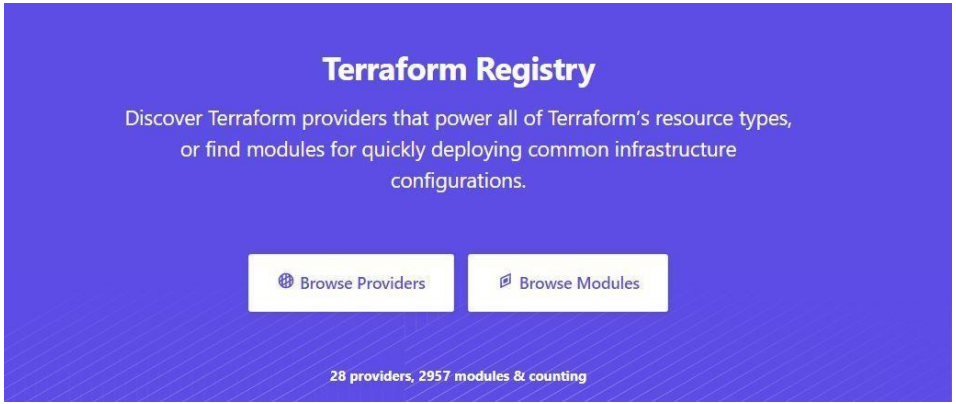
```
module "sgmodule" {
  source = "../../modules/sg"
}

resource "aws_instance" "web" {
  ami           = "ami-0ca285d4c2cda3300"
  instance_type = "t3.micro"
  vpc_security_group_ids = [module.sgmodule.sg_id]
}
```

Terraform Registry

Terraform in detail

Overview of Terraform Registry

The Terraform Registry is a repository of modules written by the Terraform community.The registry can help you get started with Terraform more quickly



Module Location

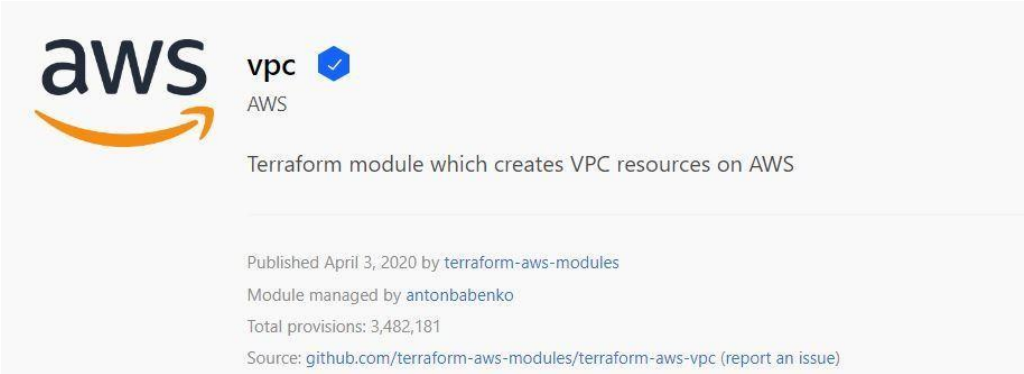
If we intend to use a module, we need to define the path where the module files are present.The module files can be stored in multiple locations, some of these include:

- Local Path
- GitHub
- Terraform Registry
- S3 Bucket
- HTTP URLs

Verified Modules in Terraform Registry

Within Terraform Registry, you can find verified modules that are maintained by various thirdparty vendors.

These modules are available for various resources like AWS VPC, RDS, ELB and others.

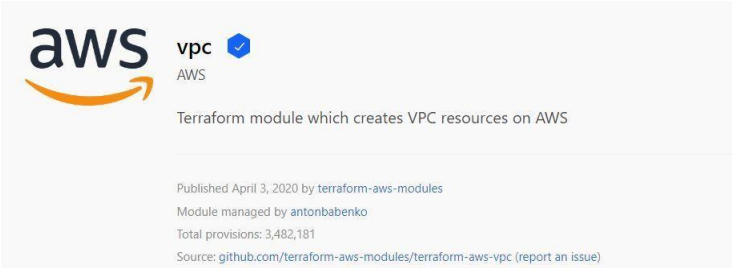


Verified Modules in Terraform Registry

Verified modules are reviewed by HashiCorp and actively maintained by contributors to stayup-to-date and compatible with both Terraform and their respective providers.

The blue verification badge appears next to modules that are verified.

Module verification is currently a manual process restricted to a small group of trustedHashiCorp partners.



Using Registry Module in Terraform

To use Terraform Registry module within the code, we can make use of the source argument thatcontains the module path.

Below code references to the EC2 Instance module within terraform registry.

```
module "ec2-instance" {
source  = "terraform-aws-modules/ec2-instance/aws"
version = "2.13.0"
# insert the 10 required variables here
}
```

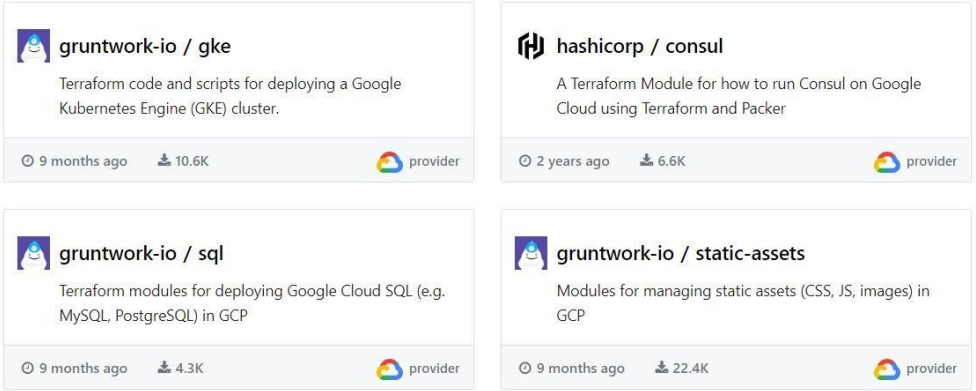
Publishing Modules

Publish Modules to Terraform Registry

Overview of Publishing Modules

Anyone can publish and share modules on the Terraform Registry.

Published modules support versioning, automatically generate documentation, allowbrowsing version histories, show examples and READMEs, and more.



Requirements for Publishing Module

Requirement	Description
GitHub	The module must be on GitHub and must be a public repo. This is only a requirement for the public registry.
Named	Module repositories must use this three-part name format terraform-<PROVIDER>-<NAME>
Repository description	The GitHub repository description is used to populate the short description ofthe module.
Standard module structure	The module must adhere to the standard module structure.
x.y.z tags for releases	The registry uses tags to identify module versions. Release tag names mustbe a semantic version, which can optionally be prefixed with a v. For example, v1.0.4 and 0.9.2

Standard Module Structure

The standard module structure is a file and directory layout that is recommend for reusable modules distributed in separate repositories

```
$ tree minimal-module/
.
├── README.md
├── main.tf
├── variables.tf
└── outputs.tf
```

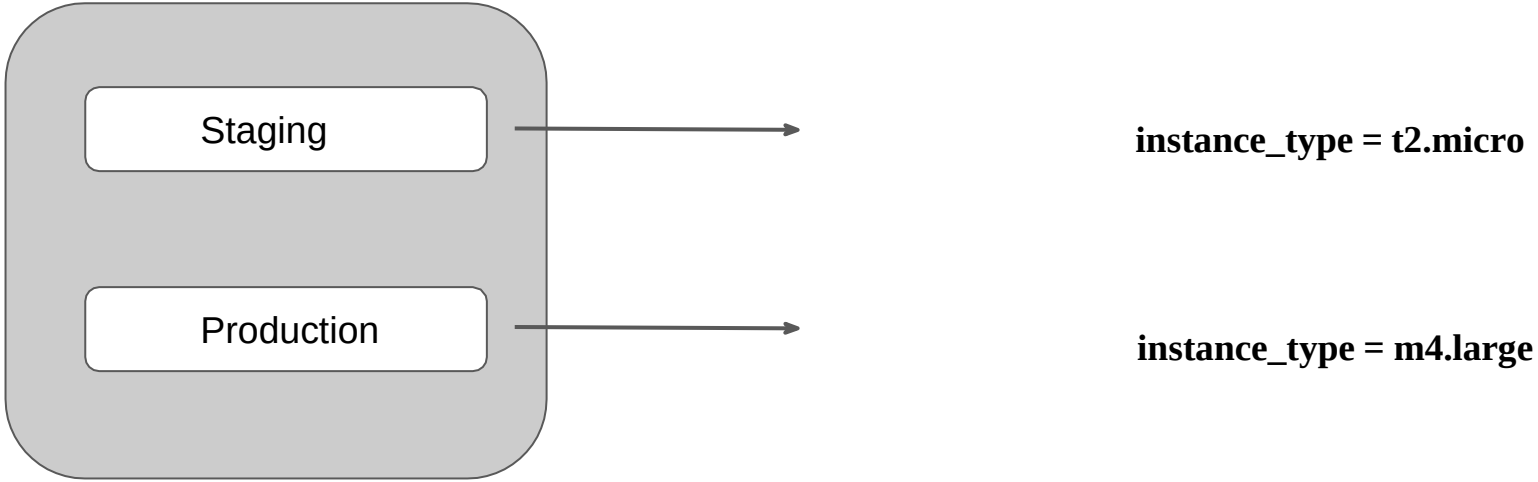
```
$ tree complete-module/
.
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
├── ...
├── modules/
│   ├── nestedA/
│   │   ├── README.md
│   │   ├── variables.tf
│   │   ├── main.tf
│   │   └── outputs.tf
│   ├── nestedB/
│   ├── .../
├── examples/
│   ├── exampleA/
│   │   └── main.tf
│   ├── exampleB/
│   ├── .../
```

Terraform Workspace

Interesting topics

Understanding WorkSpaces

Terraform allows us to have multiple workspaces, with each of the workspace we can havedifferent set of environment variables associated



Team Collaboration

Terraform in detail

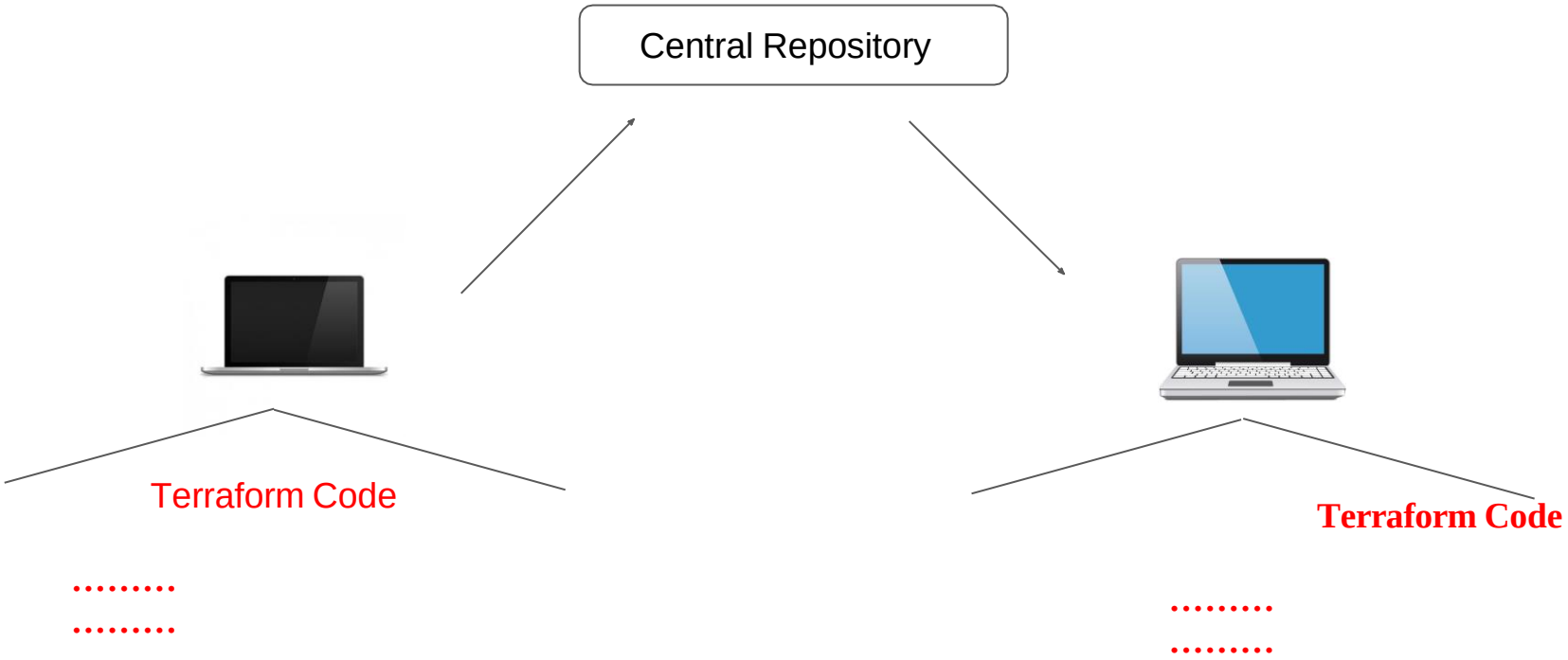
Local Changes are not always good

Currently we have been working with terraform code locally.

Terraform Code

.....
.....

Centralized Management



Terraform Module Sources

Terraform in detail

Supported Module Sources

The source argument in a module block tells Terraform where to find the source code for thedesired child module.

- Local paths
- Terraform Registry
- GitHub
- Bitbucket
- Generic Git, Mercurial repositories
- HTTP URLs
- S3 buckets
- GCS buckets

```
module "consul" {
  source = "app.terraform.io/example-corp/k8s-cluster/azurerm"
  version = "1.1.0"
}
```

Local Path

A local path must begin with either ./ or ../ to indicate that a local path is intended.

```
module "consul" {
  source = "../consul"
}
```

Git Module Source

Arbitrary Git repositories can be used by prefixing the address with the special git::prefix.

After this prefix, any valid Git URL can be specified to select one of the protocols supported by Git.

```
module "vpc" {
  source = "git::https://example.com/vpc.git"
}

module "storage" {
  source = "git::ssh://username@example.com/storage.git"
}
```

Referencing to a Branch

By default, Terraform will clone and use the default branch (referenced by HEAD) inthe selected repository.

You can override this using the ref argument:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

.

The value of the ref argument can be any reference that would be accepted by the gitcheckout command, including branch and tag names.

Terraform & GitIgnore

Terraform in detail

Overview of gitignore

The .gitignore file is a text file that tells Git which files or folders to ignore in a project.

.gitignore
conf/
*.artifacts
credentials

Terraform and .gitignore

Depending on the environments, it is recommended to avoid committing certain files to GIT.

Files to Ignore	Description
.terraform	This file will be recreated when terraform init is run.
terraform.tfvars	Likely to contain sensitive data like usernames/passwords and secrets.
terraform.tfstate	Should be stored in the remote side.
crash.log	If terraform crashes, the logs are stored to a file named crash.log

Terraform Backend

Terraform in detail

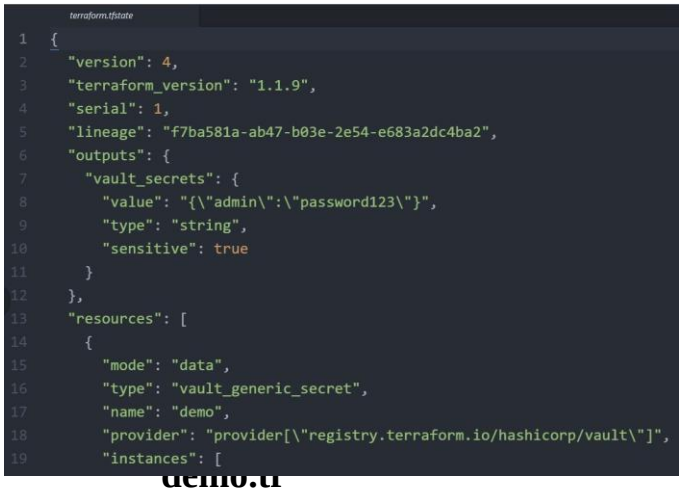
Basics of Backends

Backends primarily determine where Terraform stores its state.

By default, Terraform implicitly uses a backend called local to store state as a local file on disk.

```
provider "vault" {
  address = "http://127.0.0.1:8200"
}

data "vault_generic_secret" "demo" {
  path = "secret/db_creds"
}

output "vault_secrets" {
  value = data.vault_generic_secret.demo.data_json
  sensitive = "true"
}
```

demo.tf

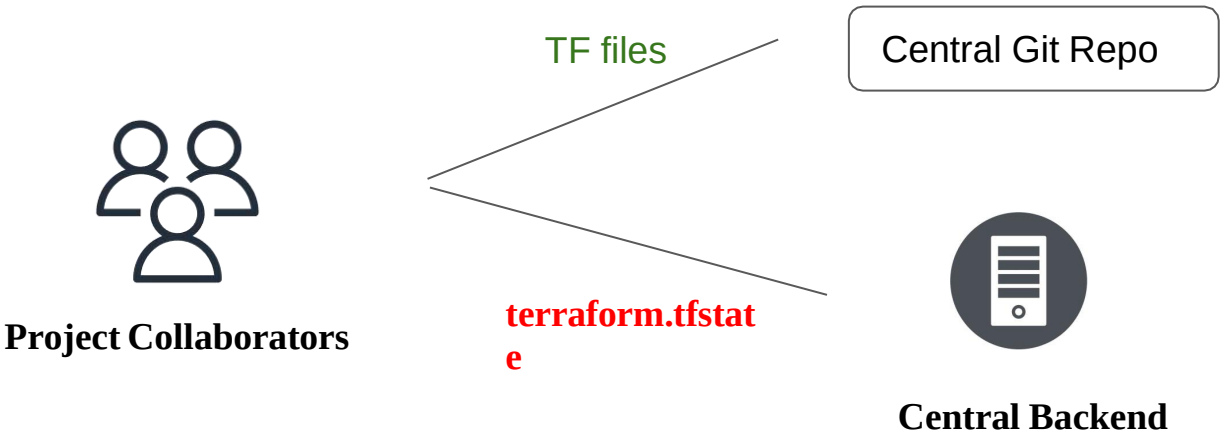


terraform.tfstate

Ideal Architecture

Following describes one of the recommended architectures:

1. **The Terraform Code is stored in Git Repository.**
2. **The State file is stored in a Central backend.**



Backends Supported in Terraform

Terraform supports multiple backends that allows remote service related operations.Some of the popular backends include:

- S3
- Consul
- Azurerm
- Kubernetes
- HTTP
- ETCD

Important Note

Accessing state in a remote service generally requires some kind of access credentials

Some backends act like plain "remote disks" for state files; others support locking the state whileoperations are being performed, which helps prevent conflicts and inconsistencies.

Terraform User

Store State File

Authenticate First

S3 Bucket

State Locking

Let’s Lock the State

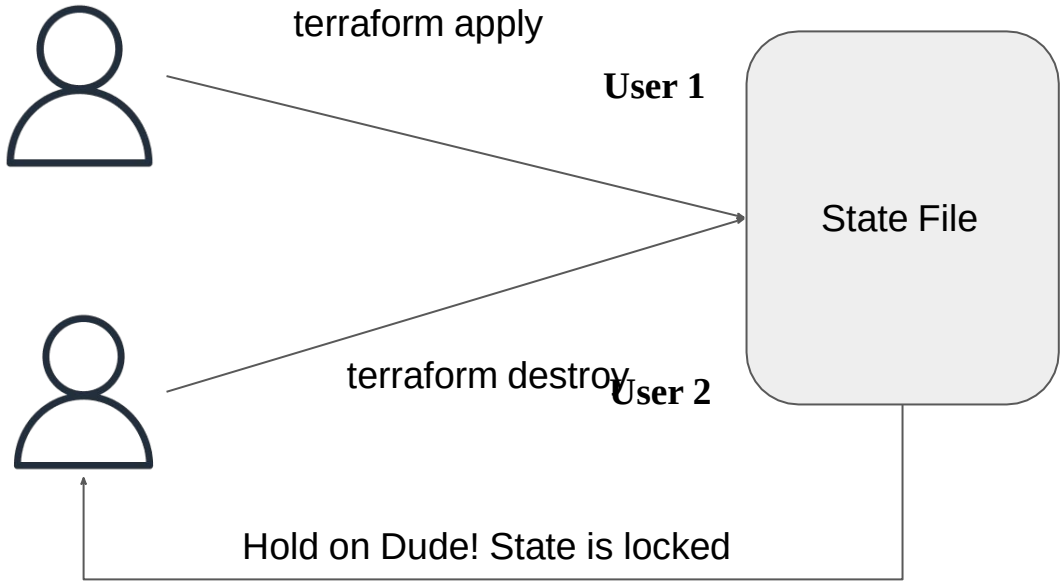
Understanding State Lock

Whenever you are performing write operation, terraform would lock the state file.

This is very important as otherwise during your ongoing terraform apply operations, if othersalso try for the same, it can corrupt your state file.

```
C:\Users\Zeal Vora\Desktop\tf-demo\remote-backend>terraform plan

Error: Error acquiring the state lock

Error message: Failed to read state file: The state file could not be read: read terraform.tfstate: The process
cannot access the file because another process has locked a portion of the file.

Terraform acquires a state lock to protect the state from being written
by multiple users at the same time. Please resolve the issue above and try
again. For most commands, you can disable locking with the "-lock=false"
flag, but this is not recommended.
```

Basic Working



Important Note

State locking happens automatically on all operations that could write state. You won't see any message that it is happening

If state locking fails, Terraform will not continue

Not all backends support locking. The documentation for each backend includes details onwhether it supports locking or not.

Force Unlocking State

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

If you unlock the state when someone else is holding the lock it could cause multiple writers.

Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed.

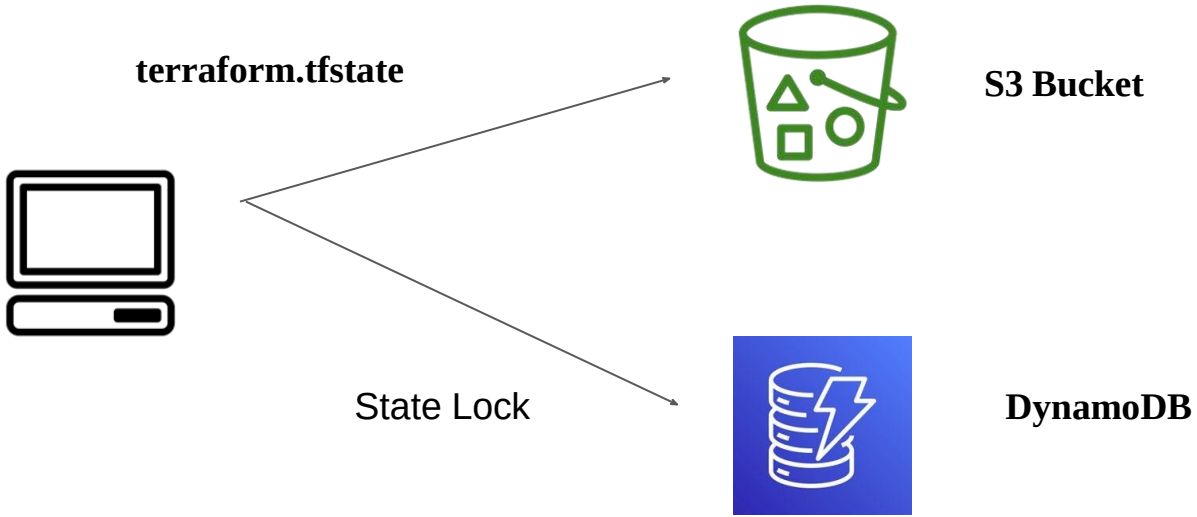
State Locking in S3 Backend

Back to Providers

State Locking in S3

By default, S3 does not support State Locking functionality.

You need to make use of DynamoDB table to achieve state locking functionality.



Terraform State Management

Advanced State Management

Overview of State Modification

As your Terraform usage becomes more advanced, there are some cases where you may need tomodify the Terraform state.

It is important to never modify the state file directly. Instead, make use of terraform statecommand.

Overview of State Modification

There are multiple sub-commands that can be used with terraform state, these include:

State Sub Command	Description
list	List resources within terraform state file.
mv	Moves item with terraform state.
pull	Manually download and output the state from remote state.
push	Manually upload a local state file to remote state.
rm	Remove items from the Terraform state
show	Show the attributes of a single resource in the state.

Sub Command - List

The terraform state list command is used to list resources within a Terraform state.

```
bash-4.2# terraform state list
aws_iam_user.lb
aws_instance.webapp
```

Sub Command - Move

The terraform state mv command is used to move items in a Terraform state.

This command is used in many cases in which you want to rename an existing resource without destroying and recreating it.

Due to the destructive nature of this command, this command will output a backup copy of thestate prior to saving any changes

Overall Syntax:

terraform state mv [options] SOURCE DESTINATION

Sub Command - Pull

The terraform state pull command is used to manually download and output the state fromremote state.

This is useful for reading values out of state (potentially pairing this command with somethinglike jq).

Sub Command - Push

The terraform state push command is used to manually upload a local state file to remote state.

This command should rarely be used.

Sub Command - Remove

The terraform state rm command is used to remove items from the Terraform

state.Items removed from the Terraform state are not physically destroyed.

Items removed from the Terraform state are only no longer managed by Terraform

For example, if you remove an AWS instance from the state, the AWS instance will continuerunning, but terraform plan will no longer see that instance.

Sub Command - Show

The terraform state show command is used to show the attributes of a single resource in the Terraform state.

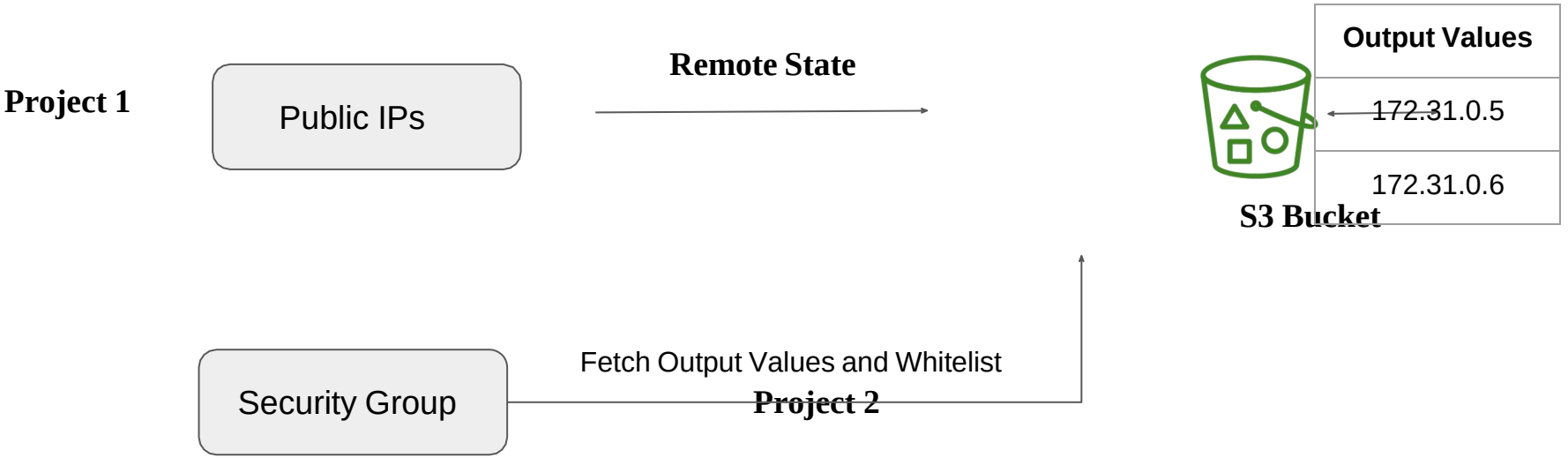
```
bash-4.2# terraform state show aws_instance.webapp
# aws_instance.webapp:
resource "aws_instance" "webapp" {
    ami                          = "ami-082b5a644766e0e6f"
    arn                          = "arn:aws:ec2:us-west-2:018721151861:instance/i-0107ea9ed06c467e0"
    associate_public_ip_address  = true
    availability_zone            = "us-west-2b"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0107ea9ed06c467e0"
    instance_state               = "running"
    instance_type                = "t2.micro"
```

Connecting Remote States

Terraform in detail

Basics of Terraform Remote State

The terraform_remote_state data source retrieves the root module output values from someother Terraform configuration, using the latest state snapshot from the remote backend.



Step 1 - Create a Project with Output Values & S3 Backend

```
resource "aws_eip" "lb" {
  vpc      = true
}

output "eip_addr" {
  value = aws_eip.lb.public_ip
}
```



```
terraform {
  backend "s3" {
    bucket = "kplabs-terraform-backend"
    key    = "network/eip.tfstate"
    region = "us-east-1"
  }
}
```

Step 2 - Reference Output Values from Different Project

```
data "terraform_remote_state" "eip" {
  backend = "s3"
  config = {
    bucket = "kplabs-terraform-backend"
    key    = "network/eip.tfstate"
    region = "us-east-1"
  }
}
```



```
resource "aws_security_group" "allow_tls" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic"

  ingress {
    description      = "TLS from VPC"
    from_port        = 443
    to_port          = 443
    protocol         = "tcp"
    cidr_blocks      = ["${data.terraform_remote_state.eip.outputs.eip_addr}/32"]
  }

}
```

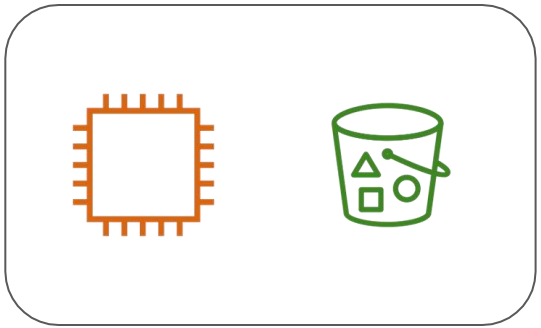
Terraform Import

Bring Infrastructure Under Terraform

Typical Challenge

It can happen that all the resources in an organization are created manually.

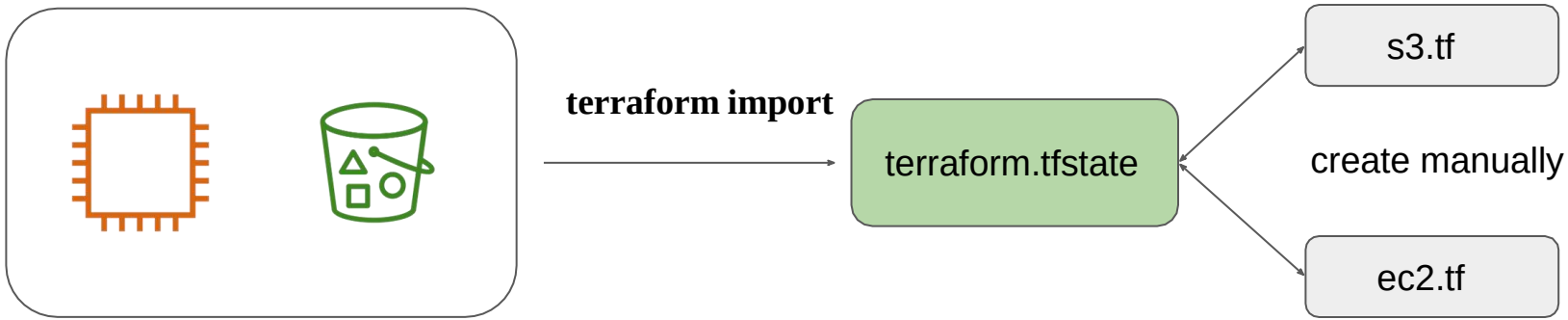
Organization now wants to start using Terraform and manage these resources via Terraform.



Terraform Import

Terraform is able to import existing infrastructure.

This allows you take resources you've created by some other means and bring it underTerraform management.



Important Pointer

The current implementation of Terraform import can only import resources into the state. Itdoes not generate configuration.

A future version of Terraform will also generate configuration.

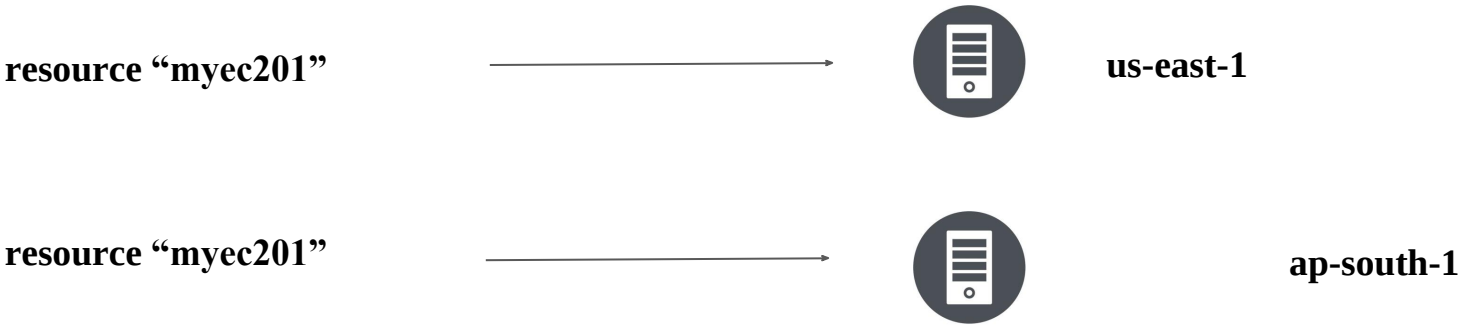
Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped.

Provider Configuration

Terraform in detail

Single Provider Multiple Configuration

Till now, we have been hardcoding the aws-region parameter within the providers.tf This means that resources would be created in the region specified in the providers.tf file.



Single Provider Multiple Configuration

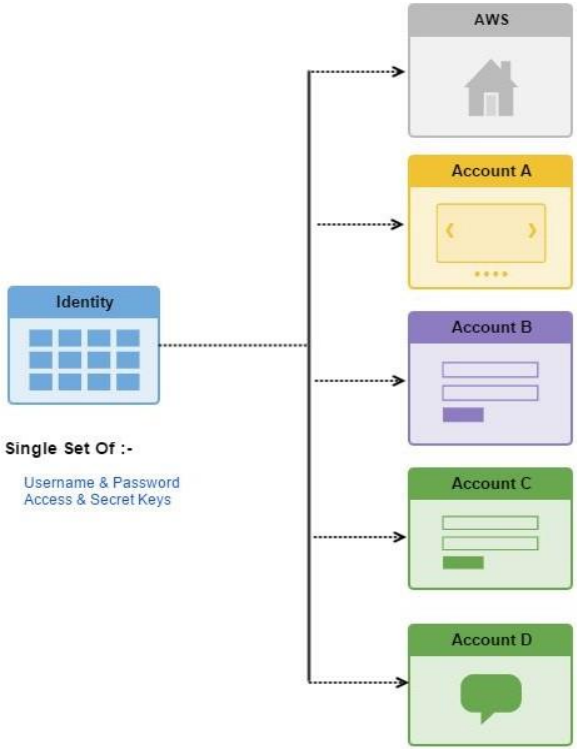
resource “myec201” → **Account 01**

resource “myec201” → **Account 02**

Terraform with STS

Terraform in detail

Definitive Use-Case



Sensitive Parameter

Terraform Security

Overview of Sensitive Parameter

With organization managing their entire infrastructure in terraform, it is likely that you will seesome sensitive information embedded in the code.

When working with a field that contains information likely to be considered sensitive, it is best toset the Sensitive property on its schema to true

```
output "db_password" {
  value       = aws_db_instance.db.password
  description = "The password for logging in to the database.
  sensitive   = true
}
```

Overview of Sensitive Parameter

Setting the sensitive to “true” will prevent the field's values from showing up in CLI output andin Terraform Cloud

It will not encrypt or obscure the value in the state, however.

```
C:\Users\Zeal Vora\Desktop\terraform\sensitive data>terraform apply

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

Outputs:

db_password = <sensitive>
```

Overview of Vault

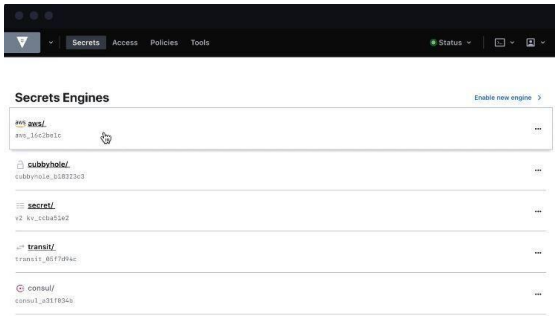
HashiCorp Certified: Vault Associate

Let’s get started

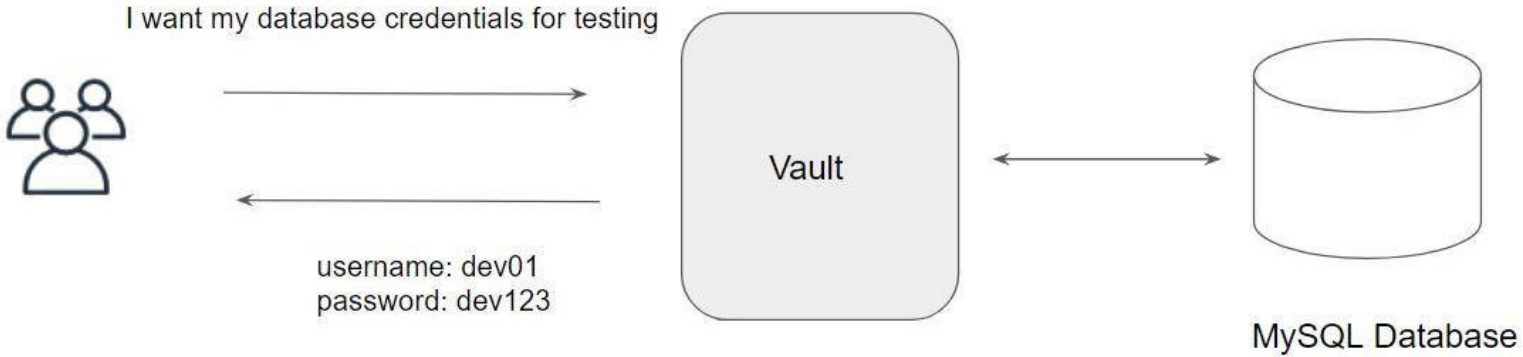
HashiCorp Vault allows organizations to securely store secrets like tokens, passwords, certificatesalong with access management for protecting secrets.

One of the common challenges nowadays in an organization is “Secrets Management”

Secrets can include, database passwords, AWS access/secret keys, API Tokens, encryption keysand others.



Dynamic Secrets



Once Vault is integrated with multiple backends, your life will become much easier and you canfocus more on the right work.

Major aspect related to Access Management can be taken over by vault.

Vault Provider

Back to Providers

Vault Provider

The Vault provider allows Terraform to read from, write to, and configure HashiCorpVault.

Important Note

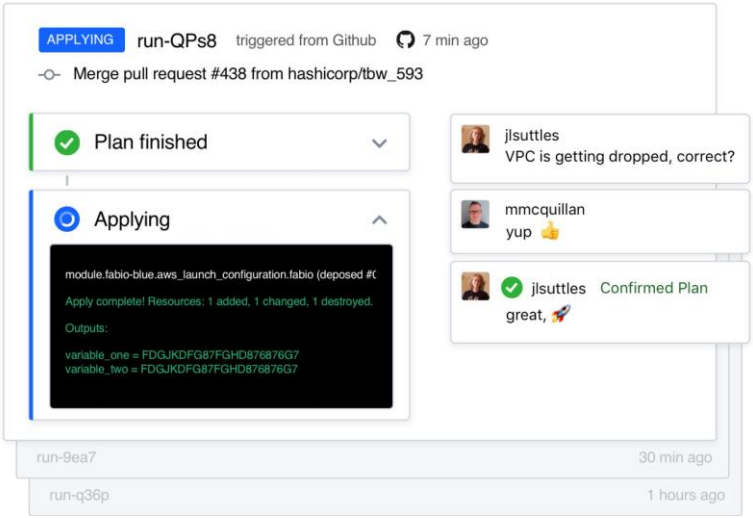
Interacting with Vault from Terraform causes any secrets that you read and write to bepersisted in both Terraform's state file.

Terraform Cloud

Terraform in detail

Overview of Terraform Cloud

Terraform Cloud manages Terraform runs in a consistent and reliable environment with variousfeatures like access controls, private registry for sharing modules, policy controls and others.



Sentinel

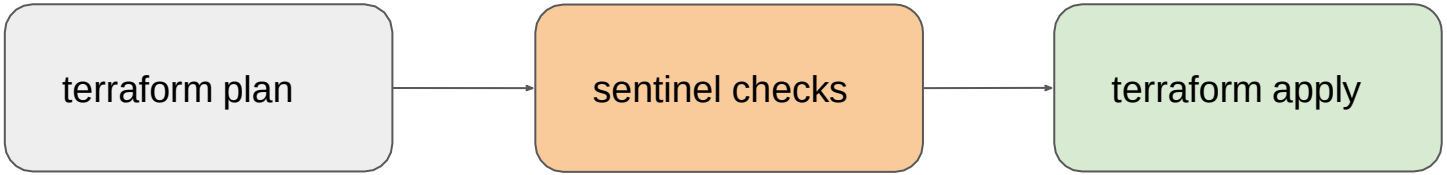
Terraform Cloud In Detail

Overview of the Sentinel

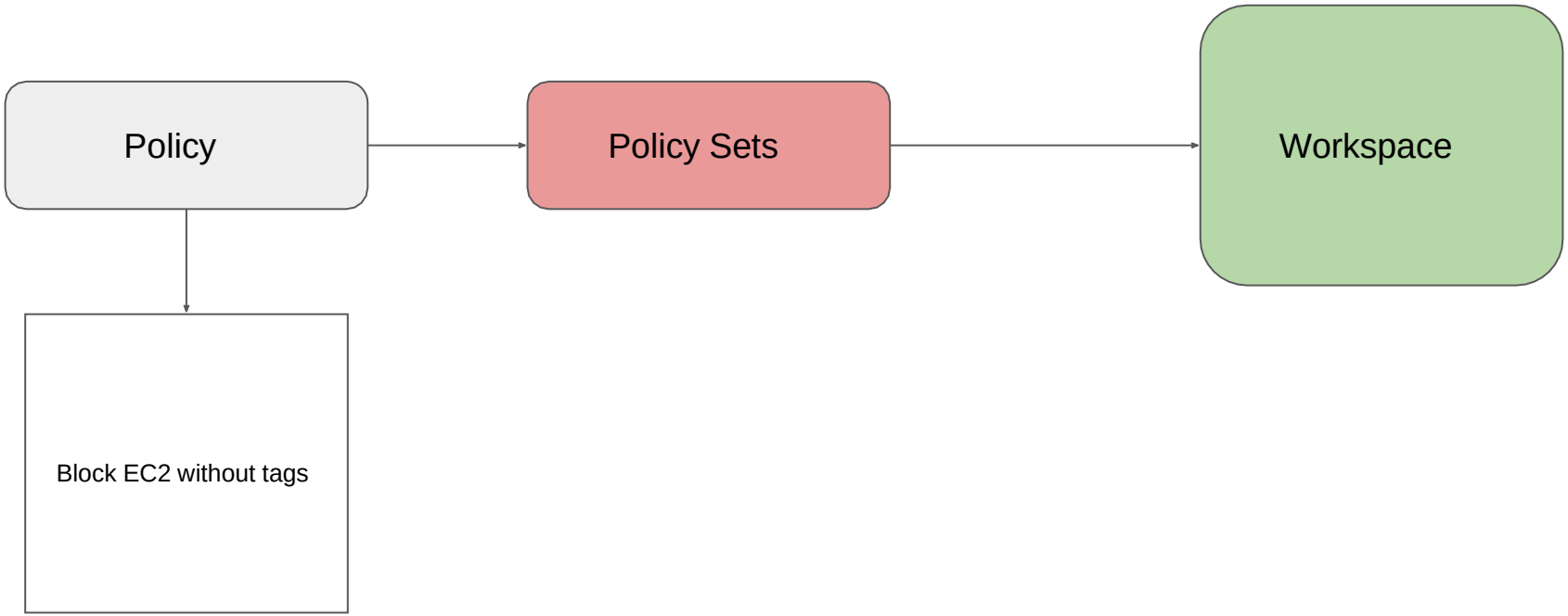
Sentinel is a policy-as-code framework integrated with the HashiCorp Enterprise products.

It enables fine-grained, logic-based policy decisions, and can be extended to use informationfrom external sources.

Note: Sentinel policies are paid feature



High Level Structure



Terraform Backend

Terraform in detail

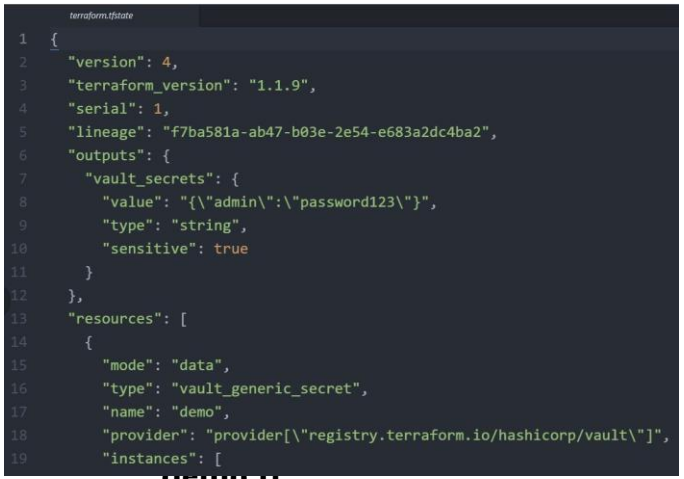
Basics of Backends

Backends primarily determine where Terraform stores its state.

By default, Terraform implicitly uses a backend called local to store state as a local file on disk.

```
provider "vault" {
  address = "http://127.0.0.1:8200"
}

data "vault_generic_secret" "demo" {
  path = "secret/db_creds"
}

output "vault_secrets" {
  value = data.vault_generic_secret.demo.data_json
  sensitive = "true"
}
```

demo.tf



terraform.tfstate

Challenge with Local Backend

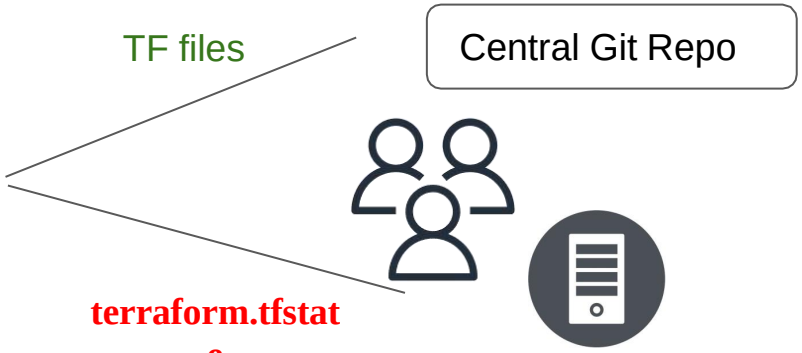
Nowadays Terraform project is handled and collaborated by an entire team.

Storing the state file in the local laptop will not allow collaboration.

Ideal Architecture

Following describes one of the recommended architectures:

1. **The Terraform Code is stored in Git Repository.**
2. **The State file is stored in a Central backend.**



Backends Supported in Terraform

Terraform supports multiple backends that allows remote service related operations.Some of the popular backends include:

- S3
- Consul
- Azurerm
- Kubernetes
- HTTP
- ETCD

Important Note

Accessing state in a remote service generally requires some kind of access credentials

Some backends act like plain "remote disks" for state files; others support locking the state whileoperations are being performed, which helps prevent conflicts and inconsistencies.

Terraform User

Store State File

Authenticate First

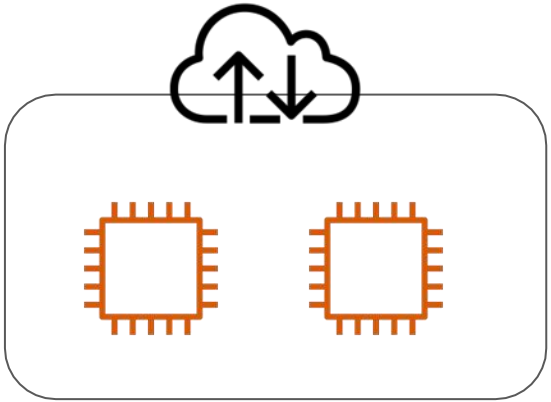
S3 Bucket

Air Gapped Environments

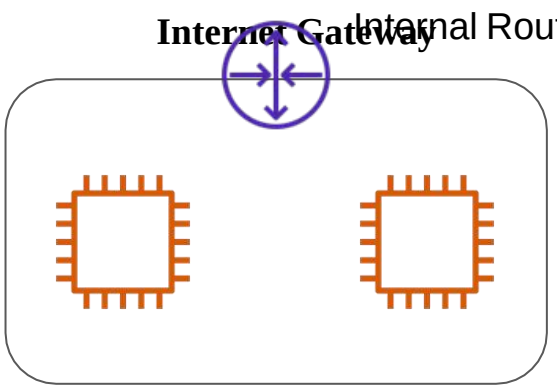
Installation Methods

Understanding Concept of Air Gap

An air gap is a network security measure employed to ensure that a secure computer networkis physically isolated from unsecured networks, such as the public Internet.



Internet Connectivity



Air Gapped System

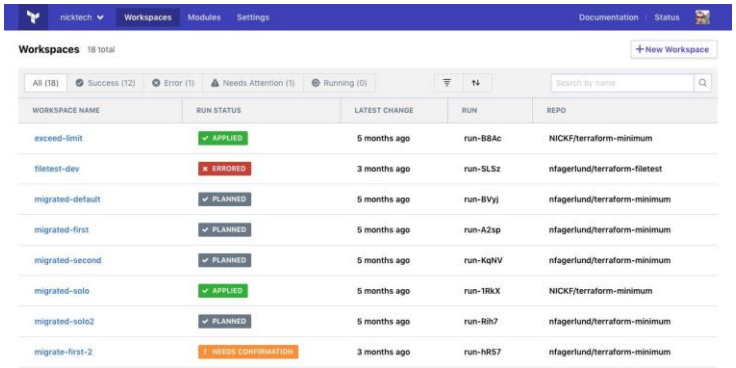
Usage of Air Gapped Systems

Air Gapped Environments are used in various areas. Some of these include:

- **Military/governmental computer networks/systems**
- **Financial computer systems, such as stock exchanges**
- **Industrial control systems, such as SCADA in Oil & Gas fields**

Terraform Enterprise Installation Methods

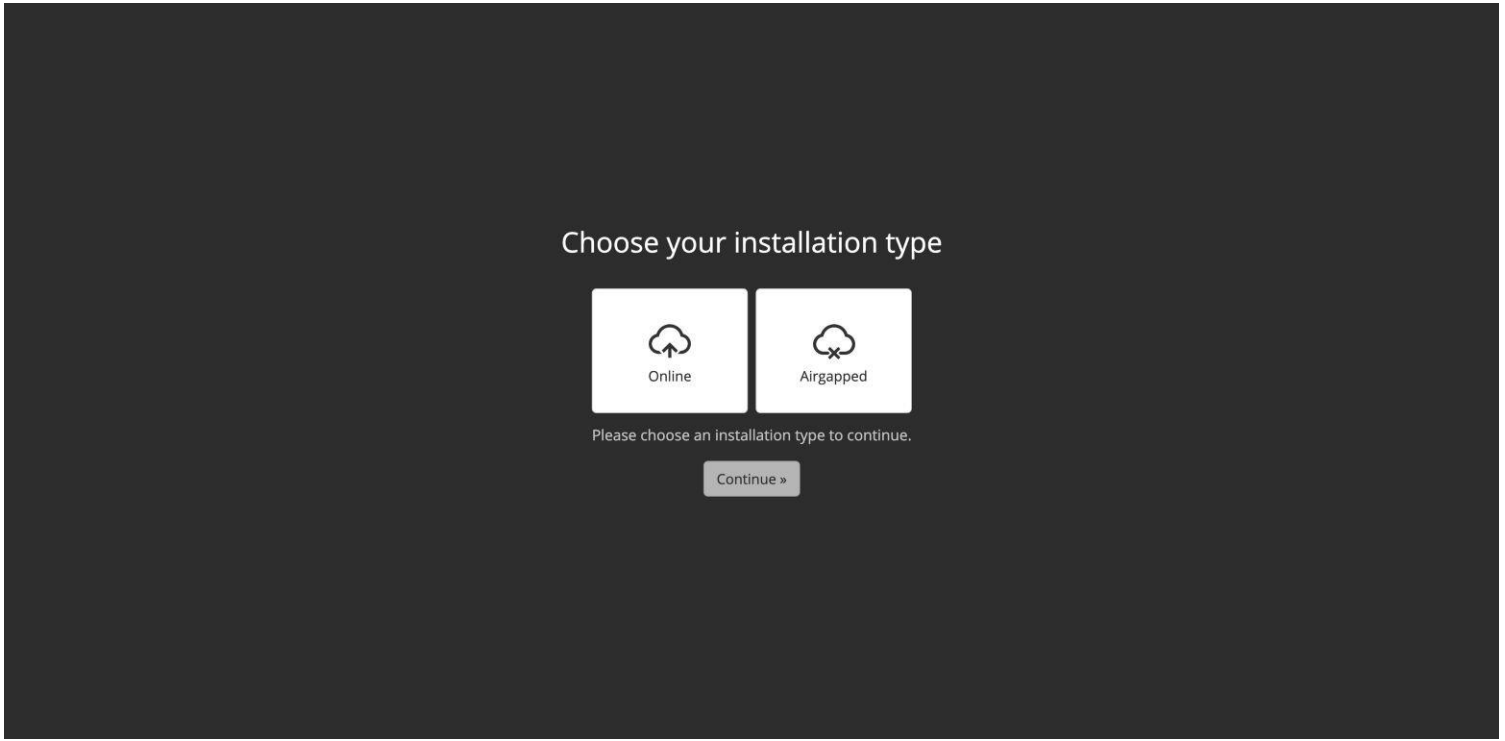
Terraform Enterprise installs using either an online or air gapped method and as thenames infer, one requires internet connectivity, the other does not



Terraform Enterprise

Air Gap Install

Isolated Server


Choose your installation type
Online
Airgapped
Please choose an installation type to continue.
Continue »


Provide path or upload airgap bundle
Provide absolute path on this server to archive file
e.g. /mnt/installers/package.airgap
Continue »
Select file for upload
Upload Airgap Bundle
To upload an app bundle, file must have a .airgap extension.
« Back

Overview of HashiCorp Exams

Let’s Get Certified!

Overview of HashiCorp Associate Exams

Overview of the basic exam related information.

Assessment Type	Description
Type of Exams	Multiple Choice
Format	Online Proctored
Duration	1 hour
Questions	57
Price	70.50 USD + Taxes
Language	English
Expiration	2 years

Multiple Choice

This includes various sub-formats, including:

- **True or False**
- **Multiple Choice**
- **Fill in the blank**

Delta Type of Question

Example 1:

Demo Software stores information in which type of backend?

Format - Online Proctored

Important Rules to be followed:

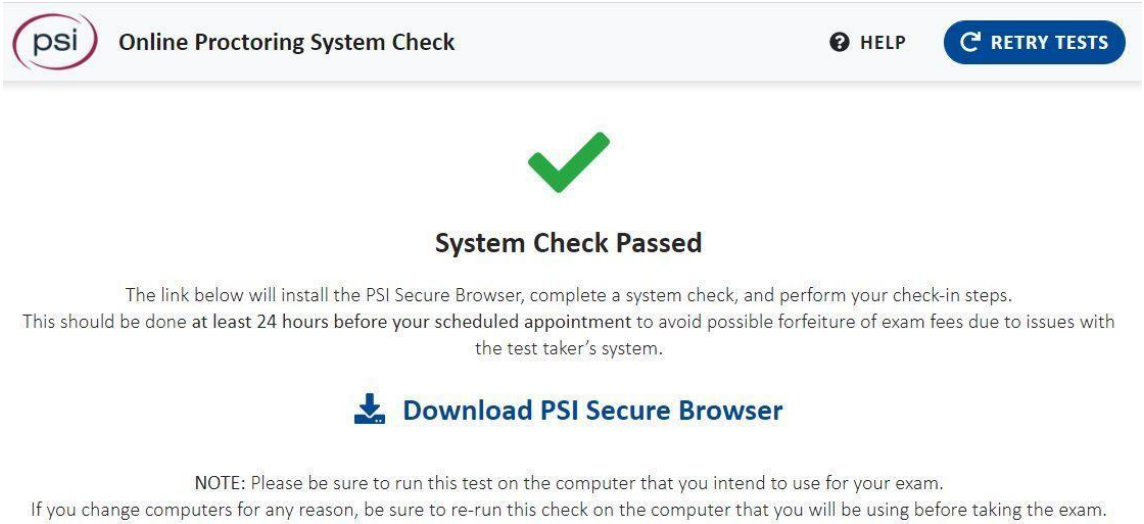
- You are alone in the room
- Your desk and work area are clear
- You are connected to a power source
- No phones or headphones
- No dual monitors
- No leaving your seat
- No talking
- Webcam, speakers, and microphone must remain on throughout the test.
- The proctor must be able to see you for the duration of the test.

Registration Process

The high-level steps for registering for the exams are as follows:

1. **Login to the HashiCorp Certification Page.**
2. **Register for Exams.**
3. **Check System Requirements**
4. **Download PSI Software**
5. **Best of Luck & Good Luck!**

Make sure to complete system check.



Online Exam

HashiCorp Certified: Consul Associate - Scheduled for Test

EXAM DATE:
Nov 24, 2020

START TIME:
05:30 PM

TIME ZONE:
Asia/Kolkata

EXAM DURATION:
60 minutes

- Before taking your remote online proctored exam, please check system compatibility - click HERE
- You can only launch the exam within 30 minutes of your appointment time.

You may launch your test in...
3 Hours

Launch Exam

View Details

Terraform commands

Initialization and Configuration commands :

- terraform init: Prepares a working directory for Terraform use, downloading plugins and initializing backend configuration.
- terraform validate: Checks configuration files for syntax errors and potential issues.
- terraform fmt: Reformats configuration files to adhere to Terraform's style guidelines, ensuring consistency and readability.

Planning and Previewing Changes commands:

terraform plan: Generates a preview of the changes Terraform will make to infrastructure, allowing review before execution.

Applying and Destroying Infrastructure:

- terraform apply: Applies the changes previewed in the plan, creating or updating infrastructure.
- terraform destroy: Destroys infrastructure managed by Terraform.

Targeting Specific Resources commands :

- terraform apply -target=resource_address: Applies changes only to the specified resource or module.
- terraform destroy -target=resource_address: Destroys only the specified resource or module.

State Management:

- terraform state list: Lists resources managed by Terraform in the current workspace.
- terraform state show resource_address: Displays detailed information about a specific resource.
- terraform state pull: Retrieves the latest state from remote storage.
- terraform state push: Pushes the current state to remote storage.
- terraform state refresh: Updates the state file to reflect any changes made outside of Terraform.
- terraform taint resource_address: Marks a resource as tainted, forcing Terraform to recreate it on the next apply.
- terraform untaint resource_address: Removes the tainted state from a resource.

Workspace Management:

- terraform workspace new workspace_name: Creates a new workspace.
- terraform workspace select workspace_name: Switches to an existing workspace.
- terraform workspace list: Lists available workspaces.

Version Control:

- terraform get: Installs or upgrades Terraform modules from a registry.

General Commands:

terraform console: Enters an interactive REPL for experimenting with Terraform expressions.

- terraform graph: Generates a Graphviz graph of the planned execution order.
- terraform providers: Lists the providers required for the configuration.
- terraform version: Displays the currently installed Terraform version.
- terraform force-unlock LOCK_ID: Forcibly unlocks a locked state file.

Additional Tips:

- **Use `-auto-approve` with `apply` and `destroy` to skip interactive confirmation.
- **Specify variables with `-var` or `-var-file`.
- **Pass environment variables with `TF_VAR_` prefix.
- **Control parallelism with `-parallelism`.
- **Lock the state file during operations with `-lock=true` (default).
- **Consult Terraform's documentation for detailed usage and examples.

Advanced Terraform commands:

- terraform import: Associates existing infrastructure with a Terraform resource definition.
- terraform output: Outputs values from your configuration at runtime.
- terraform remote-state: Manages state remotely.
- terraform module: Defines reusable components of infrastructure.
- terraform plan -destroy-output: Generates a plan showing resources to be destroyed.
- terraform taint -regex: Taints resources matching a regular expression.
- terraform taint -except: Excludes specific resources from a taint operation.
- terraform login: Stores credentials for remote hosts.
- terraform logout: Removes stored credentials for remote hosts.

Troubleshooting and Testing:

- terraform show: Shows the current state or a saved plan.
- terraform state diff: Compares state versions to identify changes.
- terraform plan -destroy: Outputs a plan for destroying existing infrastructure.
- terraform state verify: Verifies the stored state against existing infrastructure.

Other useful options:

- -no-color: Suppress color output in terminal.
- -debug: Enable debug logging for detailed execution information.
- -input=true: Wait for user input before taking specific actions.
- -state-file=filename: Use a custom state file location.

Learning resources:

- Terraform documentation: https://developer.hashicorp.com/terraform
- Terraform Cheat Sheet: https://docs.spacelift.io/vendors/terraform/
- HashiCorp Learn Platform: https://developer.hashicorp.com/tutorials
- Pluralsight Terraform course: https://www.pluralsight.com/resources/blog/cloud/the-ultimate-terraform-cheatsheet

https://www.linkedin.com/in/lokeshkumar-aws-devops

Follow us on

LinkedIn

Thank you