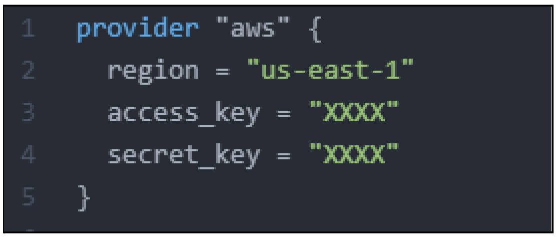
HashiCorp Certified Terraform Associate

Infrastructure as Code (IaC)

- Benefits:
  - Automation.
  - Versioning.
  - Reusability.
- Additional features:
  - Code can easily be shared.
  - Creates a blueprint of your data center.
- IaC makes changes:
  - Idempotent # *the result remains the same*
  - Consistent.
  - Repeatable.
  - Predictable.
- IaC makes infrastructure more reliable:
  - With IaC, we can test the code and review the results before the code is applied to our target environments.
  - IaC makes changes idempotent, consistent, repeatable and predictable.
  - Without IaC, an operator might open multiple sessions during infrastructure changes, which often results in skipped steps that can result in slight differences between servers that compound over time and could impact their performance, usability or security.
  - Since code is checked into version control systems such as GitHub, GitLab, BitBucket, etc., it is possible to review how the infrastructure evolves over time.

Providers

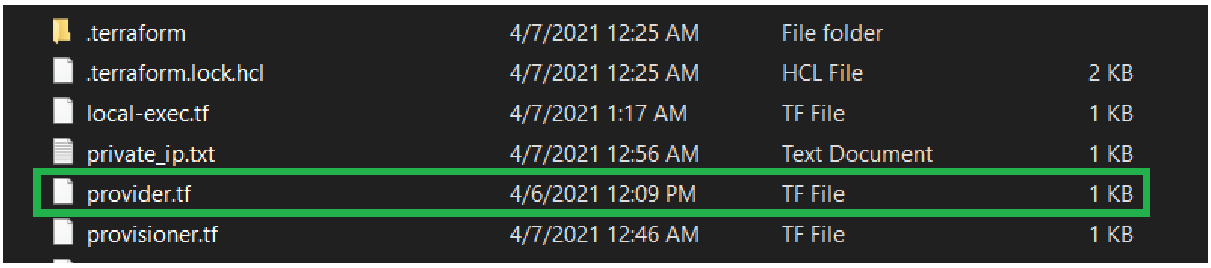
- A **provider** is responsible for understanding **API interactions (API token required).**



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

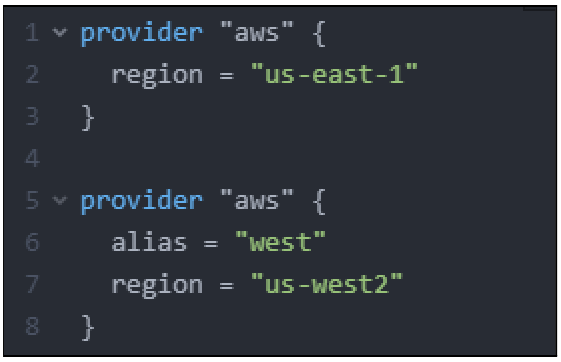
- You **can explicitly set** a specific **version of the provider** within the provider block (**required_providers**).
- It can have multiple providers in the same block.
- Providers are released on a separate schedule from Terraform itself.
- To upgrade to the latest acceptable version of each provider, run **terraform init -upgrade**

- **Additional provider configurations** (those with the **alias** argument) **are never inherited automatically by child modules**, and so must always be passed explicitly using the providers map.
- You can have a TF file only for provider configuration instead of copy and paste this configuration in all TF files.



```
.terraform            4/7/2021 12:25 AM    File folder
.terraform.lock.hcl   4/7/2021 12:25 AM    HCL File          2 KB
local-exec.tf         4/7/2021 1:17 AM     TF File           1 KB
private_ip.txt        4/7/2021 12:56 AM    Text Document     1 KB
provider.tf           4/6/2021 12:09 PM    TF File           1 KB
provisioner.tf        4/7/2021 12:46 AM    TF File           1 KB
```

- **To create multiple configurations for a given provider**, **include multiple provider blocks with the same provider name**. For each additional non-default configuration, **use the alias meta-argument to provide an extra name segment (provider = aws.west).**
- Location of the user plugins directory
  - **Windows: %APPDATA%\terraform.d\plugins**
  - **Others: ~/.terraform.d/plugins**
- You can have **multiple provider** instance with the help of **alias**



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

provider "aws" {
  alias = "west"
  region = "us-west2"
}
```

- **Statements**:
  - A child module automatically inherits default (un-aliased) provider configurations from its parent.
  - Each module must declare its own provider requirements so that Terraform can ensure that there is a single version of the provider that is compatible with all modules in the configuration.
  - Provider configurations can be defined only in a root Terraform module.

Provider Dependencies

- Resource belonging to a particular provider removed from configuration but exists in the current state.
- Explicit use of a provider block in configuration, optionally including a version constraint.
- Use of any resource belonging to a particular provider in a resource or data block in the configuration.

CLI

Terraform Init

- The **terraform init** command is used to **initialize a working directory** containing Terraform configuration files.
- **Initialization downloads and installs the provider's plugin** so that it can later be executed.
- **Downloads the declared providers** which are supported by HashiCorp.
- **Initializes the backend configuration**.
- It will **NOT create** any **sample files** like example.tf
- After terraform init is executed, **plugins download to the .terraform/plugins directory** in the directory terraform init was executed in.
- The **terraform init -upgrade** option **updates all previously installed plugins** to the newest version that complies with the configuration's version constraints.
- **It cannot install third-party plugins.**

Terraform 0.12

- Searches the configuration for direct and indirect references to providers and load plugins distributed by HashiCorp.
- Searches for module blocks and the source code for referenced modules is retrieved from the locations given in their source arguments.
- Sees the root configuration directory for backend configuration and initializes the chosen backend.

Terraform Plan

- The **terraform plan** command is used to **create an execution plan**.
- It will **NOT modify** things in **infrastructure**.
- Terraform performs a refresh, unless explicitly disabled and then determines what actions are necessary to achieve the desired state specified in the configuration files.
- The **terraform plan** command **retrieves the data required by providers**.

Terraform Apply

- The **terraform apply** command is used to apply the changes required to reach the desired state of the configuration.
- The **terraform apply** will also **write** data to the **terraform.tfstate** file
- The **terraform apply can change, destroy and provision resources** but **cannot import any resource**.
- The **terraform apply -auto-approve** (won't ask you a confirmation)
- When you execute **terraform apply**
  - **Terraform makes infrastructure changes defined in your configuration**
  - **Terraform updates the state file with configuration changes made during the execution**

- Even if the user has just changed the resource name, Terraform will delete the existing resource and create a new (**Terraform tracks resources by their name. If you change the name, you have created a new resource and deleted the old resource**)

Terraform Refresh

- The **terraform refresh syncs the state file** with the **real world infrastructure**
- The **terraform refresh** command is used to **reconcile the state Terraform** knows about (via its state file) with the real-world infrastructure.
- This does **NOT modify infrastructure** but **modifies the state file**
- The following commands will **implicitly run terraform refresh**:
  - **terraform plan**
  - **terraform destroy**
  - **terraform import**
- The following commands **run refresh first, prior to any other work unless explicitly disabled:**
  - **terraform apply**
  - **terraform plan**
  - **terraform destroy**

Terraform Destroy

- The **terraform destroy** command is used to **destroy the Terraform-managed infrastructure.**
- It is **NOT** the **only command** through **which infrastructure can be destroyed**.

Terraform Output

- The **terraform output** command is used to extract the value of an output variable from the state file
  - **terraform output instance_ips**

Terraform Format

- The **terraform fmt** command is used to **rewrite** Terraform configuration files to **canonical format and style (terraform fmt -diff #differences)**
- For use-case, where all the configuration written by team members needs to have a proper style of code, **terraform fmt** can be used.

Terraform Validate

- The **terraform validate** command validates the configuration files in a directory.
- **Verifies** whether a **configuration is syntactically valid** and thus primarily useful for general verification of reusable modules, including the correctness of attribute names and value types.
- Validation **requires an initialized** working directory with any referenced plugins and modules installed.

Terraform Import

- Terraform is able to **import existing infrastructure**. This **allows** you to **take resources that you've created** by some other means and bring it under Terraform management.
- The current implementation of Terraform import can only import resources into the state. It does **NOT generate configuration.**
- You **have to write a resource configuration block manually for the resource**, to which the imported object will be mapped.
- The terraform import command can import resources into modules as well as directly into the root of your state.
- NOT all providers and resources support Terraform import.
- **terraform import aws_instance.myEC2 <instance-id>**

Terraform Lock

- **If supported by your backend**, **Terraform will lock your state for all operations** that could write state.
- **Local state files cannot be unlocked by another process**.
- NOT all backends support locking functionality.
- If your backend supports state locking, it will automatically set up.
- Backends supported: S3, AzureRM, Consul

Terraform Unlock

- Terraform has a force-unlock command to manually unlock the state if unlocking fails.
- **terraform force-unlock <LOCK_ID [DIR]>**

Terraform Taint

- The **terraform taint** command manually marks a Terraform-managed **resource as tainted**, forcing it to be **destroyed and recreated** on the **next apply**.
- Once a resource is marked as tainted, the next plan will show that the resource will be destroyed, recreated and the next apply will implement this change.
- The **terraform taint** can also be used to taint resources with a module.
- **terraform taint aws_security_group.allow_all**

- Taint the resource "**aws_instance**" "**myEC2**" resource that lives in **module xyz** which lives in **module abc**.
  - **module abc > module xyz > aws_instance > myEC2**
  - **module.abc.module.xyz.aws_instance.myEC2**

Terraform Plan Destroy

- The behavior of any terraform destroy command can be previewed at any time with an equivalent **terraform plan -destroy** command

Terraform State

- The **terraform state** commands can be used to modify the state directly.
- **Remote state allows teams to share infrastructure resources in a read-only** way without relying on any additional configuration store.
- **Terraform forces every state modification command to write a backup file** (the path of these backup file can be controlled with **-backup**)
- Features:
  - **Mapping configuration to real-world resources.**
  - **Determining the correct order to destroy resources.**
  - **Increased performance.**

Command	Description
terraform state list	Lists the resources.
terraform state mv	Moves items. Also moves resources to an entirely new state
terraform state pull	Manually downloads and outputs the state.
terraform state push	Manually upload a local state file to a remote state.
terraform state rm	Removes items from the state
terraform state show	Shows the attributes of a single resource in the state (inspects the current state of the infrastructure applied).

Terraform Graph

- The **terraform graph** command is used to generate a visual representation of either a configuration or execution plan
- The output of the **terraform graph** is in the **DOT format**, which can easily be converted to an image.

Terraform Login

- The **terraform login** command can be used to automatically **obtain and save an API token for Terraform Cloud, Terraform Enterprise** or any **other host** that **offers Terraform services**.

Terraform Show

- The **terraform show** command is used to provide **human-readable output form a state or plan file**. This can be used to inspect a plan to ensure that the planned operations are expected, or to inspect the current state as Terraform sees it.
- You may use **show with a path** to either a Terraform **state file or plan file**. If you **don't specify a file path**, Terraform **will show the latest state snapshot.**
- The terraform show command can be used to inspect a plan to ensure that the planned operations are expected.

Terraform Provider

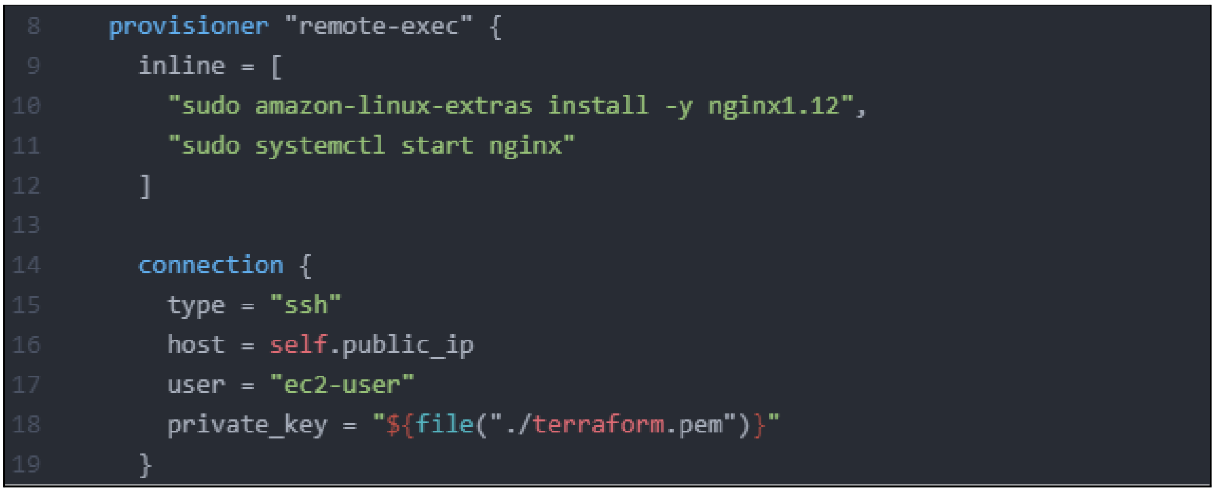
- The **terraform providers** command **prints information about the providers used in the current configuration.**
- This command gives an overview of all of the current dependencies, as an aid to understanding why a particular provider is needed.

Terraform Provisioner

- Provisioners can be used to model specific **actions** on the **local machine** or on **remote machines** in order to prepare servers or other infrastructure objects for service.
- Provisioners should only be used as a last resort. For most common situations, **there are better alternatives (minimally).**
- Provisioners are inside in the resource block
- By default, a provisioner defined in a resource is a **creation-time provisioner (except when you set "when = destroy")**
- You can use a provisioner **without** a resource attached **using "null_resource", which acts like a regular Terraform resource**, except that it doesn't create anything.

remote-exec

- The **remote-exec provisioner invokes a script on a remote resource** after it is created.
- Supports both **SSH amd WINRM type connections**.

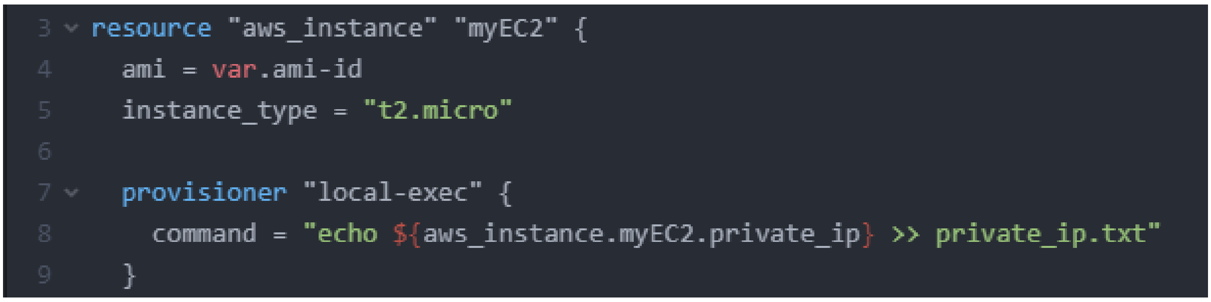


```
provisioner "remote-exec" {
  inline = [
    "sudo amazon-linux-extras install -y nginx1.12",
    "sudo systemctl start nginx"
  ]

  connection {
    type = "ssh"
    host = self.public_ip
    user = "ec2-user"
    private_key = "${file("./terraform.pem")}"
  }
```

local-exec

- The **local-exec provisioner invokes a local executable** after a resource is created**.** This invokes a **process on the machine running Terraform**, **NOT** on the **resource**



```
resource "aws_instance" "myEC2" {
  ami = var.ami-id
  instance_type = "t2.micro"

  provisioner "local-exec" {
    command = "echo ${aws_instance.myEC2.private_ip} >> private_ip.txt"
  }
```

Behaviour

- By default, provisioners that fail will also cause the terraform apply itself to fail.
- The **on_failure** setting can be used to change this.
    - **on_failure = continue**
        - **Ignore the error and continue with creation or destruction.**
    - **on_failure = fail**
        - **Raise an error and stop applying** (**default**). If this is a creation provisioner, **taint the resource**.

Types

- **Creation-Time Provisioner**
    - **Creation-Time** provisioners are only run **during creation**, NOT during updating or any other lifecycle.
    - **Hence no matter how many times you run creation-time** (default) provisioners after creating a resource it will not run.
    - **If a Creation-Time provisioner fails**, the resource **is marked as tainted**.
- **Destroy-Time Provisioner**
    - **Destroy-Time** provisioners are run **before the resource is destroyed**.

CONFIGURATION

Terraform Version

- The **required_version** specifies which **versions of Terraform** can be used with your configuration.
- **If the running version of Terraform doesn't match the constraint** specified, Terraform **will produce an error** and exit without taking any further actions.

Debugging

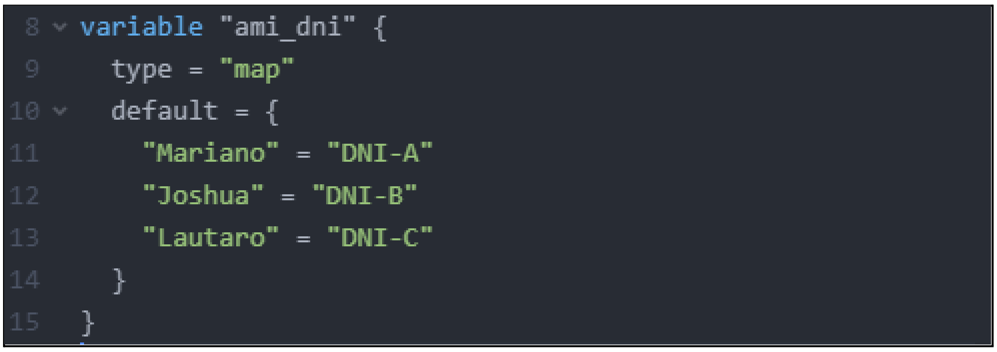
- Terraform has detailed logs that can be enabled by setting the **TF_LOG** environment variable to any value.
- **TF_LOG levels**:
    - TRACE # *Highest verbosity & RANDOM*
    - DEBUG
    - INFO
    - WARN
    - ERROR
- Example: **TF_LOG=TRACE**
- Example**: export TF_LOG= or unset TF_LOG (will disable detailed logs)**
- To persist logged output, you can set **TF_LOG_PATH.** Then you have to **set** the **TF_LOG for the logging (second - stderr).**
- The value of **TF_CLI_ARFS** will specify additional arguments to the command-line (**export TF_CLI_ARGS_plan=“-no-color”**).

Data Types

Type	Description
string	Sequence of unicode chars representing some text
list	Sequential list of values identified by **their position; starts with 0** **[“argentina”, “usa”, “canada”]**
map	A group of values identified by **name labels {name = “Mariano”, age = 32}**
number	200

Map

- To reference to DNI-A from the below map, following approaches need to be used:
    - var.ami_dni[“Mariano”]



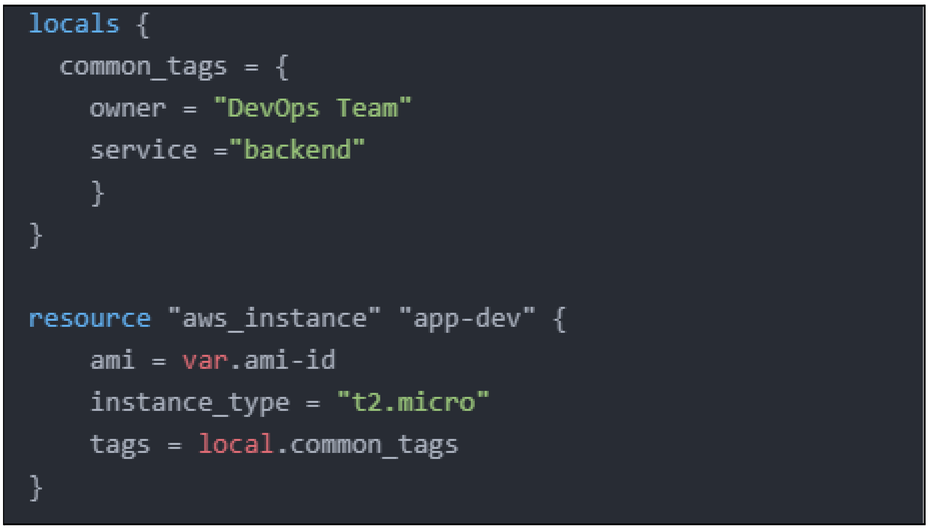
```
variable "ami_dni" {
  type = "map"
  default = {
    "Mariano" = "DNI-A"
    "Joshua" = "DNI-B"
    "Lautaro" = "DNI-C"
  }
}
```

Structural Data Types

Type	Description
object	A collection of named attributes that each have their own type. **object({ name=string, age=number})**
tuple	**tuple([<TYPE>, ...])**

Local Values

- A **local value** assigns a name to an expression, allowing it to be used **multiple times**.
- A **local value can reference to another local value**.
- Local **cannot refer to itself** or to a **variable that refers back to it** (directly or indirectly)



```
locals {
  common_tags = {
    owner = "DevOps Team"
    service ="backend"
    }
}

resource "aws_instance" "app-dev" {
    ami = var.ami-id
    instance_type = "t2.micro"
    tags = local.common_tags
}
```

Sentinel

- Sentinel is an embedded policy-as-code framework.
- It is a **proactive service**.
- The **Sentinel CLI allows for the developing and testing of policies outside of a particular Sentinel implementation.**
- It can be used for:
  - Verify if an EC2 instance has tags.
  - Verify if the S3 bucket has encryption enabled.
- Workflow: **terraform plan > sentinel checks > terraform apply**
- It is **NOT available in the open-source version**

Sensitive Output

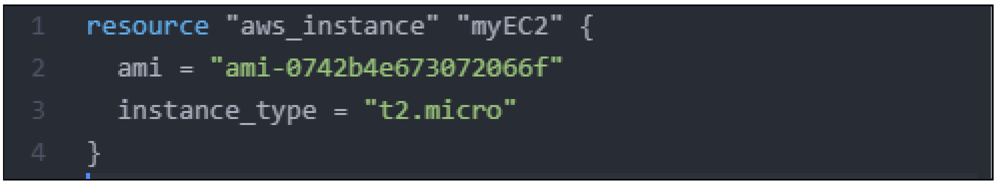
- An output can be marked as containing sensitive material using the optional sensitive argument:
  - **sensitive = true**
- Setting an output value in the root module as sensitive prevents Terraform from showing its value in the list of outputs at the end of terraform apply
- **Sensitive output values are still recorded in the state**, and so will be visible to anyone who is able to access the state data.

Vault

- Secrets are persisted to the state file and plans
- When you use Vault, the data block would look something like this:
  - **vault_aws_secret_backend.aws.path**

Resource Block

- A resource block declares a resource of a given **type** (**"aws_instance"**) with a given **local name** (**"myEC2"**)



```
resource "aws_instance" "myEC2" {
  ami = "ami-0742b4e673072066f"
  instance_type = "t2.micro"
}
```

Sensitive Data in State File

- **Terraform Cloud always encrypts the state at rest and protects it with TLS in transit**.
- The **S3 backend supports encryption** at rest **when the encrypt option is enabled**.
- **Using environment variables** won't result in sensitive information being written to the state file

Terraform Limitation & Notes

- Terraform allows us to limit the number of concurrent operations. By default, **Terraform allows 10 concurrent operations**.
- State is a requirement for Terraform to function.
- Terraform recommends **2 spaces for each nesting level**.
- HashiCorp **style** conventions suggest you that **align the equal signs**
- **Terraform workflow steps:**
  - **Write** *#Author infrastructure as code.*
  - **Plan** *#Preview changes before applying.*
  - **Apply** *#Provision reproducible infrastructure.*
- **CLI configuration** files in Terraform:
  - **.terraformrc**
  - **terraform.rc**

- Terraform is an **immutable**, **declarative IaC provisioning language** based on HashiCorp Configuration Language (**HCL**) or **optionally JSON**
- Terraform **has no mechanism to redact or protect secrets** that are returned via data sources.
- Terraform is **cloud-agostic** (single conf can be used to manage multiple providers)
- Terraform **advantages**:
  - Platform Agnostic.
  - State Management.
  - Operator Confidence.
- **Data source:**
  - **Enables Terraform to fetch data** for use elsewhere in the Terraform configuration.
  - **Allow data to be fetched or computed for use elsewhere in Terraform configuration**. Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.
- OS that Terraform is available:
  - Linux
  - macOS
  - Solaris
  - FreeBSD
  - Windows
- Some **problems** of how **infrastructure** was **traditionally** managed before IaC:
  - Traditionally managed infrastructure **can't keep up with cyclic or elastic applications.**
  - Traditional deployment methods **are NOT able to meet the demands of the modern business where resources tend to live days to weeks**, rather than months to years.
  - **Requests for infrastructure or hardware required a ticket**, increasing the time required to deploy applications.
- If **Terraform crashes** you see the logs in **"crash.log"**
- **Version constraints are supported** only for modules installed from a module registry (Terraform Registry or Terraform Cloud's private module registry).
- When using a **remote state**, the state is only ever held in **memory when used by terraform.**
- Terraform can manage low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.
- **Performance** (plan/apply):
  - Many cloud **providers do NOT provide APIs to query multiple resources at once**, and the round trip time for each resource is hundreds of milliseconds; hence for a large infrastructure querying every resource causes slowness during the plan and apply command.
  - By default, for every plan and apply, **Terraform will sync all resources in your state with the real world to know the current state** of resources to effectively determine the changes that it needs to make to reach your desired configuration. For **large infrastructures**, this behavior of querying every resource makes plan and apply **commands execution very slow.**

    - **Cloud providers have API rate-limiting**, and for a large infrastructure, Terraform has to make a lot of calls to querying every resource; hence API rate-limitation stops Terraform from making many concurrent API calls at once that make a plan and apply command slow.

Disaster Recovery

- Terraform allows cheaper and much more dynamic disaster recovery plans because you can literally build a DR environment on the fly; hence it may not make sense to keep it running all the time and pay a massive bill for it.
- Terraform can be an essential part of your disaster recovery strategy because it helps you stand up new infrastructure very quickly and efficiently.

Credentials in Config

- You can **store** the credentials **outside of terraform configuration**.
- **Storing credentials as part of environment variables** is also a much **better** approach **than hard coding** it in the system

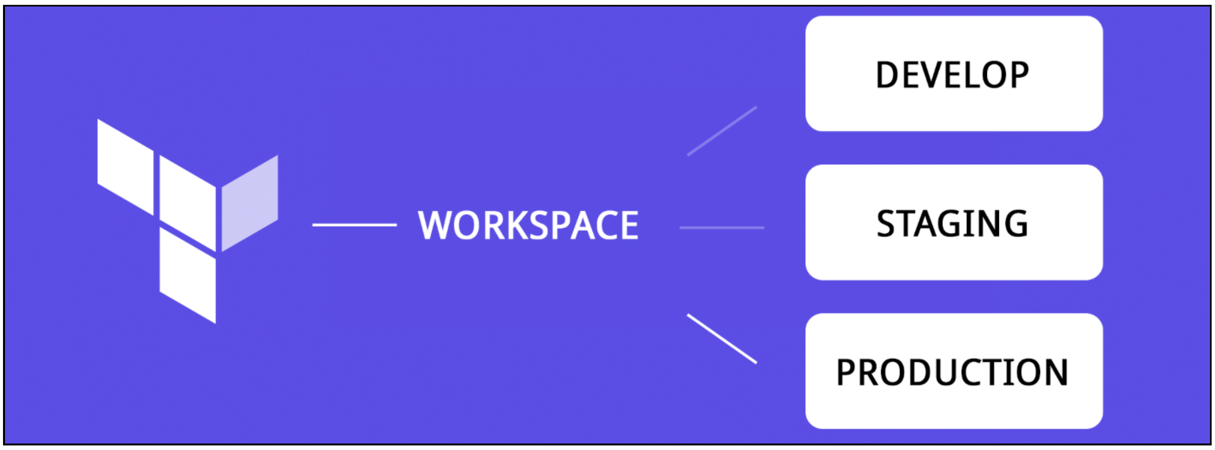
Workspaces

- Terraform allows us to have **multiple workspaces**; with each of the workspaces, we can have a **different set of environment variables associated**.
- It allows **multiple state files of a single configuration (terraform.tfstate.d).**
- Workspaces are managed with the **terraform workspace** set of commands.
- **NOT suitable for isolation or strong separation** between workspace (stage/prod/develop).
- Terraform workspaces are technically equivalent to renaming your state file.
- **CLI workspaces are alternative state files in the same working directory**
- Terraform Cloud manages infrastructure collection with a workspace whereas CLI manages collections of infrastructure resources with a persistent working directory.
- Terraform Cloud maintains the state version and runs history for each workspace. Also, **workspaces act more like completely separate working directories**.

Command	Use Case
terraform workspace new mnademo	**Create** *#by default, when you create a new workspace you are automatically switched to it*
terraform workspace select mnaprod	**Switch**

Use Case

- **PROJECT "A"**
  - STAGING - instance_type = "t2.micro"
  - PRODUCTION - instance_type = "m4.large"



Dependencies

Implicit

- With implicit dependency, **Terraform can automatically find references of the object** and create an implicit ordering requirement between the two resources.
- Terraform can infer when one resource depends on another by analyzing **the resources attributes used in interpolation expressions.** From this, Terraform builds a dependency tree to determine the correct order to create each resource.
- The **order of blocks and expression** in the Terraform configuration file **doesn't matter.**

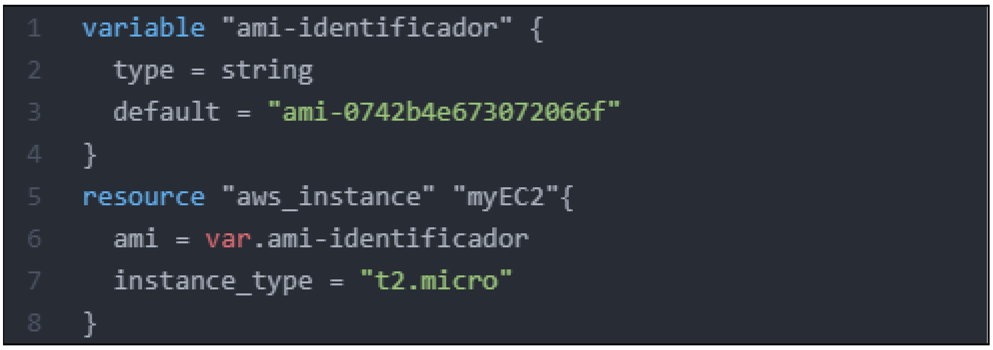
Explicit

- Explicitly specifying a dependency **is only necessary when a resource relies on some other resouce's behavior.**
- **depends_on = [aws_s3_bucket.example]**

Variables

- **If you have variables with undefined values**, **Terraform will ask you** to supply the value associated with them.
- Input variables serve as parameters for a Terraform module, **allowing aspects of the module to be customized without altering the module's own source code** and **allowing modules to be shared** between different configurations (**dynamic and/or reusable, static values should be converted in input variables**)
- It will **NOT ask you** if you **have a terraform.tfvars**
  - Following is syntax to load **custom tfvars file**:
    - **terraform apply -var-file="custom.tfvars"**
- 1 - Example: **variable ami {}** *#Terraform will ask you*

- 2 - Example:



```
variable "ami-identificador" {
  type = string
  default = "ami-0742b4e673072066f"
}
resource "aws_instance" "myEC2"{
  ami = var.ami-identificador
  instance_type = "t2.micro"
}
```

Environment Variables

- It can be used to set variables.
- Detailed logs in terraform can be enabled by setting the **TF_LOG** environment variable.
- The environment variables must be in the format **TF_VAR_<name>**
- Example: **export TF_VAR_region=us-east-1**

Precedence

1. **Environment variables.**
2. The **terraform.tfvars** file (if present).
3. The **terraform.tfvars.json** file (if present).
4. Any ***.auto.tfvars or *.auto.tfvars.json** files, processed in lexical order of their file names.
5. Any **-var and -var-file** options on the command line in the order they are provided.

- If the same variable is assigned **multiple values**, **Terraform uses the last value it finds**.

Terraform Registry

- The **Terraform Registry is integrated directly into Terraform**.
- The registry uses tags to identify module versions. Release tag names must be a semantic version, which can optionally be prefixed with a v. 1.0.4 and 0.9.2.
- **The module must be on Github** and must be a **public repo.**
- **Published modules** via Git and GitHub **provides:**
  - **Allow browsing version histories.**
  - **Automatically generated documentation.**
  - **Show examples and READMEs.**
  - **Support versioning.**

Public Registry

- Requirements for publishing a module:
  - The module must be on a public GitHub repo with at least one release tag in **x.y.z format** (version).
  - Module GitHub repositories must use this three-part name format **terraform-<PROVIDER>-<NAME>**

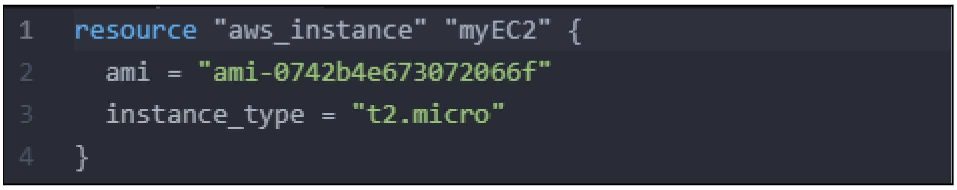
        - Example: **terraform-aws-ec2-instance**
- **Syntax** for referencing a **public registry module** is:
    - **<NAMESPACE>/<NAME>/<PROVIDER>**
        - Example: **hashicorp/consul/aws**

Private Registry

- You can also use modules from a private registry, like the one provided by Terraform Cloud.
- **Syntax** for referencing a **private registry**:
    - **<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>**
        - Example: **app.terraform.io/example_corp/vpc/aws**

Terminologies

- Resource Type: **aws_instance**
- Local name: **myEC2**
- <u>Arguments</u>
    - 1 - Argument Name: **ami**
    - 1 - Argument Value: **ami-0742b4e673072066f**
    - 2 - Argument Name: **instance_type**
    - 2 - Argument Value: **t2.micro**



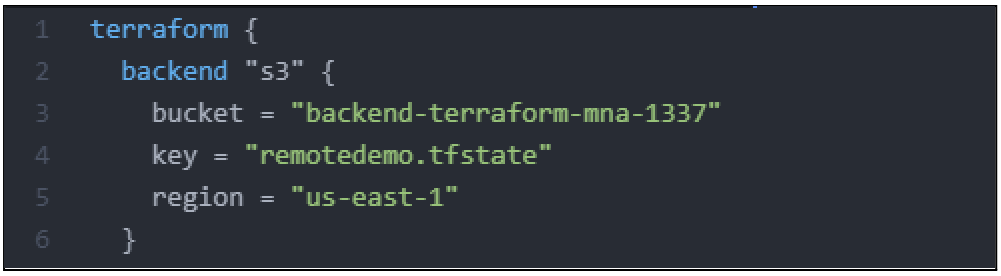
```
resource "aws_instance" "myEC2" {
  ami = "ami-0742b4e673072066f"
  instance_type = "t2.micro"
}
```

GIT

- If you are making use of GIT repository for commiting terraform code, the **.gitignore** should be configured to ignore certain terraform files that might contain sensitive data.
    - **terraform.tfstate file** (can include sensitive information)
    - ***.tfvars** (may contain sensitive data like passwords)
    - any files with names ending in **.auto.tfvars or .auto.tfvars.json**
- Example: **source = “git::ssh://username@example.com/instances.git”**
- Example: **source = “git::https://username@example.com/instances.git”**
- Arbitrary GIT repos can be used by prefixing the address with the special git:: prefix
    - **source = “git::https://example.com/vpc.git”**
- By default, Terraform will clone and use the default branch (referenced by HEAD) in the selected repository. You can override this using the ref argument:
    - **source = “git::https://example.com/vpc.git?ref=v1.2.0”**

Backend

- Backends are configured directly in Terraform files in the terraform section.
- **After configuring a backend**, **it has to be initialized**.



```
terraform {
  backend "s3" {
    bucket = "backend-terraform-mna-1337"
    key = "remotedemo.tfstate"
    region = "us-east-1"
  }
```

- By **default**, Terraform uses the **"local" backend**, which is the normal behavior.
- **Github** is **NOT supported** as a **backend type**.
- The **default backend** for **Terraform OSS** is **local.**
- Ways the **configuration** can be added to Terraform so it can **initialize and communicate with the backend**:
  - **Interactively on the command line**
  - **Command-line key/value pairs.**
  - **Use the -backend-config=PATH to specify a separate config file.**

Partial Configuration

- When some or all of the arguments are omitted, we call this a partial configuration

Changing Configuration

- You can change your backend configuration at any time

Enhanced Backends

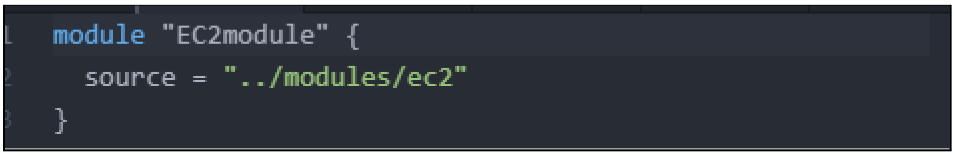
- Local
  - It **stores state on the local filesystem**, locks that state using system APIs and performs operations locally.
- Remote
  - It stores state files and may be used to run operations in **Terraform Cloud.**
  - When using full remote operations, operations like terraform plan or terraform apply can be executed in Terraform Cloud's environment, with log output streaming to the local terminal. Remote plans and applies use variable values from the associated Terraform Cloud workspace.
  - If you **delete remote backend configuration** from Terraform code and run terraform apply, it will detect backend change and **throw an error**.

Remote Backend for Terraform Cloud

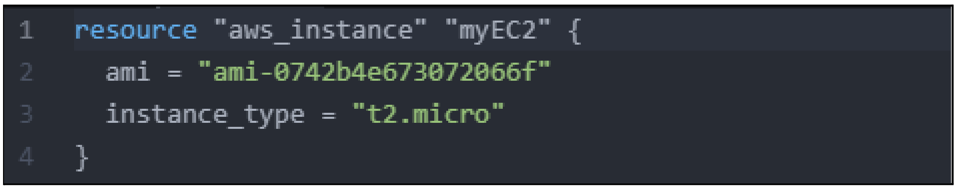
- The remote backend stores Terraform state and may be used to run operations in Terraform Cloud

Modules

- **We can centralize the terraform resources and can call out from TF files** whenever required.
- **A module can call other modules**, which lets you include the child module's resources into the configuration in a concise way.
- The **terraform get** command **allows us to download and update** the modules mentioned in the root module.
- Modules sourced from **local files paths** do **NOT support version**.
- A **local path** must begin with either **./ or ../** to indicate that a local path is intended to distinguish from a module registry address.
- Create separate tags and use a stable one for the production and a new one for the staging when you work with versioned modules and you have more than one environment.
- Uses:
  - **Input variables**: to accept values from the calling module.
  - **Output values**: to return to the calling module, which it can then use to populate arguments elsewhere.
  - **Resources**: to define on or more infrastructure objects that the module will manage.



```
module "EC2module" {
  source = "../modules/ec2"
}
```



```
resource "aws_instance" "myEC2" {
  ami = "ami-0742b4e673072066f"
  instance_type = "t2.micro"
}
```

Benefits

- Enables **code reuse**.
- Supports **versioning** to maintain compatibility.
- Supports **modules stored locally or remotely.**

Accessing Output Values

- The **resources defined in a module are encapsulated**, so the **calling module cannot access their attributes directly**
- The **child module can declare output values to selectively export certain values** to be accessed by the calling module

Versions

- **It is recommended to explicitly constraint the acceptable version numbers for each external module** to avoid unexpected or unwanted changes
- **Version constraints are supported only for modules installed from a module registry**, such as the Terraform Registry or Terraform Cloud's private module registry.

Types

The Root Module

- 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.
- You can set variables declared in the root module using CLI options and environment variables. But to set variables declared in child modules, you should pass their values in the module block.

Child Modules

- A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration.
- Child modules can be called multiple times within the same configuration and multiple configurations can use the same child module.

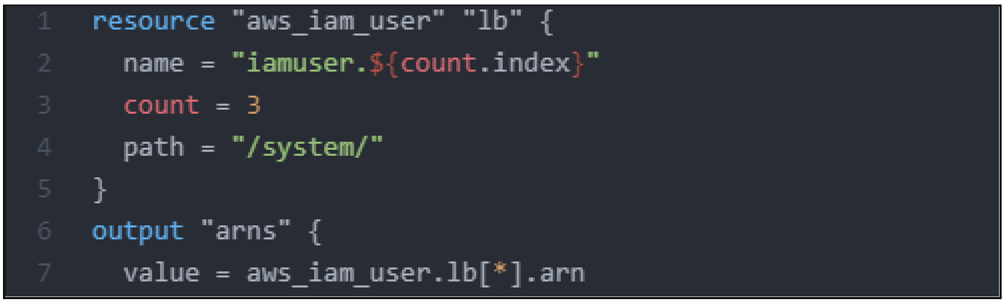
Published Modules

- Terraform can load modules from a public or private registry
- Whenever you add a new module to a configuration you have to install and update modules via:
  - **terraform get**
  - **terraform init**

Expressions

Exp:Splat

- It provides a way to express a common operation that could otherwise be performed with a "for" expression
- Splat Expression allows us to get a list of all attributes



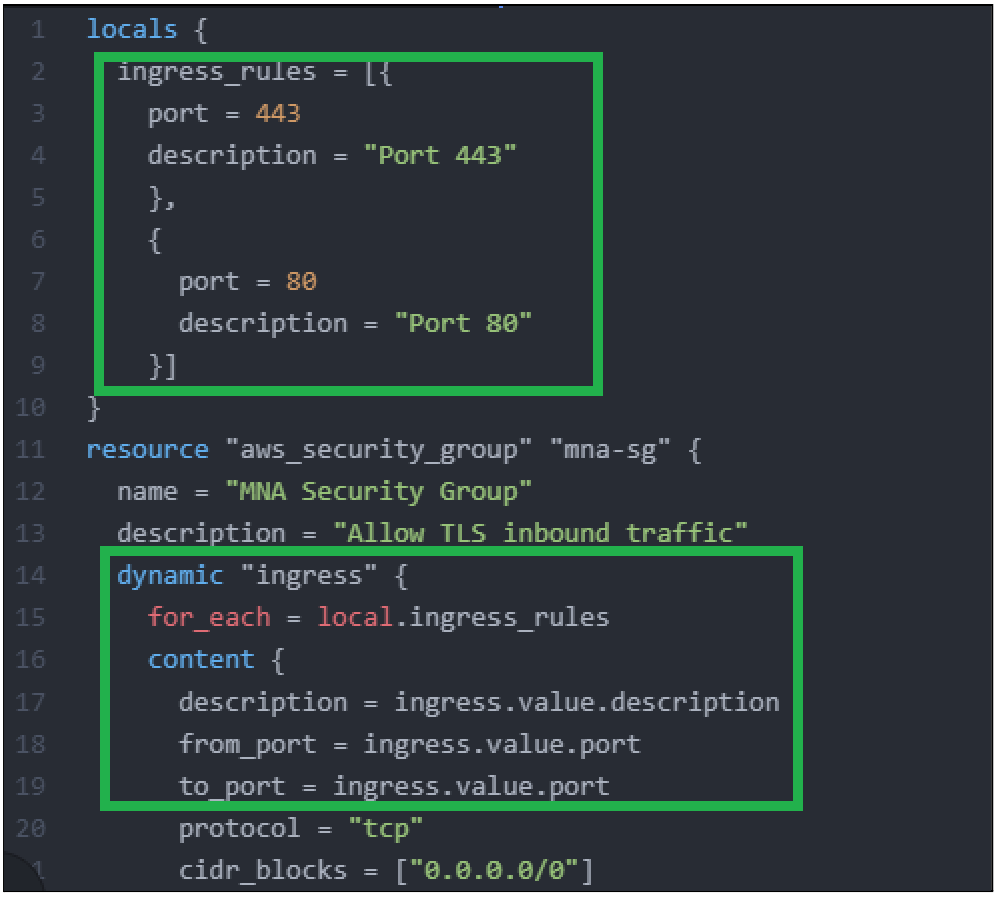
```
resource "aws_iam_user" "lb" {
  name = "iamuser.${count.index}"
  count = 3
  path = "/system/"
}
output "arns" {
  value = aws_iam_user.lb[*].arn
```

Exp:Conditional

- A conditional expression uses the value of a bool expression to select one of two values.
- **condition ? true_val : false_val**

Exp:Dynamic Blocks

- You can dynamically construct repeatable nested blocks using a special dynamic block type (e.g. settings/configs repeatable), which is supported inside resource, data, provider and provisioner blocks.
- The overuse of dynamic blocks can make configuration hard to read and maintain.



```
locals {
  ingress_rules = [{
    port = 443
    description = "Port 443"
    },
    {
      port = 80
      description = "Port 80"
    }]
}
resource "aws_security_group" "mna-sg" {
  name = "MNA Security Group"
  description = "Allow TLS inbound traffic"
  dynamic "ingress" {
    for_each = local.ingress_rules
    content {
      description = ingress.value.description
      from_port = ingress.value.port
      to_port = ingress.value.port
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
```

FUNCTIONS

- The Terraform language does **NOT support user-defined functions**, **only the functions built into** the language are available for use
- It is necessary to have a state file in order to work.

Functions

- **Numeric** (e.g. max, min, log, pow)
- **String** (e.g. chomp, format, replace, title)
- **Collection** (e.g. contains, compact, distinct, index)
- **Encoding** (e.g. urlencode, base64enconde, yamldecode,csvdecode)
- **Filesystem** (e.g. file, basename, drname, templatefile)
- **Date and Time** (e.g. formatdate, timeadd, timestamp)
- **Hash and Crypto** (e.g. md5, sha1, uuid, bcrypt)
- **IP Network** (e.g. cidrhost, cidrnetmask, cidrsubnets, cidrsubnet)
- **Type Conversion** (e.g. can, try, defaults, tolist)

Number:Min

- Right way using the min function:
  - min(12, 54, 3)
  - min([12, 54, 3]...)

Number:Max

- It takes one or more numbers and returns the greatest number from the set.
- **max(12, 54, 3) = 54**

Collection:Slice

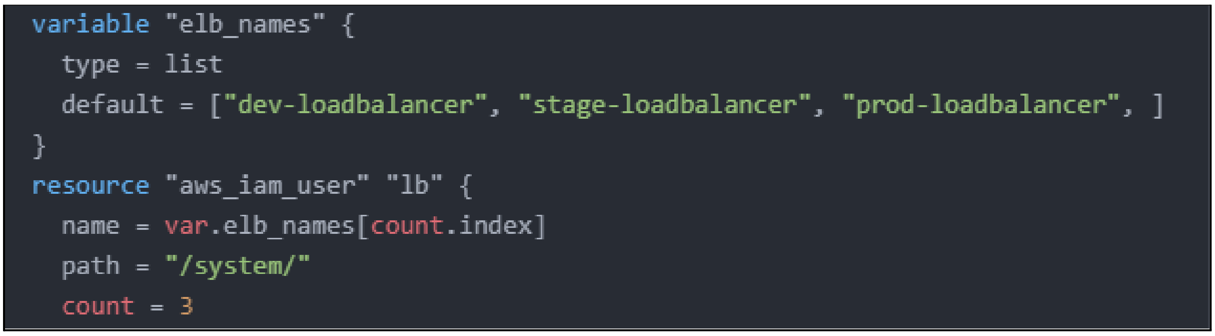
- It extracts some consecutive elements from within a list
- **slice(list, startindex, endindex)**

Collection:Zipmap

- Zipmap constructs a map from a list of keys and a corresponding list of values
- A map is denoted by {}

Count and Count Index

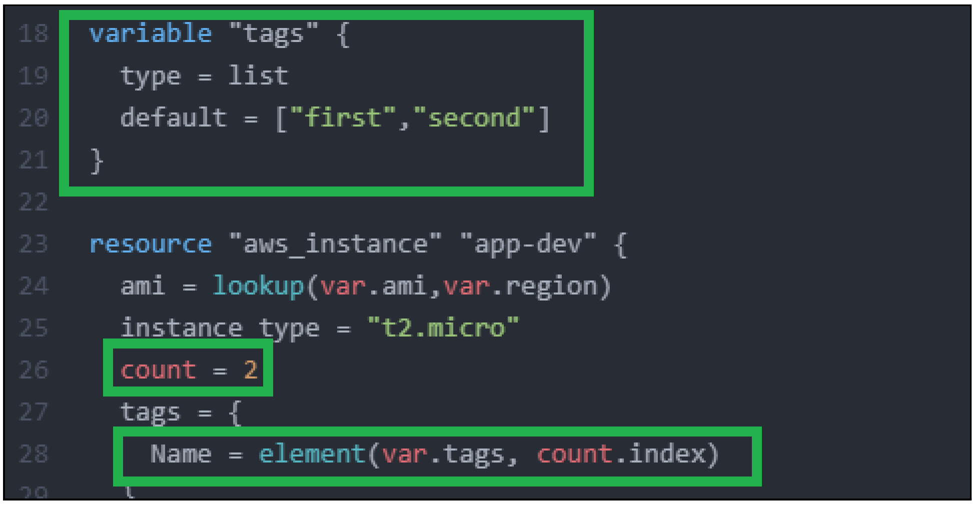
- The count parameter on resources can simplify configurations and let you scale resources by simply incrementing a number
- **Count**: Quantity - “count = 3”
- **Count Index**:Incremental - “count.index” (i++)



```
variable "elb_names" {
  type = list
  default = ["dev-loadbalancer", "stage-loadbalancer", "prod-loadbalancer", ]
}
resource "aws_iam_user" "lb" {
  name = var.elb_names[count.index]
  path = "/system/"
  count = 3
```

Collection:Element

- It retrieves a single element from a list
- If the given index is greater than the length of the list, element retrieves a single element from a list (zero-based index)
- **element(list,index)**
    - Example: element(var.tags, count.index)
        - 1 - element(var.tags, 0) = **0 - string = “first”**
        - 2 - element(var.tags, 1) = **1 - string = “second”**



```
variable "tags" {
  type = list
  default = ["first","second"]
}

resource "aws_instance" "app-dev" {
  ami = lookup(var.ami,var.region)
  instance_type = "t2.micro"
  count = 2
  tags = {
    Name = element(var.tags, count.index)
  }
```

Filesystem:File

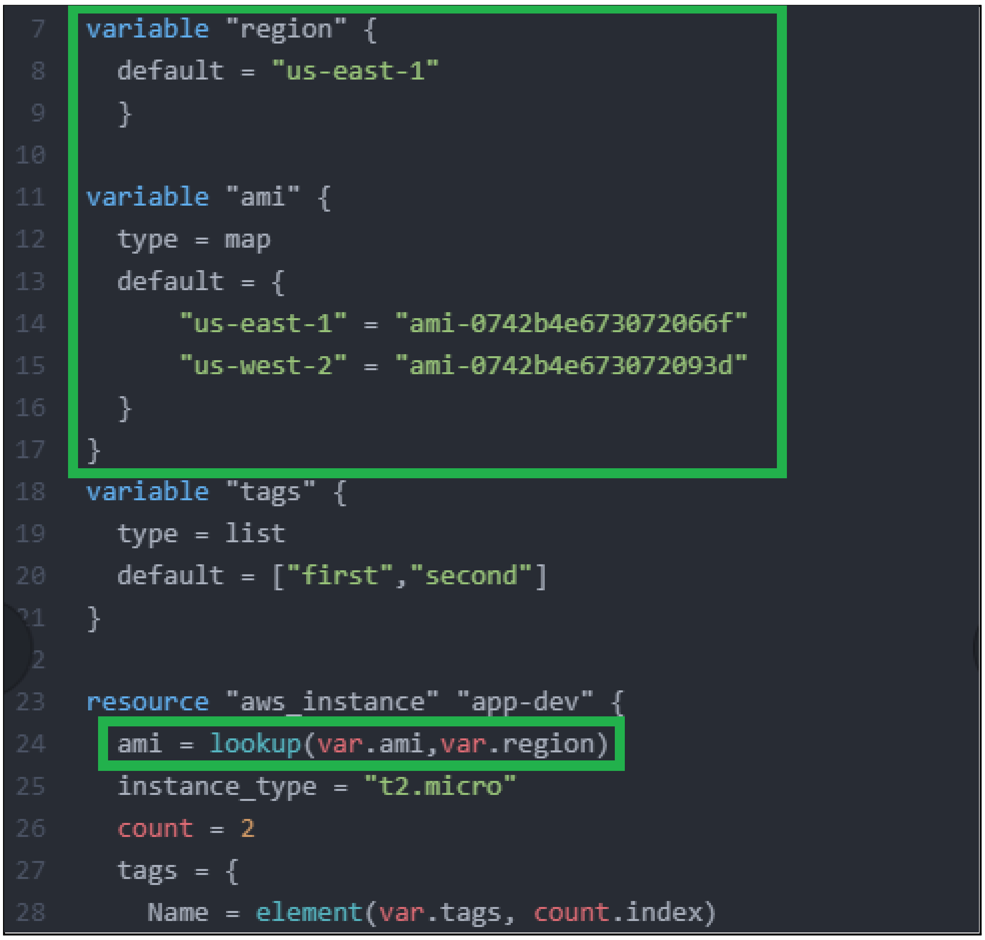
- It reads the contents of a file at the given path and returns as a string
- **file(“${path.module}/hello.txt)**

DAT:Formatdate

- It converts a timestamp into a different time format.
- **formatdate(spect, timestamp)**

Collection:Lookup

- It retrieves the value of a single element from a map, given its key.
- If the given key does NOT exist, the given default value is returned instead.
- **lookup(map, key,** default**)**
    - For historical reasons, the default parameter is actually optional.
    - Example: lookup(var.ami, var.region)
        - var.ami (map) = “us-east-1”
        - var.region (key) = “us-east-1”
            - value = ami-0742b4e673072066f



```
variable "region" {
  default = "us-east-1"
  }

variable "ami" {
  type = map
  default = {
      "us-east-1" = "ami-0742b4e673072066f"
      "us-west-2" = "ami-0742b4e673072093d"
  }
}
variable "tags" {
  type = list
  default = ["first","second"]
}

resource "aws_instance" "app-dev" {
  ami = lookup(var.ami,var.region)
  instance_type = "t2.micro"
  count = 2
  tags = {
    Name = element(var.tags, count.index)
```

DISTRIBUTION

Terraform Enterprise & Terraform Cloud

- **Terraform Enterprise is offered as a private installation**. It is designed to suit the needs of organizations with specific requirements for security, compliance and custom operations. **Terraform Cloud is offered as a multi-tenant SaaS platform and is designed to suit the needs of smaller teams and organizations**. Its smaller plans default to one run at a time, which prevents users from executing multiple runs concurrently.
- **Terraform Enterprise provides several added advantage** compared to **Terraform Cloud**
    - SSO
    - Auditing
    - Private Data Center Networking
    - Clustering
- **Team Management (Roles) & Governance** features are **NOT available** for **Terraform Cloud Free** (paid).
- **Paid** features:
    - Sentinel Policy as Code Management
    - Cost Estimation
    - Roles/Team Management

Terraform Enterprise

- In Terraform Enterprise **1 (one) workspace can be mapped with VCS repo**.
- **Exclusive Terraform Enterprise** features:
    - Clustering.
    - Private Network Connectivity.
    - Locally hosted installation.

Terraform Cloud

- **Terraform Cloud (TFC) is a free to use, self-service SaaS platform that extends the capabilities of the open source Terraform CLI**. It adds automation and collaboration features, and performs Terraform functionality remotely, making it ideal for collaborative and production environments. **The Terraform CLI performs all runs and stores state locally, unless it is used with a Terraform Cloud or Enterprise account.**
- Each workspace in Terraform Cloud retains bakcups of its previous state files.
- Terraform Cloud features:
    - Private module registry
    - Remote runs
    - VCS connection

Connecting VCS Providers to Terraform Cloud

- You can integrate Terraform Cloud with your version control system (VCS) provider.
- **VCS connection provides:**

    - When **workspaces are linked to a VCS repository**, Terraform Cloud can **automatically initiate Terraform runs** when changes are committed to the specified branch.
    - **Terraform Cloud makes code review** easier by automatically predicting how pull requests will affect infrastructure.
    - Publishing a new version of a private Terraform module is as easy as pushing a tag to the module's repository.
- **Supported VCS Providers**
    - GitHub
    - GitLab
    - Bitbucket
    - Azure

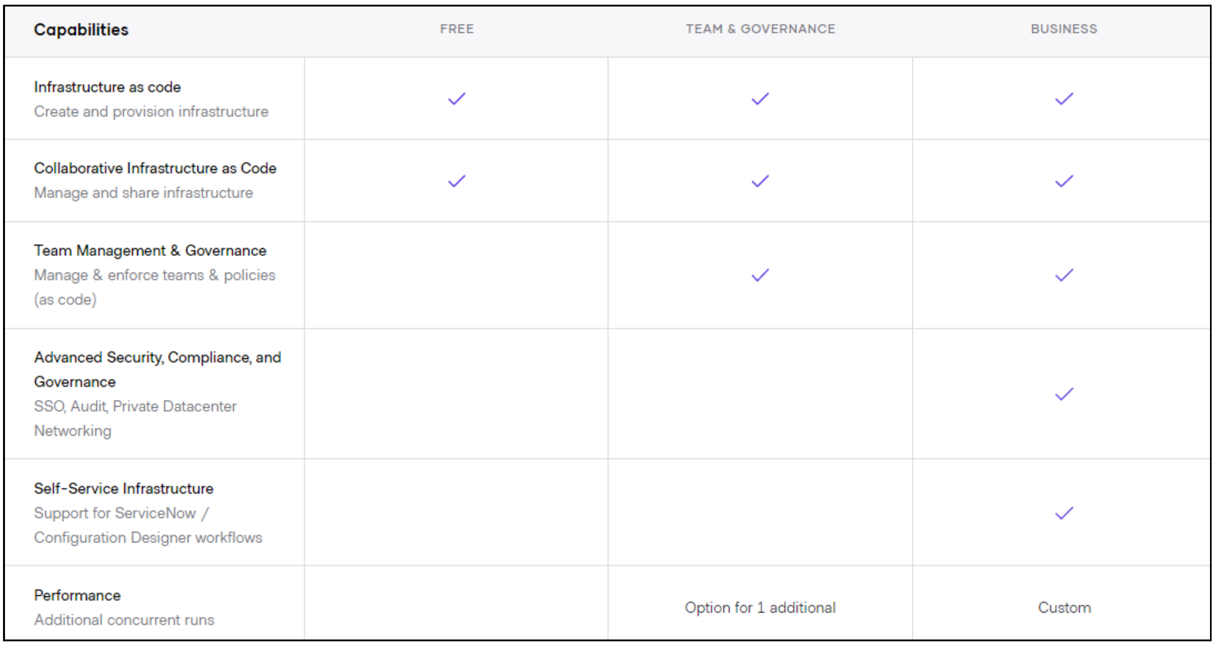
Run Triggers

- Terraform Cloud's **run triggers allow you to link workspaces** so that a successful apply in a source workspace will queue a run in the workspace linked to it with a run trigger.
- **Organizes your configuration into different workspaces** helps you to better manage and design your infrastructure.

Terraform Cloud for Business

- SAML/SSO
- Self-Service Infrastructure
- Audit Logging

Terraform Cloud Compare Offerings



Capabilities	FREE	TEAM & GOVERNANCE	BUSINESS
Infrastructure as code Create and provision infrastructure	✓	✓	✓
Collaborative Infrastructure as Code Manage and share infrastructure	✓	✓	✓
Team Management & Governance Manage & enforce teams & policies (as code)		✓	✓
Advanced Security, Compliance, and Governance SSO, Audit, Private Datacenter Networking			✓
Self-Service Infrastructure Support for ServiceNow / Configuration Designer workflows			✓
Performance Additional concurrent runs		Option for 1 additional	Custom