HashiCorp Certified Terraform Associate

Infrastructure as Code (IaC)

- Automation
- Versioning
- Reusability
- Additional benefits:
    - Code can easily be shared
    - Creates a blueprint of your data center
- IaC makes changes:
    - Idempotent
    - Consistent
    - Repeatable
    - Predictable

Providers

- A **provider** is responsible for understanding **API interactions (API token required).**
- 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.
- **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**.
- Providers are released on a separate schedule from Terraform itself.
- To upgrade to the latest acceptable version of each provider, run **terraform init -upgrade**

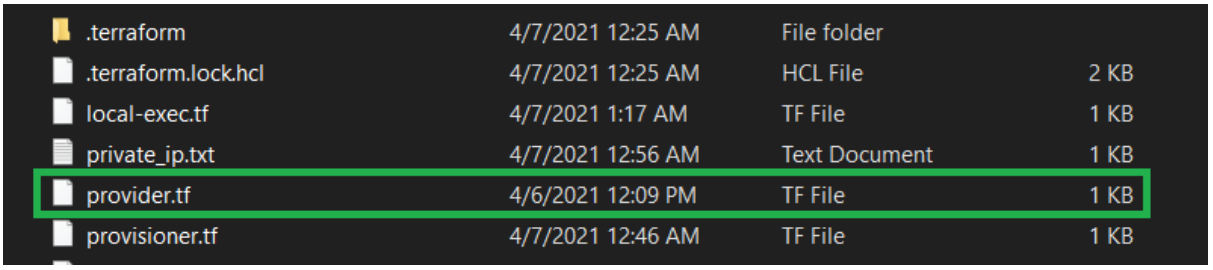
```
provider "aws" {
  region = "us-east-1"
  access_key = "XXXX"
  secret_key = "XXXX"
}
```

- You can have multiple provider instance with the help of alias

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

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

- You can have a TF file only for provider configuration instead of copy and paste this configuration in all TF files.



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.
- Initializes the backend configuration.
- Downloads the declared providers which are supported by HashiCorp.
- It will **NOT** create any sample files like example.tf

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.
- 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**.
- **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**

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 does **modify the state file**
- The following commands will implicitly run terraform refresh:
    - terraform plan
    - terraform destroy
    - terraform import

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 Format

- The **terraform fmt** command is used to **rewrite** Terraform configuration files to **canonical format and style.**
- 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.
- Validate runs checks that verify 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.
- **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.
- NOT all backends supports locking functionality
- 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.
- 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 Taint

- The **terraform taint** can also be used to taint resources with a module.
- terraform taint [options] address

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.
- 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.
terraform state pull	Manually downloads and outputs the 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.

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

remote-exec

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

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

  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**

```
variable "ami-id" {}

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.
  - **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*
  - DEBUG
  - INFO
  - WARN
  - ERROR
- Example: **TF_LOG=TRACE**
- To persist logged output, you can set **TF_LOG_PATH.** Then you have to **set** the **TF_LOG for the logging**.

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"
  }
}
```

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
}
```

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>, ...])**

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.
- **Data source enables Terraform to fetch data** for use elsewhere in the Terraform configuration.
- OS that Terraform is available:
    - Linux
    - macOS
    - Solaris
    - FreeBSD
    - Windows

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.

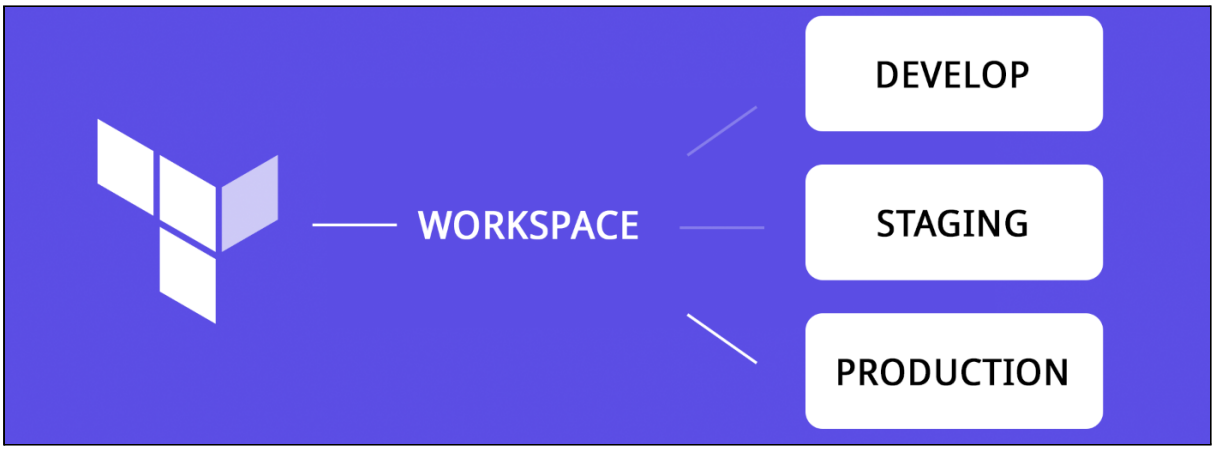
Workspaces

- 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).
- 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.

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

Use Case

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



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"
}
```

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**

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

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.**
- **Syntax** for referencing a registry module is:
    - **<NAMESPACE>/<NAME>/<PROVIDER>**
        - Example: **hashicorp/consul/aws**
- **Published modules** via Git and GitHub **provides:**
    - **Allow browsing version histories.**
    - **Automatically generated documentation.**
    - **Show examples and READMEs.**
    - **Support versioning.**

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**

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

Terminologies

- Resource Type: **aws_instance**
- Local name: **myEC2**
- Arguments
    - 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**
- 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”**

Dependencies

Implicit

- With implicit dependency, Terraform can automatically find references of the object and create an implicit ordering requirement between the two resources

Explicit

- Explicitly specifying a dependency is only necessary when a resource relies on some other resouce’s behavior but doesn’t access any of that resource’s data in its arguments.
    - **depends_on = [aws_s3_bucket.example]**

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.**

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 file 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.

Remote Backend for Terraform Cloud

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

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: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.

```
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"]
```

Exp:Conditional

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

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.
- **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.

```
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.

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**

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)

Count and Count Index

- The count parameter on resources can simplify configurations and let you scale resources by simply incrementing a number
- Quantity - “count = 3”
- Incremental Index - “count.index”

```
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
- **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”**

```
18  variable "tags" {
19    type = list
20    default = ["first","second"]
21  }
22
23  resource "aws_instance" "app-dev" {
24    ami = lookup(var.ami,var.region)
25    instance_type = "t2.micro"
26    count = 2
27    tags = {
28      Name = element(var.tags, count.index)
```

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 {}

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)
```

Filesystem:File

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

Number:Max

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

DAT:Formatdate

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

DISTRIBUTION

Terraform Enterprise & Terraform Cloud

- **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).

Terraform Enterprise

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

Terraform Cloud

- Terraform Cloud for Business
    - SAML/SSO
    - Self-Service Infrastructure
    - Audit Logging
- 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