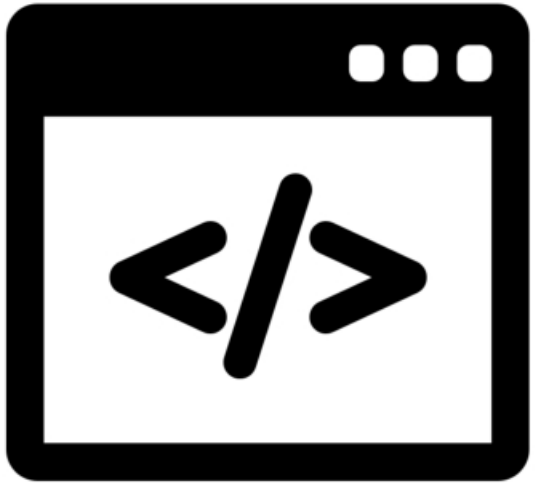
IBM Cloud Automation Manager

Intro to Terraform

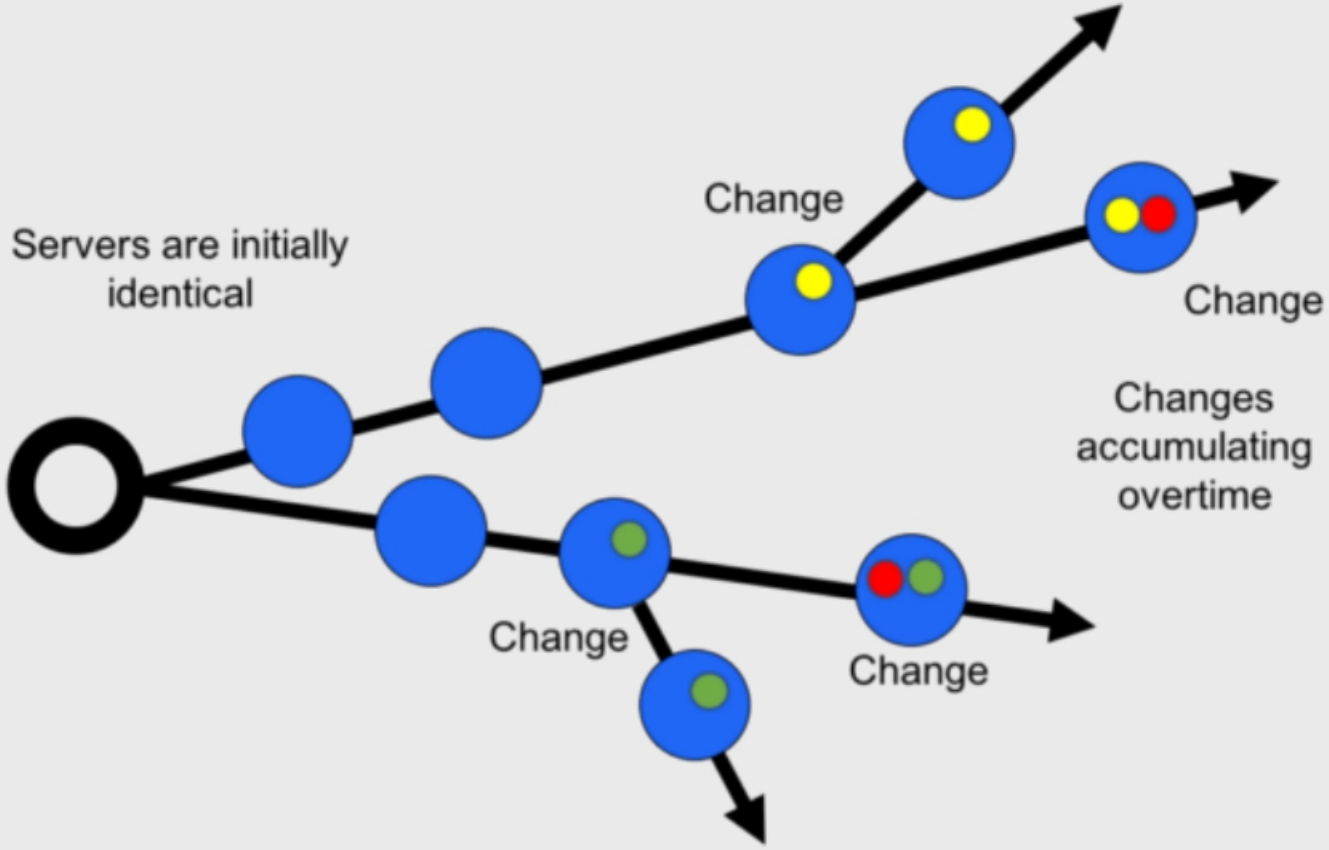
IBM Cloud

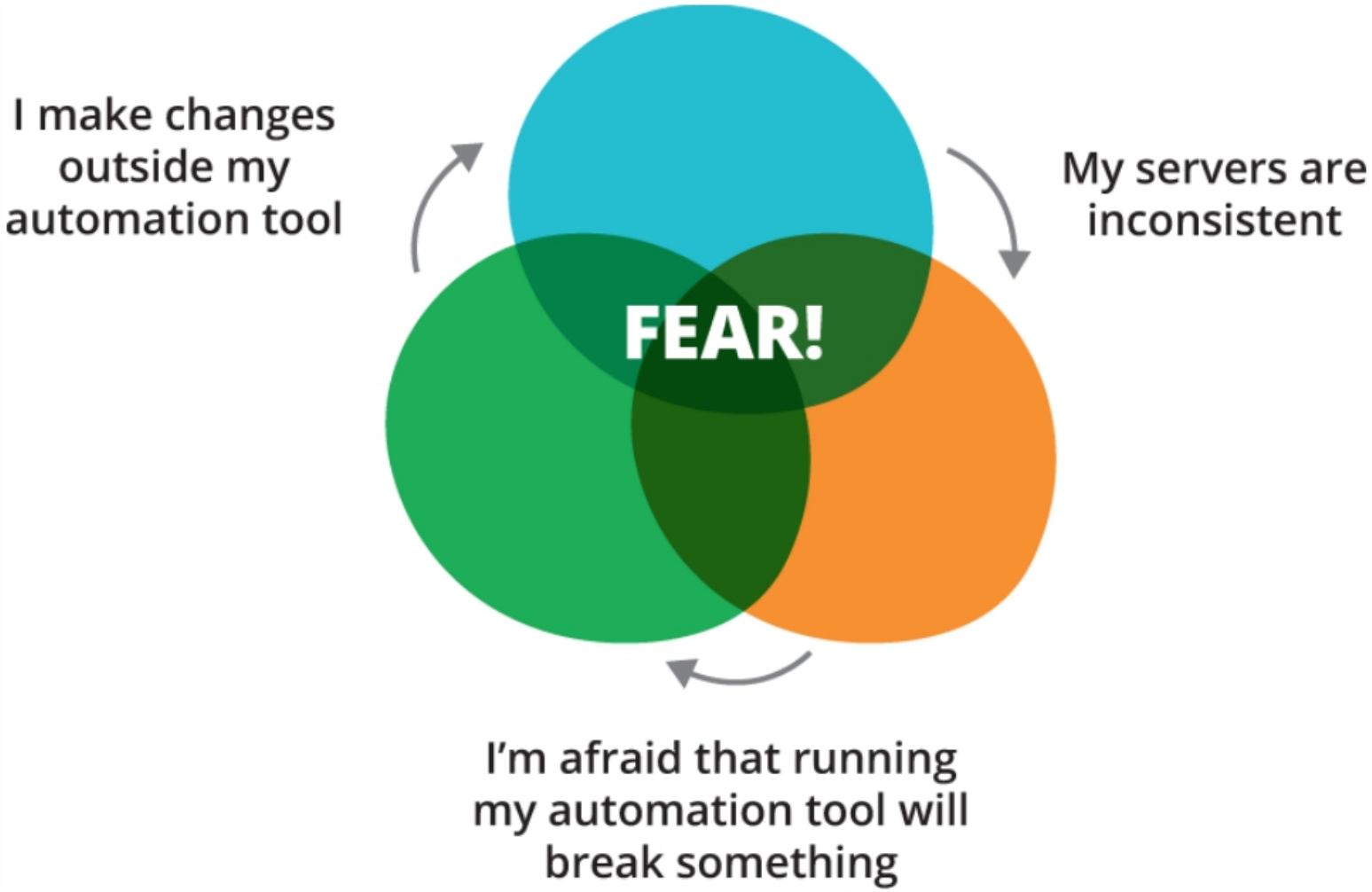


Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The value of IaC can be broken down into three measurable categories: cost (reduction), speed (faster execution) and risk (remove errors and security violations).[

Configuration Drift





The Automation Fear Spiral

What's infrastructure ?

The first generation of these IaC tools, like Puppet and Chef, focused heavily on configuring operating systems and applications but not the hosts, services, and networking that they were built on

Traditionally the Cloud services, hosts, virtual machines, Docker containers, networking *(routing, switching, firewalls),* and storage were considered infrastructure

Now infrastructure also includes more complex services or Software-as-a-Service products delivered by third parties such as DNS, Content Delivery Networks *(CDN),* databases, job scheduling, queues, K8s, monitoring.... Terraform can configure components like GitHub organizations and repositories, Grafana monitoring console,.....

Terraform

Providers, Data Sources and Resources

HashiCorp

Terraform

Write, Plan, and Create Infrastructure as Code

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently

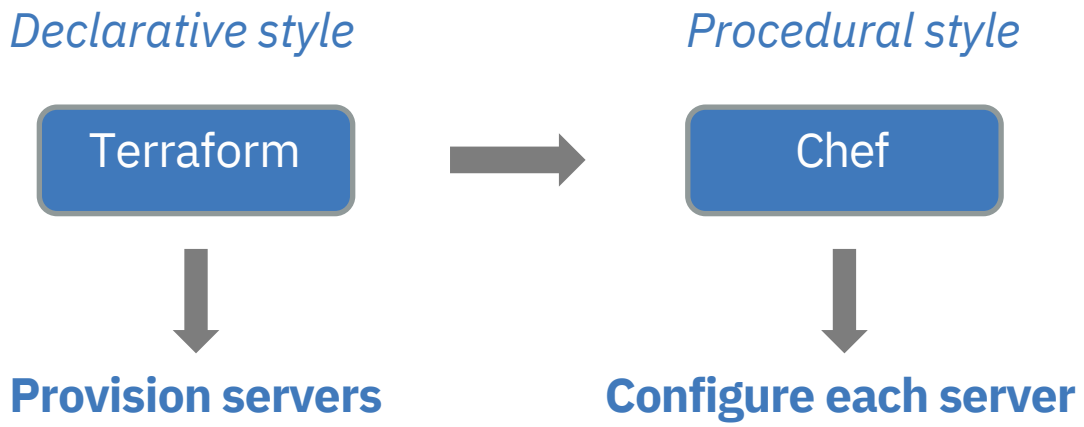
Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

- *https://www.terraform.io/*
- *https://github.com/hashicorp/terraform*

Declarative definitions style to define resources to provision

The Terraform language is declarative, describing an intended goal rather than the steps to reach that goal.

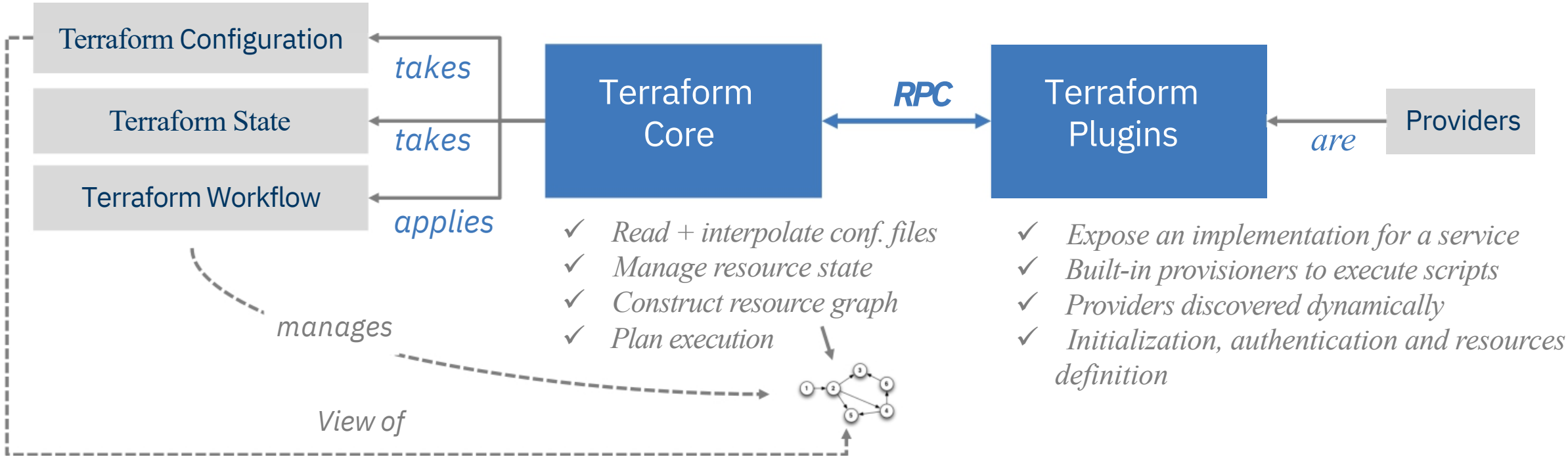
- ✓ With declarative definitions you specify what should be there and not how to do
- ✓ Only missing parts must be created, existing ones must be in the desired state, and obsolete ones must be destroyed
- ✓ Ensure the correct order of creation
- ✓ Integrate with other tools such as Chef
- ✓ Platform agnostic
- ✓ Update management
- ✓ Extension capabilities



How Terraform operates ?

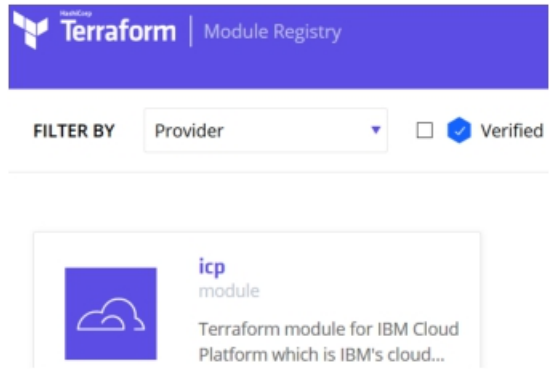
Terraform examines each resource and uses a graph-based approach to model and apply the desired state.

Each resource is placed inside the graph, its relationships with other resources are calculated, and then each resource is automatically built in the correct order to produce your infrastructure.

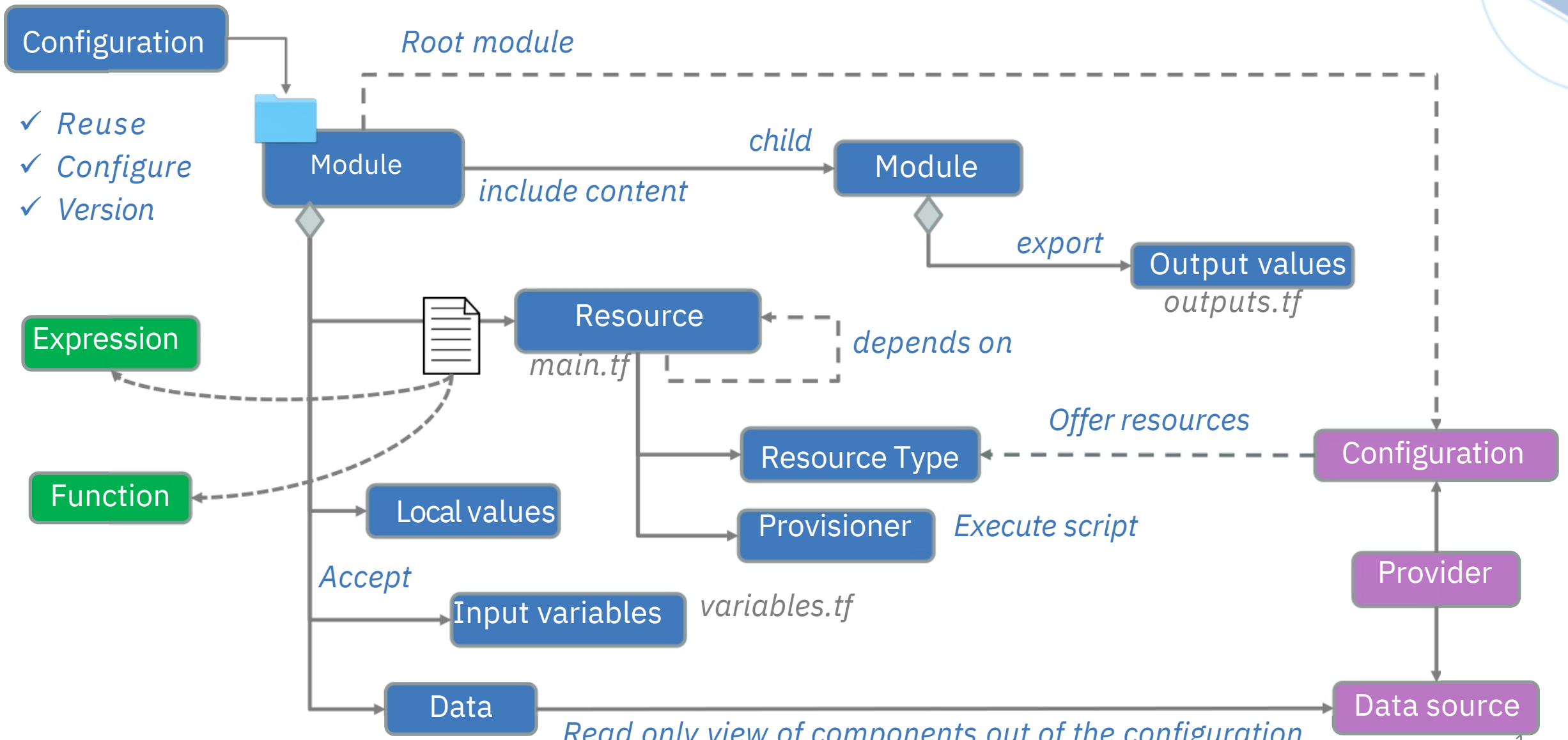


Introduction & Concepts

- You define **resources** as code in Terraform templates
  - ✓ Specify the provider
  - ✓ Specify provisioners
  - ✓ Specify the resources
  - ✓ Parameterize your template using **variables**
- **Provider**: a source of resources with API endpoint & authentication
- **Provisioner** execute local or remote scripts during resource creation or destroy time
- **Data Source**: information read from provider
- **Modules** allows to reuse the same code in different environments
  - ✓ Modules should be sourced from git tags / branches
  - ✓ Module registry
- You follow a **Workflow**:
  - ✓ **Plan** => see what you are about to deploy
  - ✓ **Apply** => to apply the changes
- Terraform generates a **state file**:
  - ✓ Store information about your managed infrastructure and configuration *(apply execution result)*
  - ✓ Generated during the apply stage, don't edit manually
- A **backend** determines how a state is loaded and how operations are executed

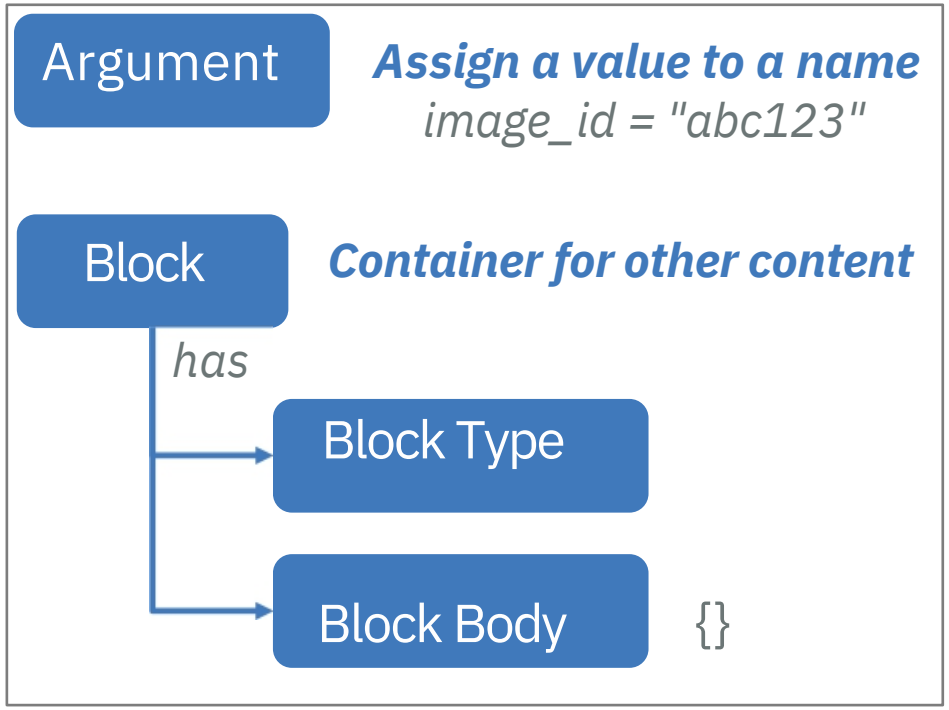


Key configuration language elements



1

Terraform Syntax



type labels

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

Top level block type
- ✓ Resource
- ✓ Input variable
- ✓ Output value
- ✓ Data source

comment

```
variable "variable_name" {
  type = "variable_type"
  default = "variable_default_value" # optional
}
```

- ✓ string
- ✓ map
- ✓ list
- ✓ boolean

```
resource "aws_instance" "web" {
  ami           = "${var.ami}"
  instance_type = "t2.micro"

  tags {
    Name = "HelloWorld"
  }
}
```

Variable interpolation
- ✓ Evaluate the expression
- ✓ Convert the value to a string
- ✓ Insert it into the string

Providers

The provider block is used to configure the named provider

A provider is responsible for creating and managing resources

Multiple provider blocks can exist if a Terraform configuration is composed of multiple providers

Credential can be passed in different ways (provider dependent)

```
provider "vsphere" {
        version = "~> 1.11.0"
        vsphere_server = "var.vsphere_server"

        # if you have a self-signed cert
        allow_unverified_ssl = "var.allow_unverified_ssl"
}
```

https://www.terraform.io/docs/configuration/providers.html

Data Sources

Used to discover information about existing objects

Can be nested in on other data objects

```
data "vsphere_datacenter" "dc" {
        name = "var.vsphere_datacenter"
}

data "vsphere_datastore_cluster" "datastore_cluster" {
        name = "var.datastore_cluster"
        datacenter_id = "data.vsphere_datacenter.dc.id"
}

data "vsphere_resource_pool" "pool" {
        name =  "var.vsphere_cluster/Resources/var.vsphere_resource_pool"
        datacenter_id = "data.vsphere_datacenter.dc.id"
}

data "vsphere_network" "network" {
        name = "var.network_label"
        datacenter_id = "data.vsphere_datacenter.dc.id"
}

data "vsphere_virtual_machine" "template" {
        name = "var.template"
        datacenter_id = "data.vsphere_datacenter.dc.id"
}
```

https://www.terraform.io/docs/configuration/data-sources.html

Resources

Defines a resource that exists within the infrastructure

May be a physical component such as an EC2 instance, or it can be a logical resource such as an SSH key

Resource definition is described by the provider

```
resource "vsphere_virtual_machine" ”camlab" {
        name = "terraform-test"
        resource_pool_id =        "data.vsphere_resource_pool.pool.id"
        datastore_id = "data.vsphere_datastore.datastore.id"

        num_cpus = 2
        memory = 1024
        guest_id = "other3xLinux64Guest"

        network_interface {
                network_id = "data.vsphere_network.network.id"
        }

        disk {
                label = "disk0"
                size = 20
        }
}
```

https://www.terraform.io/docs/configuration/resources.html

Defining Variables

All input variables must be defined in variable blocks

Multiple types of variables string, map, list, etc.

Can define a default or leave default black

Variables that are not defined will need to be defined a run time

```
variable "vsphere_server" {
        description = "vsphere server to connect to"
        default = ”10.0.0.210"
}


variable "camlab" {
        type = "map"

        default = {
                nodes = "1"
                vcpu = "2"
                memory = "4096"
        }
}


variable "dns_servers" {
        description = "DNS Servers to configure on VMs"
        default = ["8.8.8.8", "8.8.4.4"]
}
```

https://www.terraform.io/docs/configuration/variables.html

Referencing Variables

Input variables are referenced in the “var.<name>” syntax

Data variables are referenced in the “data.<name>” syntax

Local variables are referenced in the “local.<name>” syntax

```
vsphere_server = "var.vsphere_server"

num_cpus = "var.camlab["vcpu"]"

path = "local.team_folder"
```

Terraform

Interpolation, modules and provisioners

Modules

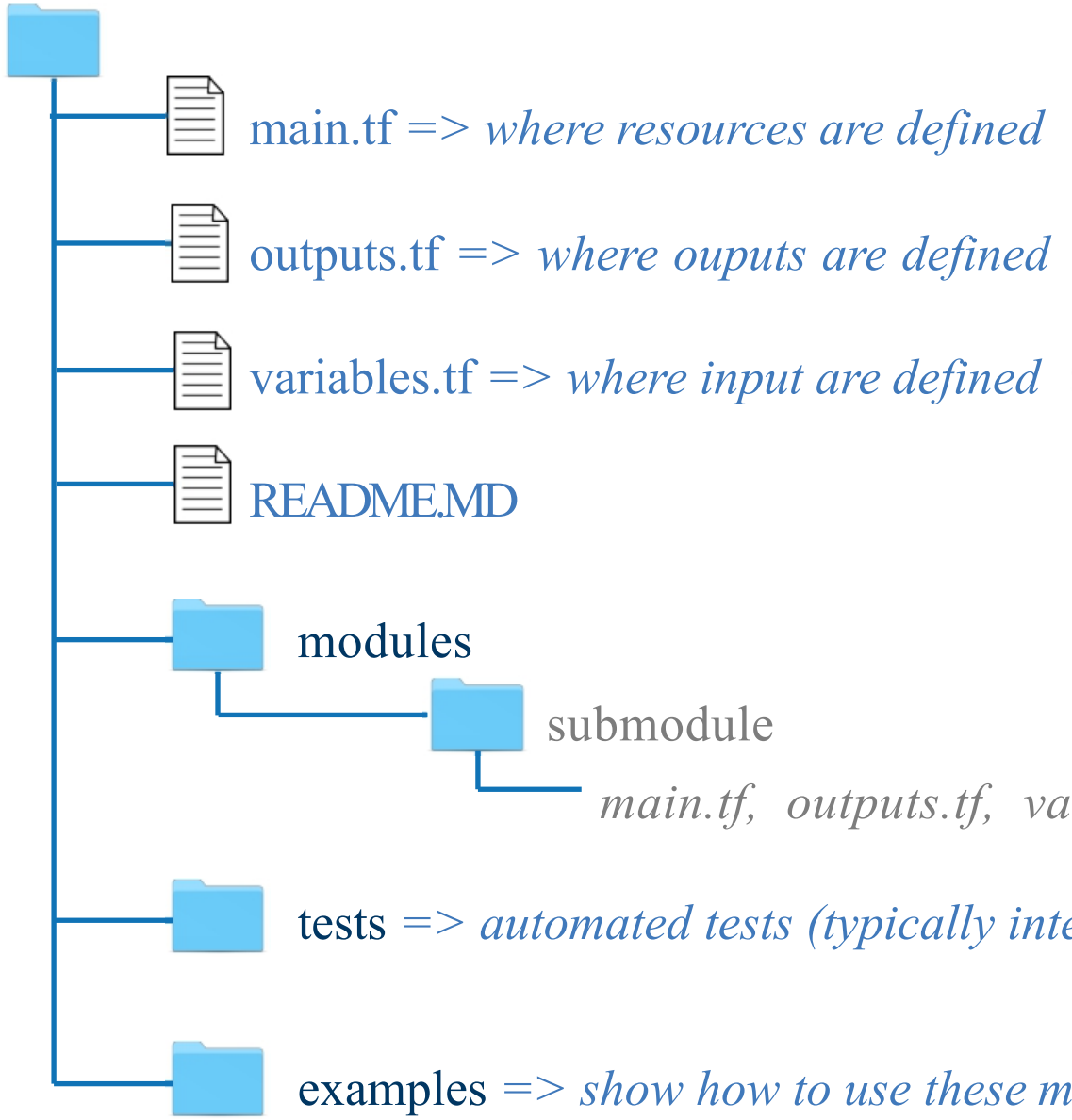
- ✓ A module is a container for multiple resources that are used together.
- ✓ Modules are used to abstract common configurations as building blocks which can be packaged and reused by you and the rest of the organization
- ✓ Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory.
- ✓ A module can call other modules, which lets you include the child module's resources into the configuration in a concise way.

```
module "example-server-linuxvm-withdatadisk" {
  source              = "Terraform-VMWare-Modules/vm/vsphere"
  version             = "0.9.2"
  vmtemp              = "TemplateName"
  instances           = 1
  vmname              = "example-server-windows"
  vmrp                = "esxi/Resources"
  vlan                = "Name of the VLAN in vSphere"
  data_disk           = "true"
  data_disk_size_gb = 20
  dc                  = "Datacenter"
  ds_cluster          = "Data Store Cluster name"
}

module "example-server-windowsvm-withdatadisk" {
  source              = "Terraform-VMWare-Modules/vm/vsphere"
  version             = "0.9.2"
  vmtemp              = "TemplateName"
  instances           = 1
  vmname              = "example-server-windows"
  vmrp                = "esxi/Resources"
  vlan                = "Name of the VLAN in vSphere"
  data_disk           = "true"
  data_disk_size_gb = 20
  is_windows_image    = "true"
  dc                  = "Datacenter"
  ds_cluster          = "Data Store Cluster name"
  winadminpass        = "Str0ngP@ssw0rd!"
}
```

https://www.terraform.io/docs/configuration/modules.html

Module structure




```
output "url" {
    value = " http://${icp_instance.ip}:${var.port}"
}
```

```
variable "name" {
    description = " The name of the instance"
}
```

Go library

```
func vaultTest (…,…,…) {
    terratest.Apply(options)
}
```

Condition Expression syntax

```
CONDITION ? TRUEVAL : FALSEVAL
```

Example

```
resource "aws_instance" "web" {
  subnet = "${var.env == "production" ? var.prod_subnet : var.dev_subnet}"
}
```

https://www.terraform.io/docs/configuration/expressions.html

Interpolation

Functions

```
variable "environment" {
default = {
    "test" = "us-east-1"
    "prod" = "us-west-2"
  }
}

variable "availzone" {
  description = "Availability Zones Mapping"
  default = {
    "us-east-1" = "us-east-1a,us-east-1b,us-east-1c"
    "us-west-2" = "us-west-2a,us-west-2b,us-east-1c"
  }
}

output "availabiltyzones" {
  value = "${element(split(",", lookup(var.availzone,var.environment.prod)), 1)}"
}


Output: availabiltyzones = us-west2b
```

https://www.terraform.io/docs/configuration/functions.html

Misc. Examples

Format the name variable to be lower case and use two digit numeral

```
name = "${format("${lower(var.instance_name)}-camlab%02d", count.index + 1 + (var.team_number * 100)) }"
```

Calculate the IP address based on the team number

```
ipv4_address = "${var.staticipblock != "0.0.0.0/0" ? cidrhost(var.staticipblock, (var.team_number * 10) +
var.staticipblock_offset + count.index) : ""}"
```

https://www.terraform.io/docs/configuration/functions.html

Provisioners

Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction. Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.

```
# Specify the ssh connection
connection {
          user = "var.ssh_user"
          password = "var.ssh_password"
          host = "${var.staticipblock != "0.0.0.0/0" ? cidrhost(var.staticipblock, (var.team_number * 10) +
          var.staticipblock_offset + count.index) : ""}"
}

provisioner "file" {
          source = "${path.module}/scripts"
          destination = "/tmp/terraform_scripts"
}

provisioner "remote-exec" {
          inline = [
                    "sudo chmod u+x /tmp/terraform_scripts/*.sh", "/tmp/terraform_scripts/add-public-ssh-key.sh
                    \"${tls_private_key.ssh.public_key_openssh}\"", "/tmp/terraform_scripts/add-private-ssh-key.sh
                    \"${tls_private_key.ssh.private_key_pem}\" \"${var.ssh_user}\""
          ]
}
```

https://www.terraform.io/docs/provisioners/index.html

Terraform

Project layout

Basic Project Layout

Sample Layout

- Instances and resources are defined in the **instances.tf** file
- Output variables are defined the **outputs.tf**
- Input variables defined in the **terraform.tfvars** file
- Variables are defined in the **variables.tf** file

Note: You can defined your Terraform in one big .tf file, but usually makes sense to separate the files by function

```
drwxr-xr-x   6 john  staff   192 Jul  8 04:45 .
drwxr-xr-x  78 john  staff  2496 Jul  8 04:40 ..
-rw-r--r--   1 john  staff  4883 Jul  8 04:41 instances.tf
-rw-r--r--   1 john  staff   196 Jul  8 04:41 outputs.tf
-rw-r--r--   1 john  staff  1184 Jul  8 04:45 terraform.tfvars
-rw-r--r--   1 john  staff  4301 Jul  8 04:41 variables.tf
```

Project Initialization

terraform init

- Used to initialize a working directory containing Terraform configuration files
- Will download the provider binaries defined in the configuration files (version specific)
- Creates the .terraform directory in the working directory

```
➜ terraform terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "vsphere" (terraform-providers/vsphere) 1.11.0...
- Downloading plugin for provider "tls" (terraform-providers/tls) 2.0.1...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.tls: version = "~> 2.0"

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.
```

```
drwxr-xr-x   7 john  staff   224 Jul  8 04:52 .
drwxr-xr-x  78 john  staff  2496 Jul  8 04:40 ..
drwxr-xr-x   3 john  staff    96 Jul  8 04:52 .terraform
-rw-r--r--   1 john  staff  4883 Jul  8 04:41 instances.tf
-rw-r--r--   1 john  staff   196 Jul  8 04:41 outputs.tf
-rw-r--r--   1 john  staff  1184 Jul  8 04:45 terraform.tfvars
-rw-r--r--   1 john  staff  4301 Jul  8 04:41 variables.tf
```

```
➜ terraform ls -l ./.terraform/plugins/darwin_amd64
total 137488
-rwxr-xr-x  1 john  staff       160 Jul  8 04:52 lock.json
-rwxr-xr-x  1 john  staff  23414136 Jul  8 04:52 terraform-provider-tls_v2.0.1_x4
-rwxr-xr-x  1 john  staff  46969736 Jul  8 04:52 terraform-provider-vsphere_v1.11.0_x4
```

Template planning

terraform plan

- Connects to the hypervisor and performs a dry run to show you what will be created if you apply the template
- Will perform syntax checks and error if you have errors in you code

```
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.vsphere_datacenter.dc: Refreshing state...
data.vsphere_resource_pool.pool: Refreshing state...
data.vsphere_network.network: Refreshing state...
data.vsphere_datastore_cluster.datastore_cluster: Refreshing state...
data.vsphere_virtual_machine.template: Refreshing state...

------------------------------------------------------------------------

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

Terraform will perform the following actions:

  # tls_private_key.ssh will be created
  + resource "tls_private_key" "ssh" {
      + algorithm                   = "RSA"
      + ecdsa_curve                 = "P224"
      + id                          = (known after apply)
      + private_key_pem             = (known after apply)
      + public_key_fingerprint_md5  = (known after apply)
      + public_key_openssh          = (known after apply)
      + public_key_pem              = (known after apply)
      + rsa_bits                    = 2048
    }

  # vsphere_folder.icpenv[0] will be created
  + resource "vsphere_folder" "icpenv" {
      + datacenter_id = "datacenter-21"
      + id            = (known after apply)
      + path          = "Target/Team01/Lab2"
      + type          = "vm"
    }
```

Applying your template

terraform apply

- Will first connect to the hypervisor and perform a plan

- If approved it will apply the changes and create the resources defined

```
vsphere_virtual_machine.camlab[0]: Destroying... [id=42242bf6-329b-0c9d-234a-6ee84276353b]
vsphere_virtual_machine.camlab[0]: Still destroying... [id=42242bf6-329b-0c9d-234a-6ee84276353b, 10s elapsed]
vsphere_virtual_machine.camlab[0]: Destruction complete after 19s
vsphere_virtual_machine.camlab[0]: Creating...
vsphere_virtual_machine.camlab[0]: Still creating... [10s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [20s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [30s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [40s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [50s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [1m0s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [1m10s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [1m20s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [1m30s elapsed]
vsphere_virtual_machine.camlab[0]: Still creating... [1m40s elapsed]
vsphere_virtual_machine.camlab[0]: Creation complete after 1m50s [id=4224ee76-e1fd-25bd-fa82-5a39321be001]

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

Terraform state files

tfstate

- Tracks the changes that have been made by terraform
- Used to define the declared state and will be compared to actual provider resource to determine configuration change
- Format in JSON


```
{
  "version": 4,
  "terraform_version": "0.12.0",
  "serial": 6,
  "lineage": "043e9eea-b239-fb5f-c1d7-6def604feefa",
  "outputs": {},
  "resources": [
    {
      "mode": "data",
      "type": "vsphere_datacenter",
      "name": "dc",
      "provider": "provider.vsphere",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "id": "datacenter-21",
            "name": "Datacenter"
          }
        }
      ]
    },
    {
      "mode": "data",
      "type": "vsphere_datastore_cluster",
      "name": "datastore_cluster",
      "provider": "provider.vsphere",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "datacenter_id": "datacenter-21",
            "id": "group-p163",
            "name": "DatastoreCluster"
          },
          "depends_on": [
            "data.vsphere_datacenter.dc"
          ]
        }
      ]
```

Template Deletion

terraform destroy

- Will first connect to the hypervisor compare what is there to the .tfstate file
- It will show you want is going to be deleted
- If approved it will destroy all the resources

```
# vsphere_folder.icpenv[0] will be destroyed
- resource "vsphere_folder" "icpenv" {
    - datacenter_id = "datacenter-21" -> null
    - id            = "group-v466" -> null
    - path          = "Target/Team10/Lab2" -> null
    - type          = "vm" -> null
  }

# vsphere_virtual_machine.camlab[0] will be destroyed
- resource "vsphere_virtual_machine" "camlab" {
    - boot_delay                          = 0 -> null
    - boot_retry_delay                    = 10000 -> null
    - boot_retry_enabled                  = false -> null
    - change_version                      = "2019-07-08T01:41:36.94448Z" -> null
    - cpu_hot_add_enabled                 = false -> null
    - cpu_hot_remove_enabled              = false -> null
    - cpu_limit                           = -1 -> null
    - cpu_performance_counters_enabled    = false -> null
    - cpu_reservation                     = 0 -> null
    - cpu_share_count                     = 2000 -> null
    - cpu_share_level                     = "normal" -> null
    - datastore_cluster_id                = "group-p163" -> null
    - datastore_id                        = "datastore-107" -> null
    - default_ip_address                  = "10.0.0.190" -> null
    - efi_secure_boot_enabled             = false -> null
    - enable_disk_uuid                    = false -> null
    - enable_logging                      = false -> null
    - ept_rvi_mode                        = "automatic" -> null
    - firmware                            = "bios" -> null
    - folder                              = "Target/Team10/Lab2" -> null
    - force_power_off                     = true -> null
    - guest_id                            = "rhel7_64Guest" -> null
    - guest_ip_addresses                  = [
        - "10.0.0.190",
        - "fe80::250:56ff:fea4:4792",
      ] -> null
    - host_system_id                      = "host-44" -> null
    - hv_mode                             = "hvAuto" -> null
    - id                                  = "4224ee76-e1fd-25bd-fa82-5a39321be001" -> null
```

Putting it all together