Terraform Associate Cheat Sheets

Good luck on your exam!

Terraform Associate *CheatSheet*

Infrastructure as Code (IaC) You write a configuration script to **automate** **creating, updating or destroying** cloud infrastructure

- **Popular IaC Tools**
  - Microsoft Azure – ARM Templates, Azure Blueprints
  - Amazon Web Services – AWS CloudFormation, AWS Cloud Development Kit (CDK)
  - Google Cloud Platform – GCP Cloud Deployment Manager
  - HashiCorp - Terraform

IaC tools come in two types: **Declarative** and **Imperative**

Declarative:

- You say what you want, and the rest is filled in. ***Implict***
- More verbose, but zero chance of mis-configuration
- Uses scripting languages eg. JSON, YAML, XML, HCL

Imperative

- What you see is what you get. ***Explicit***
- Less verbose, you could end up with misconfiguration
- Does more than Declarative
- Uses programming languages eg. Python, Ruby, JavaScript

Idempotent is a term to describe that no matter how many times you run an IaC script, environment will be the same everytime.

- E.g. Running a script multiple times will not result in multiple servers. One server defined will always result in one server

Common DevOps Terms:

- **Provisioning**: To prepare a server with systems, data and soft, making a system or service ready for network operations
- **Deployment**: The act of delivering a version of your application to run a provisioned server.
- **Orchestration**: The act of coordination multiple systems or services.

Configuration Drift is when provisioned infrastructure has an unexpected configuration change due to:

- team members manually adjusting configuration options
- malicious actors
- side effects from APIs, SDK or CLIs.

Terraform Associate *CheatSheet*

Exam Pro

Immutable Infrastructure are servers or cloud resources which are never modified after deployment.
GitOps is when you take Infrastructure as Code (IaC) and you use a git repository to introduce a formal process to review and accept changes to infrastructure code, once that code is accepted, it automatically triggers a deploy
HashiCorp is a company specializing in managed open-source tools used to support the development and deployment of large-scale service-oriented software installations
Terraform is an open-source and cloud-agnostic Infrastructure as Code (IaC) tool.

- Terraform uses **declarative** configuration files.
- Terraform language is a subset of HashiCorp Configuration Language

Terraform Cloud is a Software as Service (SaaS) that manages your state file when working in a team. It can integrate with various Version Control Systems (VCS)
Terraform Lifecycle

- **Code** - Write or Update your Terraform configuration file
- **Init -** Initialize your project. Pull latest providers and modules
- **Plan** - Speculate what will change or Generate a saved execution plan
- **Validate** - Ensure types and values are valid. Ensures required attributes are present
- **Apply** - Execute the terraform plan provisioning the infrastructure
- **Destroy** - Destroy the remote infrastructure

Multi-Cloud Deployment – Terraform is cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to even handle cross-cloud dependencies. Terraform supports a variety of providers outside of GCP, AWS, Azure and sometimes is the only provider.
Terraform is open-source an extendable so any API could be used to create IaC tooling for any kind of cloud platform or tech. E.g. Heroku, Spotify playlist
Terraform Core and Terraform Plugins - Terraform is logically split into two main parts:

1. **Terraform Core**
   - uses remote procedure calls (RPC) to communicate with Terraform Plugins
2. **Terraform Plugins**
   - expose an implementation for a specific service, or provisioner

Terraform Associate *CheatSheet*

Terraform Provisioners install software, edit files, and provision machines created with Terraform (Provisioners should only be used as a **last resort**)
Terraform allows you to work with two different provisioners:

- **Cloud-Init** is an industry standard for cross-platform cloud instance initializations.
- **Packer** is an automated image-builder service.

Local-exec allows you to execute **local commands** *after* a resource is provisioned.
A local environment could be: **Local Machine -** Your laptop/workstation,
Build Server eg. GCP Cloud Build, AWS CodeBuild, Jenkins, **Terraform Cloud Run Environment -** single-use Linux virtual machine

Local-exec

command (required)

- The command you want to execute

working_dir

- Where the command will be executed

eg. /user/andrew/home/project

interpreter

- The entry point for the command.

What local program will run the command eg. Bash, Ruby, AWS CLI, PowerShell

Environment

- Key and value pair of environment variables

Remote-exec

Remote-exec allows you to execute **commands on a target resource** *after* a resource is provisioned.
Remote-Exec is useful for provisioning a Virtual Machine with a simple set of commands.
Remote Command has three different modes:
Inline - list of command strings
Script - relative or absolute local script that will be copied to the remote resource and then executed
Scripts - relative or absolute local scripts that will be copied to the remote resource and then executed and executed in order.
You can only choose to use one mode at a time

Terraform Associate CheatSheet

file provisioner is used to copy files or directories from our local machine to the newly created resource
Source – the local file we want to upload to the remote machine
Content – a file or a folder
Destination – where you want to upload the file on the remote machine
A connection block tells a **provisioner** or **resource** how to establish a connection
null_resource is a placeholder for resources that have no specific association to a provider resources.

Terraform Providers

Providers are Terraform Plugins that allow you to interactive with:

- Cloud Service Providers (CSPs) eg. **AWS, Azure, GCP**
- Software as a Service (SaaS) Providers eg. **Github, Angolia, Stripe**
- Other APIs eg. **Kubernetes, Postgres**

Providers are required for your Terraform Configuration file to work.
Providers come in three tiers:
Official — Published by the company that owns the provider technology or service
Verified — actively maintained, up-to-date and compatible with both Terraform and Provider
Community — published by a community member but no guarantee of maintenance, up-to-date or compatibility
Providers are distributed separately from Terraform and the plugins must be downloaded before use.
terraform init will download the necessary provider plugins listed in a Terraform configuration file.
Terraform Registry is a **website portal** to **browse, download or publish** available **Providers** or **Modules** **https://registry.terraform.io**

Provider

A provider is a plugin that is mapping to a Cloud Service Provider (CSPs) API.

Module

A module is a group of configuration files that provide common configuration functionality.

- Enforces best practices
- reduce the amount of code
- Reduce time to develop scripts

Terraform Associate *CheatSheet*

Exam Pro

Terraform Language

Terraform files contain the configuration information about **providers** and **resources**.

Terraform files end in the extension of **.tf** or either **.tf.json**

Terraform files are written in the **Terraform Language** and is the extension of HCL

Terraform language consists of only a few basic elements:

- **Blocks —** containers for other content, represent an object
  - block type — can have zero or more labels and a body
  - block label — name of a block
- **Arguments —** assign a value to a name
  - They appear within blocks
- **Expressions —** represent a value, either literally or by referencing and combining other values
  - They appear as values for arguments, or within other expressions.

Identifiers, uniquely identify objects of constructs in a Terraform Language:

- Argument names
- Block type names
- names Terraform-specific constructs eg. resources, input variables

The special **terraform configuration block type** eg. **terraform { ... }** is used to configure some behaviors of Terraform itself

In Terraform settings we can specify:

- **required_version**
  - The expected version of terraform
- **required_providers**
  - The providers that will be pull during an terraform init
- **experiments**
  - Experimental language features, that the community can try and provide feedback
- **provider_meta**
  - module-specific information for providers

Terraform Associate CheatSheet

Exam Pro

Input variables (aka variables or Terraform Variables) are **parameters** for Terraform modules
You can declare variables in either:

- The root module
- The child modules

Default A default value which then makes the variable optional
type This argument specifies what value types are accepted for the variable
Description This specifies the input variable's documentation
Validation A block to define validation rules, usually in addition to type constraints
Sensitive Limits Terraform UI output when the variable is used in configuration
Variable definition files are named **.tfvars** or **tfvars.json**
By default **terraform.tfvars** will be autoloaded when included in the root of your project directory
A variable value can be defined by Environment Variables
Variable starting with **TF_VAR_**<name> will be read and loaded
Output Values are computed values after a **Terraform apply** is performed.
Outputs allows you:

- to obtain information after resource provisioning e.g. public IP address
- output a file of values for programmatic integration
- Cross-reference stacks via outputs in a state file via terraform_remote_state

To print all the outputs for a state file use the **terraform output**
Print a specific output with **terraform output <name>**
A local value (**locals**) **assigns a name to an expression**, so you can **use it multiple times within a module** without repeating it.
Data sources allow Terraform **use information defined outside of Terraform**, defined by another separate Terraform configuration, or modified by functions.
You specific what kind of external resource you want to select
You use **data.** to reference data sources

Terraform Associate CheatSheet

Exam Pro

References to Named Values

Named Values are **built-in expressions** to **reference various values** such as:

Resources	<Resource Type>.<Name> e.g. aws_instance.my_server
Input variables	**var**.<Name>
Local values	**local**.<Name>
Child module outputs	**module**.<Name>
Data sources	**data**.<Data Type>.<Name>

Resource Meta Arguments

Terraform language defines several meta-arguments, which can be used with any **resource type** **to change the behavior of resources**.

depends_on allows you to explicitly specific a dependency of a resource.

count for creating multiple resource instances according to a count

- Get the current count **value** (index) via **count.index**
- **This value starts at 0**

for_each to create multiple instances according to a map, or set of strings

With a map:

- **each.key** – print out the current key
- **each.value** – print out the current value

With a list:

- **each.key** – print out the current key

provider, for selecting a non-default provider configuration

lifecycle, for lifecycle customizations

provisioner and connection, for taking extra actions after resource creation

Lifecycle block allows you to change what happens to resource e.g. create, update, destroy.

Lifecycle blocks are nested within resources

Terraform Associate CheatSheet

Resource Providers and Alias If you need to override the default provider for a resource you can create alternative provider with **alias**
You reference the alias under in the attribute **provider** for a resource.

Types and Values

Strings

When quoting strings, you use **double quotes** e.g. ami = "ami-830c94e3"

Strings Templates

String interpolation allows you to evaluate an expression between the markers e.g. **${ …. }** and coverts it to a string.

Operators are **mathematical operations** you can perform to numbers within expressions e.g.

- Multiplication a * b
- Division a / b
- Modulus a % b
- Addition a + b

Conditional Expressions - Terraform support ternary if else conditions.

```
condition ? true_val : false_val
```

For expressions allows you to iterative over a complex type and apply transformations

A for expression can accept as input **a list, a set, a tuple, a map, or an object**.

Implicit Element Ordering on Conversion

Since Terraform can convert a unordered type (maps objects and sets) to a ordered type (list and tuples) it will need to choose an implied ordering.

- **Maps and Objects** – stored by key A-Z
- **Sets of Strings** – stored by string A-Z
- **Everything else** – arbitrary ordering

A **splat** expression provides a **shorter expression** for **for expressions**

What is a splat operator?

A splat operator is represented by an asterisk (*), it originates from the ruby language

Splats in Terraform are used to rollup or soak up a bunch of iterations in a *for expression*

Splat expressions have a special behavior when you apply them to **lists**

Terraform Associate CheatSheet

Exam Pro

Dynamic blocks allows you **dynamically construct repeatable nested blocks**
Terraform utilizes Semantic Versioning for specifying Terraform, Providers and Modules versions
A **version constraint** is a string containing one or more conditions, separated by commas.

- = or no operator. Match exact version number e.g. “1.0.0”, “=1.0.0”
- != Excludes an exact version number e.g. “!=1.0.0”
- > >= < <= Compare against a specific version e.g. “>= 1.0.0”
- ~> Allow only the rightmost version (last number) to increment e.g. ~> 1.0.0”

What is State?
A particular condition of cloud resources at a specific time.
e.g. We expect there to be a Virtual Machine running CentOS on AWS with a compute type of t2.micro.
How does Terraform preserve state?
When you provision infrastructure via Terraform it will create a state file named **terraform.tfstate**
This **state file is a JSON data structure** with a one-to-one mapping from **resource instances** to **remote objects**
Configuration files:

- main.tf
- variables.tf
- terraform.tfvars

State file - terraform.tfstate
terraform state **list -** List resources in the state
terraform state **mv -** Move an item in the state, can rename existing resources, move a resource into a module, move a module into a module
terraform state **pull -** Pull current remote state and output to stdout (Standard output)
terraform state **push -** Update remote state from a local state
terraform state **replace-provider -** Replace provider in the state
terraform state **rm -** Remove instances from the state
terraform state **show -** Show a resource in the state

Terraform Associate CheatSheet

Exam Pro

Terraform State Backups

All terraform state subcommands that *modify state* will write a backup file.

Read only commands will not modify state e.g. list, show

Terraform will take the current state and store it in a file called **terraform.tfstate.backup**

terraform init initializes your terraform project by:

- Downloading plugin dependencies e.g. Providers and Modules
- Create a .terraform directory
- Create a dependency lock file to enforce expected versions for plugins and terraform itself.

Terraform init is generally the first command you will run for a new terraform project

If you modify or change dependencies run **terraform init** again to have it apply the changes

terraform init **-upgrade -** Upgrade all plugins to the latest version that complies with the configuration's version constraint

terraform init **-get-plugins-false -** Skip plugin installation

terraform init **-plugin-dir=PATH -** Force plugin installation to read plugins only from target directory

terraform init -**lockfile=MODE -** Set a dependency lockfile mode

Dependency lock file: **.terraform.lock.hcl**

State lock file: **.terraform.tfstate.lock.hcl**

terraform get command is used to download and **update modules** in the root module.

Terraform has three CLI commands that **improve debugging configuration scripts**:

terraform **fmt -** rewrites Terraform configuration files to a standard format and style

terraform **validate -** validates the syntax and arguments of the Terraform configuration files in a directory

terraform **console -** an **interactive shell** for evaluating Terraform expressions

Terraform plan creates an **execution plan** *(aka Terraform Plan).*

A plan consist of:

- Reading the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
- Comparing the current configuration to the prior state and noting any differences.
- Proposing a set of change actions that should, if applied, make the remote objects match the configuration.

Terraform Associate CheatSheet

Terraform plan **does not carry out** proposed changed.

Speculative Plans

When you run **terraform plan -** Terraform will output the description of the effect of the plan but without any intent to actually apply it.

Saved Plans

When you run **terraform plan -out=FILENAME**

You can generate a saved plan file which you can then pass along to terraform apply. e.g. **terraform apply FILE (**terraform apply [options] [plan file])

Terraform apply command executes the actions proposed in an execution plan

Automatic Plan Mode

When you run **terraform apply**

Executes plan, validate and the apply.

Requires users to **manually approve** the plan by writing "yes"

Terraform apply **-auto-approve** flag will automatically approve the plan.

Saved Plan Mode

When you provide a filename to terraform to saved plan file **terraform apply FILE**

Drift (Configuration or Infrastructure) is when your expected resources are in different state then your expected state

We can resolve Drift in three ways in Terraform:

Replacing Resources

When a resource has become damaged or degraded that cannot be detected by Terraform, we can use the **–replace** flag

Importing Resources

When an approved manual addition of resource needs to be added to our state file

We use the **import** command.

Refresh State

When an approved manual configuration of a resource has changed or removed

We use the **–refresh-only** flag to reflect the changes in our state file

Terraform Associate *CheatSheet*

Exam Pro

Terraform taint is used to **mark a resource** for *replacement*, the next time you run apply.
A cloud resource can become **degraded or damaged** and you want to return the expected resource to a healthy state
It is recommended in 0.152+ to use the **-replace** flag and providing a resource address
Terraform apply -replace="aws_instance.example[0]"
The replace flag appears to only work for a single resource
A resource address is a string that **identifies zero or more resource instances** in your configuration.
An address is composed of two parts:

[module path][resource spec]

module.module_name[module index] resource_type.resource_name[instance index]

module A namespace for modules
module_name User-defined name of the module
[module index] When multiple specific an index
resource_type Type of the resource being addressed
resource_name User-defined name of the resource
[instance index] when multiple specific an index
E.g. aws_instance.**web**[3]
Select the 4th Virtual Machine
The **terraform import** command is used to import ***existing resources*** into Terraform
The **terraform refresh** command **reads the current settings** from all managed remote objects **and updates the Terraform state to match.**
The terraform refresh command is an alias for the refresh only and auto approve
Terraform apply –refresh-only –auto-approve
The **–refresh-only** flag for terraform plan or apply allows you to refresh and update your state file without making changes to your remote infrastructure.
There are **four types of errors** you can encounter with Terraform:
Language errors, State errors, Core errors, Provider errors

Terraform Associate CheatSheet

Exam Pro

Debugging Terraform

Terraform has detailed logs which can be enabled by setting the **TF_LOG environment** variable to:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- JSON — outputs logs at the TRACE level or higher and uses a parseable JSON encoding as the formatting.

Logging can be enabled separately:

- TF_LOG_CORE
- TF_LOG_PROVIDER

takes the same options as TF_LOG

Choose where you want to log with **TF_LOG_PATH**

If Terraform ever crashes (a "panic" in the **Go runtime**), it saves a log file with the debug logs from the session as well as the panic message and back trace to **crash.log**

Terraform Modules can be publicly found in the **Terraform Registry**

*Only **verified modules** will be displayed in search terms*

Public Modules

Terraform Registry is integrated **directly into** Terraform

The syntax for specifying a registry module is <NAMESPACE>/<NAME>/<PROVIDER>

terraform init command will **download and cache any modules** referenced by a configuration

Private registry modules have source strings of the form <HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>

To configure private module access, you need to authenticate against Terraform Cloud via **terraform login**

Anyone can publish and share modules on the Terraform Registry.

Repos names must be in the following format: terraform-<PROVIDER>-<NAME>

Terraform Associate CheatSheet

Exam Pro

Verified modules are reviewed by HashiCorp and actively maintained by contributors to stay up-to-date and compatible with both Terraform and their respective providers.
The Standard Module Structure is a file and directory layout recommend for module development
The primary entry point is the **Root Module.**
These are required files in the root directory:

- Main.tf - the entry point file of your module
- Variables.tf – variable that can be passed in
- Outputs.tf – Outputted values
- README – Describes how the module works
- LICENSE – The license under which this module is available

Nested modules are optional must be contained in the **modules/** directory:

- A submodule that contains a README is consider usable by external users
- A submodule that does not contain a README is consider internal use only
- Avoid using relative paths when sourcing module blocks.

Terraform Workflows
The core Terraform workflow has three steps:

1. **Write** - Author infrastructure as code.
2. **Plan** - Preview changes before applying.
3. **Apply** - Provision reproducible infrastructure.

Team Workflows — Individual Practitioner
Write

- You write your Terraform configuration in your editor of choice
- You'll store your terraform code in a VCS e.g. GitHub
- You repeatedly run terraform plan and validate to find syntax errors.
- Tight feedback loop between editing code and running test commands

Terraform Associate CheatSheet

Plan

- When the developer is confident with their work in the write step, they commit their code to their local repository.
- They may be only using a single branch e.g. main
- Once their commit is written they'll proceed to apply

Apply

- They will run terraform apply and be prompted to review their plan
- After their final review they will approve the changes and await provisioning
- After a successful provision they will push their local commits to their remote repository.

Team Workflows — Team

Write

- Each team members writes code locally on their machine in their editor of choice
- A team member will store their code to a branch in their code repository
    - Branches help avoid conflicts while a member is working on their code
- Terraform plan can be used as a quick feedback loop for small teams
- For larger teams, a concern over sensitive credentials becomes a concern.
    - A CI/CD process may be implemented so the burden of credentials is abstracted away

Plan

- When a branch is ready to be incorporated on Pull Request an Execution Plan can be generated and displayed within the Pull Request for Review

Apply

- To apply the changes the merge needs to be approved and merged, which will kick of a code build server that will run terraform apply.

Team Workflows — Terraform Cloud

Write

- A team will use Terraform Cloud as their remote backend.
- Inputs Variables will be stored on Terraform Cloud instead of the local machines.
- Terraform Cloud integrates with your VCS to quickly setup a CI/CD pipeline
- A team member writes code to branch and commits per usual

Terraform Associate CheatSheet

Plan

- A pull request is created by a team member and Terraform Cloud will generate the speculative plan for review in the VCS. The member can also review and comment on the plan in Terraform Cloud.

Apply

- After the Pull request is merge Terraform Cloud runtime environment will perform a Terraform apply. A team member can Confirm and apply the changes.

Terraform Cloud streamlines a lot of the CI/CD effort, storing and securing sensitive credentials and makes it is easier to go back and audit the history of multiple runs.

Terraform's backends are divided into two types:

Standard backends

- only store state
- does not perform terraform operations e.g. Terraform apply
    - To perform operations, you use the CLI on your local machine
- **third-party backends** are Standard backends e.g. AWS S3

Enhanced backends

- can both store state
- can perform terraform operations

enhanced backends are subdivided further:

- **local** – files and data are stored on the local machine executing terraform commands
- **remote** – files and data are stored in the cloud e.g. Terraform Cloud

The local backend:

- stores state on the local filesystem
- locks that state using system APIs
- performs operations locally

By default, you are using the backend state when you have no specified backend

You specific the backend with argument **local**, and you can change the path to the local file and **working_directory**

Terraform Associate CheatSheet

A **Remote backend** uses the Terraform platform which is either:

- Terraform Cloud
- Terraform Enterprise

With a remote backend when terraform apply is performed via the CLI
The Terraform Cloud Run Environment is responsible for executing the operation
Because the Terraform Cloud Run Environment is executing the command. Your provider credentials **need to be configured** in Environment Variables in Terraform Cloud
When using a remote backend, you need to set a Terraform Cloud **Workspaces**
You can set a single workspace via **name** **E.g., name = "my-app-prod"**
You can set multiple workspaces via **prefix** **E.g., prefix = "my-app-"**
The **-backend-config** flag for **terraform init** can be used for partial backend configuration
terraform_remote_state data source **retrieves the root module output values from another Terraform configuration**, using the latest state snapshot from the remote backend.

State Locking
Terraform will lock your state for all operations that could write state.
This prevents others from acquiring the lock and potentially corrupting your state

Disabling Locking
You can disable state locking for most commands with the **-lock** flag but it is not recommended

Force Unlock
Terraform has a **force-unlock** command to manually unlock the state if unlocking failed.
If you unlock the state when someone else is holding the lock **it could cause multiple writers**.
Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed.
To protect you, the force-unlock command **requires a unique lock ID**

- Terraform will output this lock ID if unlocking fails

-force flag will skip use confirmation

Terraform Associate CheatSheet

Exam Pro

Terraform State file **can contain sensitive data** eg. long-lived AWS Credentials
and is a possible **attack vector** for malicious actors.

Local State

When using local backend, state is stored in plain-text JSON files.

- You need to be careful you don't share this state file with anyone
- You need to be careful you don't commit this file to your git repository

Remote State with Terraform Cloud

When using the Terraform Cloud remote backend:

- That state file is held in memory and is not persisted to disk
- The state file is encrypted-at-rest
- The state file is encrypted-in-transit
- With Terraform Enterprise you have detailed audit logging for tamper evidence

Remote State with Third-Party Storage

You can store state with various third-party backends.

Resources in configuration files represent **infrastructure objects** e.g.
Virtual Machines, Databases, Virtual Network Components, Storage…..

A **resource type** determines the kind of infrastructure object:
e.g. **aws_instance** is an AWS EC2 instance

A resource belongs to a provider.

Some resource types provide a **special timeouts nested block** argument that allows you to customize how long certain operations are allowed to take before being considered to have failed

A ***complex* type** is a type that groups multiple values into a single value.

Complex types are represented by type constructors, but several of them also have shorthand keyword versions.

There are two categories of complex types:

- **collection types** (for grouping similar values) - **List, Map, Set**
- **structural types** (for grouping potentially dissimilar values) - **Tuple, Object**

Terraform Associate *CheatSheet*

Exam Pro

List – It's like an array, you use an integer as the index to retrieve the value
Map It's like a ruby hash or single nested Json object. You use a key as the index to retrieve the value
Set - Is similar to a list but has no secondary index or preserved ordering, all values must of the same type and will be cast to match based on first element
A ***structural* type** allows multiple values of *several distinct types* to be grouped together as a single value.
Structural types require a ***schema*** as an argument, to specify which types are allowed for which elements.
The two kinds of **structural type**, **object** and **tuple**
Object is a map with more explicit keying: object({ name=string, age=number })
Tuple. Multiple return types with parameters: tuple([string, number, bool])

Numeric Functions

abs returns the absolute value of the given number
ceil returns the closest whole number that is greater than or equal to the given value
floor returns the closest whole number that is less than or equal to the given value, which may be a fraction
min takes one or more numbers and returns the smallest number from the set
Log returns the logarithm of a given number in a given base
max takes one or more numbers and returns the greatest number from the set
parseint parses the given string as a representation of an integer in the specified base and returns the resulting number
pow calculates an exponent, by raising its first argument to the power of the second argument.
signum determines the sign of a number, returning a number between -1 and 1 to represent the sign

String Functions

chomp removes newline characters at the end of a string.
format produces a string by formatting a number of other values according to a specification string
formatlist produces a list of strings by formatting a number of other values according to a specification string
indent adds a given number of spaces to the beginnings of all but the first line in a given multi-line string
join produces a string by concatenating together all elements of a given list of strings with the given delimiter
lower converts all cased letters in the given string to lowercase.
regex applies a regular expression to a string and returns the matching substrings

Terraform Associate *CheatSheet*

Exam Pro

regexall applies a regular expression to a string and returns a list of all matches
replace searches a given string for another given substring, and replaces each occurrence with a given replacement string
split produces a list by dividing a given string at all occurrences of a given separator.
strrev reverses the characters in a string
substr extracts a substring from a given string by offset and length
title converts the first letter of each word in the given string to uppercase
trim removes the specified characters from the start and end of the given string
trimprefix removes the specified prefix from the start of the given string. If the string does not start with the prefix, the string is returned unchanged
trimsuffix removes the specified suffix from the end of the given string
trimspace removes all types of whitespace from both the start and the end of a string
upper converts all cased letters in the given string to uppercase

Collection Functions

alltrue returns true if all elements in a given collection are true or "true". It also returns true if the collection is empty.
anytrue returns true if any element in a given collection is true or "true". It also returns false if the collection is empty
chunklist splits a single list into fixed-size chunks, returning a list of lists
coalesce takes any number of arguments and returns the first one that isn't null or an empty string
coalescelist takes any number of list arguments and returns the first one that isn't empty
compact takes a list of strings and returns a new list with any empty string elements removed
concat takes two or more lists and combines them into a single list
contains determines whether a given list or set contains a given single value as one of its elements
distinct takes a list and returns a new list with any duplicate elements removed
element retrieves a single element from a list
Index finds the element index for a given value in a list
flatten takes a list and replaces any elements that are lists with a flattened sequence of the list contents
keys takes a map and returns a list containing the keys from that map
length determines the length of a given list, map, or string

Terraform Associate *CheatSheet*

Lookup 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.
Matchkeys constructs a new list by taking a subset of elements from one list whose indexes match the corresponding indexes of values in another list
merge takes an arbitrary number of maps or objects, and returns a single map or object that contains a merged set of elements from all arguments
one takes a list, set, or tuple value with either zero or one elements. If the collection is empty, one returns null. Otherwise, one returns the first element. If there are two or more elements, then one will return an error
range generates a list of numbers using a start value, a limit value, and a step value
reserve takes a sequence and produces a new sequence of the same length with all of the same elements as the given sequence but in reverse order
setintersection function takes multiple sets and produces a single set containing only the elements that all of the given sets have in common. In other words, it computes the intersection of the sets
setproduct function finds all of the possible combinations of elements from all of the given sets by computing the Cartesian product.
Setsubtract function returns a new set containing the elements from the first set that are not present in the second set. In other words, it computes the relative complement of the first set in the second set
setunion function takes multiple sets and produces a single set containing the elements from all of the given sets. In other words, it computes the union of the sets
slice extracts some consecutive elements from within a list
sort takes a list of strings and returns a new list with those strings sorted lexicographically
sum takes a list or set of numbers and returns the sum of those numbers
transpose takes a map of lists of strings and swaps the keys and values to produce a new map of lists of strings
values takes a map and returns a list containing the values of the elements in that map
zipmap constructs a map from a list of keys and a corresponding list of values

Encoding and Decoding Functions

Functions that will **encode** and **decode** for various formats

- Base64encode, jsonencode, textencodebase64, Yamlencode, base64gzip, **urlencode**
- Base64decode, csvdecode, jsondecode, textdecodebase64, yamldecode

Terraform Associate *CheatSheet*

Exam Pro

Filesystem Functions

abspath takes a string containing a filesystem path and converts it to an absolute path. That is, if the path is not absolute, it will be joined with the current working directory

dirname takes a string containing a filesystem path and removes the last portion from it.

pathexpand takes a filesystem path that might begin with a ~ segment, and if so, it replaces that segment with the current user's home directory path

basename takes a string containing a filesystem path and removes all except the last portion from it

file reads the contents of a file at the given path and returns them as a string

fileexists determines whether a file exists at a given path

fileset enumerates a set of regular file names given a path and pattern

filebase64 reads the contents of a file at the given path and returns them as a base64-encoded string

templatefile reads the file at the given path and renders its content as a template using a supplied set of template variables

Date and Time Functions

formatdate converts a timestamp into a different time format

timeadd adds a duration to a timestamp, returning a new timestamp

timestamp returns a UTC timestamp string in RFC 3339 format

Hash and Crypto Functions

Generate Hashes and cryptographic strings.

- Base64sha256, base64sha512, **bcrypt,** filebase64sha256, filebase64sha512, filemd5, filesha1, filesha256
- Filesha512, **md5,** rsadecrypt, **sha1,** sha256, sha512, **uuid,** uuidv5

IP Network Functions

- **cidrhost** calculates a full host IP address for a given host number within a given IP network address prefix
- **cidrnetmask** converts an IPv4 address prefix given in CIDR notation into a subnet mask address
- **cidrsubnet** calculates a subnet address within given IP network address prefix
- **cidrsubnets** calculates a sequence of consecutive IP address ranges within a particular CIDR prefix.

Terraform Associate CheatSheet

Type Conversion Functions

can evaluates the given expression and returns a Boolean value indicating whether the expression produced a result without any errors

defaults a specialized function intended for use with input variables whose type constraints are object types or collections of object types that include optional attributes

nonsensitive takes a sensitive value and returns a copy of that value with the sensitive marking removed, thereby exposing the sensitive value

sensitive takes any value and returns a copy of it marked so that Terraform will treat it as sensitive, with the same meaning and behavior as for sensitive input variables.

tobool converts its argument to a Boolean value

tomap converts its argument to a map value

toset converts its argument to a set value.

try evaluates all of its argument expressions in turn and returns the result of the first one that does not produce any errors

tolist converts its argument to a list value

tonumber converts its argument to a number value

tostring converts its argument to a set value

Terraform Cloud is an application that helps teams use Terraform together.

Terraform Cloud features:

- Manages your state files
- History of previous runs
- History of previous states
- Easy and secure variable injection
- Tagging
- Run Triggers (chaining workspaces)
- Specify any version of terraform per workspace
- Global state sharing
- Commenting on Runs

Terraform Associate CheatSheet

Exam Pro

- Notifications via Webhooks, Email, Slack
- Organization and Workspace Level Permissions
- Policy as Code (via Sentinel Policy Sets), MFA
- Single Sign On (SSO) (at Business tier)
- Cost Estimation (at Teams and Governance Tier)
- Integrations with ServiceNow, Splunk, K8, and custom Run Tasks

Terraform Cloud is available as a hosted service at app.terraform.io

Terraform Cloud - Terms

Organizations - An organization is a collection of workspaces

Workspaces - A workspace belongs to an organization, A workspace represents a unique environment or stack.

Teams - A team is composed of multiple members (users), A team can be assigned to workspaces

Runs - A run represents a single-run of the terraform run environment that is operating on an execution plan.

- Runs can be UI/VCS driven API driven or CLI driven

Terraform Cloud Run Workflows

Terraform Cloud offers **3 types** of Cloud Run Workflows

When you create a workspace, you must **choose a workflow**

Version control workflow (most common)

- Terraform Cloud is integrated with a specific branch in your VCS eg. Github via webhooks.
- Whenever pull requests are submitted for the branch speculative plans are generated
- Whenever a merge occurs to that branch, then a run is triggered on Terraform Cloud

CLI-driven workflow

Runs are triggered by the user running terraform CLI commands e.g. terraform apply and plan locally on their own machine.

API-driven workflow

- Workspaces are not directly associated with a VCS repo, and runs are not driven by webhooks on your VCS provider
- A third-party tool or system will trigger runs via upload a configuration file via the Terraform Cloud API
- The configuration file is a bash script that is packaged in an archive (.tar.gz). You are pushing a **Configuration Version**

Terraform Associate CheatSheet

Organization-Level Permissions manage certain resources or settings **across an organization**

- **Manage Policies** - create, edit, and delete the organization's Sentinel policies
- **Manage Policy Overrides** - override soft-mandatory policy checks.
- **Manage Workspaces** - create and administrate all workspaces within the organization
- **Manage VCS Settings** - set of VCS providers and SSH keys available within the organization

Organization Owners

Every organization has organization owner(s). This is a special role that has every available permission and some actions only available to owners: E.g.

- Publish private modules
- Invite users to organization
- Manage team membership
- View all secret teams
- Manage organization permissions
- And more

Workspace-Level Permissions manage resource and settings for a **specific workspace**

General Workspace Permissions

- Granular permissions you can apply to a user via **custom workspace permissions**
- Read runs
- Queue plans
- Apply runs
- Lock and unlock workspaces
- Download sentinel mocks
- Read variable
- Read and write variables
- Read state outputs
- Read state versions
- Read and write state versions

Terraform Associate CheatSheet

Fixed Permission Sets - Premade permissions for quick assignment.

- **Read**
  - Read runs
  - Read variables
  - Read state versions
- **Plan**
  - Queue Plans
  - Read variables
  - Read state versions
- **Write**
  - Apply runs
  - Lock and unlock workspaces
  - Download Sentinel mocks
  - Read and write variables, Read and write state versions

Workspace Admins

A workspace admin is a special role that grants the all level of permissions and some workspace-admin-only permissions:

- Read and write workspace settings
- Set or remove workspace permissions of any team
- Delete workspace

Terraform Cloud – API Tokens

Organization API Tokens

- Have permissions across the entire organization
- Each organization can have one valid API token at a time
- Only organization owners can generate or revoke an organization's token.
- Organization API tokens are designed for creating and configuring workspaces and teams.
  - Not recommended as an all-purpose interface to Terraform Cloud

Terraform Associate CheatSheet

Team API Tokens

- allow access to the workspaces that the team has access to, without being tied to any specific user
- Each team can have **one** valid API token at a time
- any member of a team can generate or revoke that team's token
- When a token is regenerated, the previous token immediately becomes invalid
- designed for performing API operations on workspaces.
- same access level to the workspaces the team has access to

User API Tokens

- Most flexible token type because they inherit permissions from the user they are associated with
- Could be for a real user or a machine user.

Terraform Cloud allows you to **publish private modules** for your Organization within the **Terraform Cloud Private Registry**

Terraform Cloud's private module registry helps you share Terraform modules across your organization.

It includes support for:

- module versioning
- a searchable and filterable list of available modules
- a configuration designer

Authentication

You can use either a user token or a team token for authentication, but the type of token you choose may grant different permissions.

Using terraform login will obtain a user token

To use a team token, you'll need to manually set it to your terraform CLI configuration file

Cost Estimation is a feature to get a monthly cost of resources display alongside your runs.

Cost Estimation is available with Teams and Governance plan and above

Cost Estimation is only for specific cloud resources for AWS, Azure and GCP.

Migrating Default Local State

To migrate a local, Terraform project that only uses the **default** workspace to Terraform Cloud

Terraform Associate *CheatSheet*

Exam Pro

- Create a workspace in Terraform Cloud
- Replace your Terraform Configuration with **a remote backend**.

Run terraform init, and copy the existing state by typing **"yes"**

Terraform Cloud VCS Integration

Terraform Cloud can integrate with the following Version Control Systems (VCS):

- Github
- Github (OAuth)
- Github Enterprise
- GitLab
- GitLab EE and CE
- Bitbucket Cloud
- Bitbucket Server and Data Center
- Azure DevOps Service
- Azure DevOps Server

When Terraform Cloud executes your terraform plan it runs them in its own **Run Environment**.

A **Run Environment** is a Virtual Machine or container intended for the execution of code for a specific runtime environment. A run environment is essentially a code build server.

Terraform Cloud will inject the following environment variables automatically on each run:

- **TFC_RUN_ID** - a unique identifier for this run (e.g. "run-CKuwsxMGgMd3W7Ui").
- **TFC_WORKSPACE_NAME** - name of the workspace used in this run
- **TFC_WORKSPACE_SLUG** - full slug of the configuration used in this run org/workspace eg. "acme-corp/prod-load-balancers".
- **TFC_CONFIGURATION_VERSION_GIT_BRANCH** - name of the branch used eg. "main"
- **TFC_CONFIGURATION_VERSION_GIT_COMMIT_SHA** - full commit hash of the commit used
- **TFC_CONFIGURATION_VERSION_GIT_TAG** - name of the git tag used eg. 1.1.0

Terraform Cloud Agents is a paid feature of the **Business** plan to allow Terraform Cloud to **communicate with isolated, private or on-premise infrastructure**.

Terraform Associate CheatSheet

Terraform Enterprise is our **self-hosted distribution** of Terraform Platform
Terraform Enterprise offers a private instance of the Terraform Platform application with benefits such as :

- no resource limits
- with additional enterprise-grade architectural features
  - Audit logging
  - SAML single sign-on (SSO)

Operational mode: - how data should be stored

- External Services
  - Postgres
  - and AWS S3 Bucket, GCP Cloud Storge Bucket or Azure Blob Storage or Minio Object Storage
- Mounted Disk – stores data in a separate directory on a host intended for an external disk e.g. EBS, iSCSI
- Demo – Stores all the data on the instance, data can be backed up with snapshots*, not recommend for production use*

Credentials – Ensure you have credentials to use Enterprise and have Secure connection

- Terraform Enterprise License – You obtain a license form HashiCorp
- TLS Certificate and Private Key – You need to prove you're your own TLS Certificate

Linux Instance – Terraform Enterprise is designed to run on Linux:

Air Gapped Environments

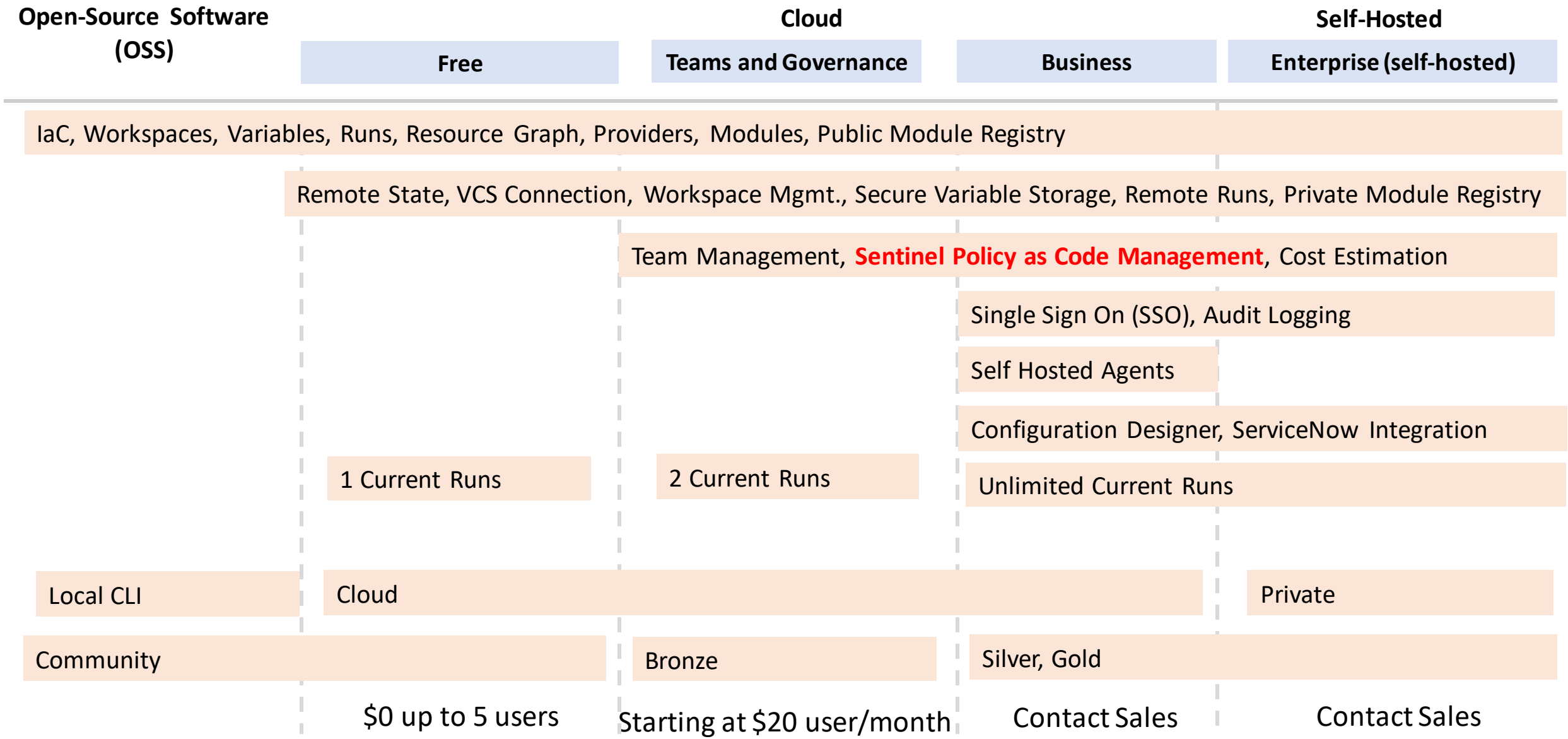
Air Gap or disconnected network is a network security measure employed on one or more computers to ensure that a secure computer network is physically isolated from unsecure networks e.g. Public Internet. **Public Sector** (e.g. government, military) or **large enterprise**
No internet. No outside connectivity. HashiCorp Terraform Enterprise **supports** an installation type for Air Gapped Environments

Workplaces allows you to manage multiple environments or alternate state files.
e.g. Development, Production
There are two variants of workspaces:

- CLI Workspaces – A way of managing alternate state files (locally or via remote backends)
- Terraform Cloud Workspaces – acts like completely separate working directories

By default, you already have a single workspace in your local backend called **default**

Terraform Cloud Features and Pricing



Terraform Associate CheatSheet

Depending if you a local or remote backend changes how the state file is stored

Local State

- Terraform stores the workspace states in a folder called **terraform.tfstate.d**
- In practice individuals or very small teams will have been known to commit these files to their repositories.

Remote State

- The workspace files are stored directly in the configured backend.

You can reference the current workspace name via **terraform.workspace**

Multiple Workspaces

A Terraform configuration has a backend that:

- defines **how operations are executed**
- where **persistent data is stored** e.g. Terraform State

Multiple workspaces are currently supported by the following backends

- AzureRM
- Consul
- COS
- GCS
- Kubernetes
- Local
- Manta
- Postgres
- Remote
- S3

Certain backends support *multiple* named workspaces

- allowing multiple states to be associated with a single configuration.

Terraform Cloud provides a way to **connect your workspace to one or more workspaces** via **Run Triggers** within your organization, known as "source workspaces".

Terraform Associate CheatSheet

Exam Pro

Run Triggers

- allow runs to queue automatically in your workspace on successful apply of runs in any of the source workspaces.
- You can connect each workspace to up to 20 source workspaces.

terraform workspace

terraform workspace **list -** list all existing workspaces current workspace is indicated using an asterisk (*)
terraform workspace **show -** Show the current
terraform workspace **select -** Switch to target workspace
terraform workspace **new -** Create and **switch** to workspace
terraform workspace **delete -** Delete target workspace

Sentinel is an embedded **policy-as-code framework** integrated with the Terraform Platform

What is Policy as Code?

When you write code to automate regulatory or governance policies

Features of Sentinel:

Embedded - enable policy enforcement in the data path to actively reject violating behavior instead of passively detecting.

Fine-grained, condition-based policy - Make policy decisions based on the condition of other values

Multiple Enforcement Levels - Advisory, soft and hard-mandatory levels allow policy writers to warn on or reject behaviour

External Information - Source external information to make holistic policy decisions

Multi-Cloud Compatible - Ensure infrastructure changes are within business and regulatory policy across multiple providers

Benefits of Policy as Code

- **Sandboxing** – The ability to create guardrails to avoid dangerous actions or remove the need of manual verification
- **Codification** – The policies are well documented and exactly represent what is enforced
- **Version Control** – Easy to modify or iterate on policies, with a chain of history of changes over time
- **Testing** - syntax and behavior can be easily validated with Sentinel, ensuring policies are configured as expected
- **Automation** – policies existing as code allows you direct integrate policies in various systems to auto-remediate, notify.

Terraform Associate CheatSheet

Sentinel and Policy as Code

- **Language** - All Sentinel policies are written using the Sentinel language
  - Designed to be non-programmer and programmer friendly, embeddable and safe.
- **Development** - Sentinel provides a CLI for development and testing.
- **Testing** - Sentinel provides a test framework designed specifically for automation.

Packer is a developer tool to **provision a build image** that will be stored in a repository.

- Using a build image before you deploy provides:
- immutable infrastructure
- your VMs in your fleet are all one-to-one in configuration
- Faster deploys for multiple servers after each build
- Earlier detection and intervention of package changes or deprecation of old technology

Consul is a **service networking platform** which provides:

- **service discovery** – central registry for services in the network
  - Allows for direct communication, no single-point of failure via load balancers
- **service mesh** – managing network traffic between services
  - A communication layer on top of your container application, *think middleware*
- **application configuration capabilities**

What is HashiCorp Vault?

Vault is a tool for **securely accessing secrets** from multiple secrets data stores.

Vault is deployed to a server where:

- Vault Admins can directly manage secrets
- Operators (developers) can access secrets via an API

Gruntwork is a software company that builds DevOps tools **that extends or leverages Terraform**

Terraform Associate CheatSheet

Terragrunt is a thin wrapper for Terraform that provides extra tools for:

- keeping your configurations **DRY**
- working with multiple Terraform modules
- managing remote state.

TerraTest allows you to **perform Unit Test and Integration Tests** on your Infrastructure.

It tests your infrastructure by:

- temporarily deploying it
- validating the results
- then tearing down the test environment.