Terraform Notes

IAC Benefits – automation, versioning, reusability

stores a cache of the attribute values for all resources in the state, uses state to track resource metadata such as resource dependencies

Terraform and Terraform Plugins are written in the **Go** programming language

Each TF config has an associated backend that defines how operations are executed and where state is stored

Providers

* multiple providers w/ alias
* terraform providers command prints info about providers used in current configuration
* Provider configuration block is not mandatory for all terraform configuration
* Many cloud providers do NOT provide APIs to query multiple resources at once and also have API rate limiting – for large infrastructures this can affect latency w/plan and apply
* new provider can be added to a configuration -- either explicitly via a provider block **or** by adding a resource from that provider -- TF must initialize it before it can be used

Table

Description automatically generated Graphical user interface, text

Description automatically generated Graphical user interface, text, application

Description automatically generated

**Terraform does not recommend using the version argument in provider configurations**. In Terraform 0.13 and later, version should always be declared in required\_providers block:

The name = { source, version } syntax for required\_providers was added in **Terraform 0.13**

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application

Description automatically generated

* Multiple providers for multiple accounts

Text

Description automatically generated Graphical user interface, text, application, Teams

Description automatically generatedGraphical user interface, text

Description automatically generated

Dependencies:

Graphical user interface, text, application, email

Description automatically generated

Init

* plugins are downloaded in the sub-directory of the present working directory at the path of **.terraform/plugins**
* **NOTE:** In Terraform v0.12 terraform init command cannot automatically download [Third-party Plugins](https://www.terraform.io/docs/configuration/providers.html#third-party-plugins) providers that are not distributed by HashiCorp (need to be done manually), but in Terraform v0.13 terraform init command can do it automatically
* terraform init -**upgrade -get-plugins=false -plugin-dir=PATH**
* terraform init **-lock=false** (disable state file locking)
* **Plugin Installation**
* searches the configuration for both direct and indirect references to providers and attempts to load the required plugins.
* **Backend Initialization**
* root configuration directory is consulted for backend configuration and the chosen backend is initialized using the given configuration settings.
* **Child Module Installation**
* configuration is searched for module blocks, and the source code for referenced [modules](https://www.terraform.io/docs/modules/) is retrieved from the locations given in their source arguments

Plan

* reads from data sources during this phase
* not encrypted by TF
* Generated plan can be saved to a specific path
* This plan can be used with tf apply to be certain only changes shown in plan are applied
* **terraform plan -out=path**
* **-target=<resource>** flag for specific resource
* The behavior of any terraform destroy command can be previewed at any time with an equivalent **terraform plan -destroy** command
* The tilda ~ symbol means there was an update in-place:

Text

Description automatically generated

Apply

* applies config changes and updates terraform.tfstate
* does NOT import resources
* **-auto-approve** flag used so you don’t have to manually enter **yes**
* Terraform tracks resources by their name. If you change the name, you have *created a new resource* and *deleted the old resource*

Refresh

* modifies state file (use **terraform show** to see human readable output of state/plan)
* you can prevent tf from querying the current state during operations like plan which will reduce number of API calls
* **-refresh=false** flag

Destroy

1. can destroy infrastructure by commenting out resources (followed by terraform plan)
2. explicitly listing resource terraform destroy -target aws\_instance.myec2
3. simple terraform destroy in cli terraform destroy

fmt

validate - Requires initialized directory; validates config files are syntactically correct

Provisioners

* Not ideal – last resort
* You can run provisioners on a **null\_resource**
* Expressions in provisioner blocks cannot refer to their parent resource by name; Instead, they can use the special **self** object which represents the provisioner's parent resource, and has all of that resource's attributes
* all log output from the provisioner is automatically suppressed to prevent sensitive values from being displayed
* 2 types: local-exec, remote-exec; others types include chef, connection, file, **null\_resource**, habitat, salt-masterclass

Text

Description automatically generated

Graphical user interface, application, table

Description automatically generated Graphical user interface, text, application

Description automatically generated

\*destroy provisioner will NOT run on tainted resources

Local Exec Provisioners

* invoke local executable after resource is created on machine running tf NOT resource

Graphical user interface, text, application

Description automatically generated

Remote Exec Provisioners

* Allow to invoke scripts directly on remote **resource**
* supports ssh and WinRM type connections

Graphical user interface, text, application

Description automatically generated Text

Description automatically generated

**Packer** is HashiCorp's open-source tool for creating machine images from source configuration. You can configure Packer images with an operating system and software for your specific use-case.

Terraform configuration for a compute instance can use a Packer image to provision your instance without manual configuration. Recommended instead of provisioners.

Debugging

* **TF\_LOG** – TRACE, DEBUG, INFO, WARN, ERROR
* Detailed logs will appear on **stderr**
* To persist logged output set **TF\_LOG\_PATH**
* crash.log – when terraform crashes search for **panic:**

Table

Description automatically generated

Import

* Find resource ID and import into state at given ADDRESS
* Can only import one resource at a time
* Not all providers and resources support tf import
* Must write a resource configuration block manually for a manually created resource prior to running terraform import
* only limitation Terraform has when reading the configuration files is that the import provider configurations must not depend on non-variable inputs
* Tf does NOT automate creation of tf files only state
* current implementation of Terraform import can only import resources into the [**state**](https://www.terraform.io/docs/state) **and modules**
* terraform import aws\_instance.myec2 instance-id

Local Values

* assigns a name to an expression so it can be used **multiple** times w/in a module w/o repeating it
* The expression of a local value can refer to other locals, but as usual reference cycles are not allowed - a local cannot refer to itself or to a variable that refers (directly or indirectly) back to it
* in below example the resources on the right will inherit both parts of common\_tags

Diagram

Description automatically generated

Graphical user interface, application

Description automatically generated

Variables

* Input variables are created by a variable block, but you reference them as attributes on an object named var
* variables declared in the root module of your configuration can be set values using CLI options and environment variables
* if variable is empty it will ask you for input at the cli
* From Terraform v0.14 there is experimental support for marking particular attributes as optional in an object type constraint

Graphical user interface, text, application

Description automatically generated Text

Description automatically generated Graphical user interface, text, application

Description automatically generated

Resources

* **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

The following are meta-arguments:

* [depends\_on, for specifying hidden dependencies](https://www.terraform.io/docs/configuration/meta-arguments/depends_on.html)
* [count, for creating multiple resource instances according to a count](https://www.terraform.io/docs/configuration/meta-arguments/count.html)
* [for\_each, to create multiple instances according to a map, or set of strings](https://www.terraform.io/docs/configuration/meta-arguments/for_each.html)
* [provider, for selecting a non-default provider configuration](https://www.terraform.io/docs/configuration/meta-arguments/resource-provider.html)
* [lifecycle, for lifecycle customizations](https://www.terraform.io/docs/configuration/meta-arguments/lifecycle.html)
* [provisioner and connection, for taking extra actions after resource creation](https://www.terraform.io/docs/configuration/blocks/resources/provisioners/index.html)

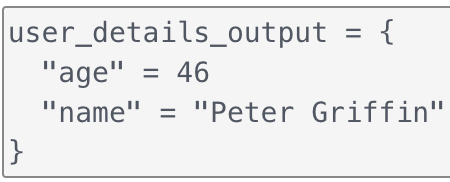
Data Types

* primitive types: string, number, bool
* complex types – collection and structural
* collection types: list, set, map
* structural types: object, tuple (allows multiple values of several distinct types to be grouped together as a single value)
* **any** can be used
* Slice Function is not part of the string function; join, split, chomp are part of it
* List is a sequence of values identified by consecutive whole numbers starting with zero where all elements are of the same type
* list, list(any), list(string)
* NOTE: If a module argument requires a value of type list(string) and a user provides the tuple ["a", 15, true], Terraform will internally transform the value to ["a", "15", "true"] by converting the elements to the required string element type as all of the elements of a collection must have the same type.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated output: 

Workspace

* VCS or non-VCS
* Name must be valid to use in URL path segment w/o escaping
* Equivalent to renaming state file
* There is always one DEFAULT workspace that can NOT be deleted
* contain tf configuration files, environment variables, Terraform variables, and state files — everything tf needs to manage a given collection of infrastructure
* tf maintains the tfstate file separately for EACH workspace (multiple files)
* For local state, Terraform stores the workspace states in a directory called terraform.tfstate.d. This directory should be treated similarly to local-only terraform.tfstate
* Workspaces are different between TF CLI and TF Cloud; TF Cloud has state version and run history

Text, letter

Description automatically generated 

A picture containing graphical user interface, text

Description automatically generated

NOTE: Certain backends support multiple named workspaces, allowing multiple states to be associated with a single configuration. The configuration still has only one backend, but multiple distinct instances of that configuration to be deployed without configuring a new backend or changing authentication credentials.

Organizations commonly want to create a strong separation between multiple deployments of the same infrastructure serving different development stages (e.g. staging vs. production) or different internal teams. In this case, the backend used for each deployment often belongs to that deployment, with different credentials and access controls. Named workspaces are *not* a suitable isolation mechanism for this scenario because each subsystem should have its own separate configuration and backend, and will thus have its own distinct set of workspaces.

Within your Terraform configuration, you may include the name of the current workspace using the ${terraform.workspace} interpolation sequence. This can be used anywhere interpolations are allowed. However, it should **not** be used in remote operations against Terraform Cloud workspaces.

The [${terraform.workspace}](https://www.terraform.io/docs/state/workspaces.html#current-workspace-interpolation) interpolation sequence should be removed from Terraform configurations that run remote operations against Terraform Cloud workspaces. The reason for this is that each Terraform Cloud workspace currently only uses the single default Terraform CLI workspace internally. In other words, if your Terraform configuration used ${terraform.workspace} to return dev or prod, remote runs in Terraform Cloud would always evaluate it as default regardless of which workspace you had set with the terraform workspace select command.

Modules

* <https://github.com/WillBrock/terraform-course-examples/blob/master/module-examples>
* Public or private
* **terraform get** (install model)
* All modules require a source argument; version is recommended but not required
* Version constraints are supported only for modules installed from a module registry (TF public or private)
* module installer supports installation from a number of different source types like Local paths, Terraform Registry, GitHub, S3 buckets, etc
* After adding, removing, or modifying module blocks, you must re-run terraform init
* It is not possible to taint an entire module. Instead, each resource within the module must be tainted separately
* the calling module **cannot** access child attributes directly. However, the child module can declare output values to selectively export certain values accessible by calling module
* When you declare variables in the root module of your configuration, you can set their values using CLI options and environment variables. When you declare them in [child modules](https://www.terraform.io/docs/configuration/modules.html), the calling module should pass values in the module block explicitly

Graphical user interface, text

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

* A module can NOT access all parent module variables; hence to pass variables to a child module, the calling module should pass specific values in the module block.
* A local path must begin with either ./ or ../ to indicate that a local path is intended
* Provider configurations can only be defined in root module
* each module must declare its own [provider requirements](https://www.terraform.io/docs/configuration/provider-requirements.html), so that Terraform can ensure that there is a single version of the provider that is compatible with all modules in the configuration and to specify the [source address](https://www.terraform.io/docs/configuration/provider-requirements.html#source-addresses) that serves as the global (module-agnostic) identifier for a provider
* Providers can be passed down to descendent modules in two ways: either *implicitly* through inheritance, or *explicitly* via the providers argument within a module block
* You call a module by using the “module” block; the variables you define in this block are passed to the child module
* With and Without Modules:

A picture containing graphical user interface

Description automatically generated

ROOT and Child Modules

Graphical user interface, text, application, email

Description automatically generated

Suppress Values in CLI Output

* still recorded in state

A picture containing graphical user interface

Description automatically generated

Terraform Registry (public and private – Private Module Registry)

* public – verified and community modules
* Repo of modules written by tf community (verified modules supported by 3rd parties with blue verification badge)
* PUBLIC: <NAMESPACE>/<NAME>/<PROVIDER>
* PRIVATE: **<HOSTNAME>**/<NAMESPACE>/<NAME>/<PROVIDER>

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application, email

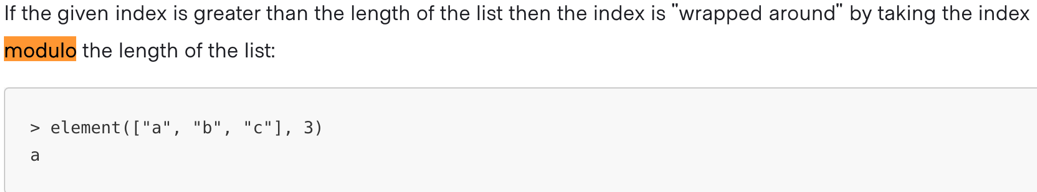
Description automatically generated

Reqs for publishing public module to Terraform Registry

* GitHub – must be a public repo on GitHub (does not apply for private modules)
* named terraform-<PROVIDER>-<NAME>; name can include additional dashes
* Repo Description
* Standard module structure
* Tags for releases – **semantic** version ie: v1.0.4

Functions

* built in functions (max, min, element, lookup); no user-defined functions allowed
* use command **terraform console** to test out these functions
* lookup(map, key, default)
* > lookup({a="ay", b="bee"}, "a", "what?")
* ay
* > lookup({a="ay", b="bee"}, "c", "what?")
* what?
* element(list, index)
* > element(["a", "b", "c"], 1)
* b



Graphical user interface, text, application, email

Description automatically generated

Count and Count Index

* the code snippet below will create 5 iam users; index starts at 0

Text

Description automatically generated

Splat Expression

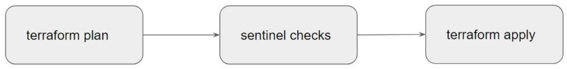
* Allows us to get a list of all the attributes
* The special [\*] symbol iterates over all of the elements of the list given to its left and accesses from each one the attribute name given on its right.

Text

Description automatically generated

Sentinel

* proactive policy as code (paid feature); policy sets contain policies
* You can also use the **[tfe\_sentinel\_policy](https://www.terraform.io/docs/providers/tfe/r/sentinel_policy.html)**resource from [Terraform Enterprise provider](https://www.terraform.io/docs/providers/tfe/) to upload a policy using Terraform itself
* supports native VCS integration and direct policy set uploads
* Soft mandatory requires org owner overriding failure before applying

 Graphical user interface, text, application, chat or text message

Description automatically generated

Graph

Diagram

Description automatically generated

Output Command

* Used to extract the output value of an output variable from the state file

Output Values

* TF can output the attribute of a resource with output values

Conditional Expressions

Graphical user interface, text

Description automatically generated

Variable Assignment

* terraform.tfvars overrides variable defaults in variables.tf file
* Environment variable format – **TF\_VAR\_<variable>**

Terraform loads variables in following order (lowest to highest precedence):

* Environment variables
* terraform.tfvars file, if present.
* terraform.tfvars.json file, if present.
* Any \*.auto.tfvars or \*.auto.tfvars.json files, processed in lexical order of their filenames.
* 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.

Data Sources

* Fetches data from resource for use elsewhere in TF configuration
* refreshing the data instance will be deferred until the **apply** phase
* Not recommended to use **depends\_on**
* support same [meta-arguments](https://www.terraform.io/docs/configuration/blocks/resources/syntax.html#meta-arguments) of resources with the exception of **lifecycle**

Graphical user interface

Description automatically generated

Load Order and Semantics

Graphical user interface, text, application

Description automatically generated

Dynamic Blocks (don’t overuse)

Graphical user interface, text

Description automatically generated A picture containing graphical user interface

Description automatically generated

Taint

* Taint command manually marks a TF managed resource as tainted forcing it to be destroyed and recreated on the next apply
* ie: terraform taint aws\_instance.ec2

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

DRY Principle - Don’t Repeat Yourself

State (JSON)

* necessary for TF to function
* Desired – what you have in your resources of tf files
* Current – what infrastructure is (include manual changes that may have been made)
* Always mention items like security groups in the resources of your tfstate file so they will be picked up if manual changes are made
* Resources removed from tf **state** will not be physically destroyed
* If a resource has an attribute updated manually that is not part of tf state then on terraform apply will update the state file but not touch the running instance (assuming no other attributes in the state file were manually updated – otherwise this would result in the resource destroyed and recreated)

Sensitive Data in State File

* Terraform Cloud encrypts state at rest and with TLS in transit

A picture containing text, screen, screenshot

Description automatically generated

Terraform Lock/Unlock

* can lock state for write operations but not all backends support this
* State locking happens automatically on all operations that could write state
* **force-unlock LOCK\_ID** [DIR] command
* S3 does not support state locking but you can use it with DynamoDb

Remote State Management and Git

* Remote state allows teams to share infrastructure resources in a read-only way without relying on any additional configuration store
* TF state can contain sensitive data, hence while using remote state, TF does not persist state to the local disk; instead, state is only ever held in memory when used by TF
* Remote Backend Types:
  1. **Standard** – state storage and locking (artifactory, azurerm, consul, gcs, http, s3, tf enterprise)
  2. **Enhanced** – all features of Standard plus remote management (local)

Table

Description automatically generated Graphical user interface, text, application

Description automatically generated

* Never modify state file directly – instead use **state** command
* rm does not destroy the resource, just no longer managed by tf
* All terraform state subcommands that modify the state write backup files. The path of these backup file can be controlled with -backup (can not be disabled)

Table

Description automatically generated

Credentials in Config - Use environment variables

Remote Backend for Terraform Cloud

* Backends are optional – can just use local default
* Only one backend may be specified and configuration **can not contain interpolations**
* Remote backend stores Terraform state; run operations may be used
* When using full remote operations, operations like tf plan or apply can be executed in TF Cloud’s run env with log output streaming to the local terminal

Diagram

Description automatically generated Graphical user interface, text, application

Description automatically generated

Partial Configuration

* Omitted arguments provided during init
* Ways to apply remaining args: interactively, file, CLI

Graphical user interface, text

Description automatically generated

Terraform Cloud

* permissions
* can configure VCS (GitHub) where you will store tf code
* When branch in a linked repo receives a PR from another branch in that repo, TF Cloud runs a [**speculative plan**](https://www.terraform.io/docs/cloud/run/index.html#speculative-plans) in every workspace linked to the destination branch
* knows identity of user requesting state and maintains history of state changes
* detailed audit logging
* workspaces retain backups of prev state files
* If your tf configuration will be used within Terraform Cloud, **only SSH key authentication** is supported (SSH is for non public repos too)

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

TF Cloud has three workflows for managing TF runs:

1. UI/VXS-driven – workspaces are associated with specific branch; runs automatically started when commit or merged
2. API-driven which is more flexible but requires you to create some tooling
3. CLI-driven

Terraform Enterprise

* offers enterprises a private instance of the Terraform Cloud application, with no resource limits and with additional enterprise-grade architectural features like:

1. Single Sign-On
2. Auditing
3. Private Data Center Networking
4. Clustering

Team Mgmt, Sentinel and Cost Estimation are also paid features

API and CLI access for Terraform Cloud can be managed through API tokens that can be generated from Terraform Cloud UI.

Difference 0.11 and 0.12

              “${var.instance\_type}”    → 0.11

             var.instance\_type            → 0.12

**air-gapped** - If terraform needs to be installed in an environment without internet access

**Zipmap function**

Terraform Versions

<https://www.hashicorp.com/blog/automatic-installation-of-third-party-providers-with-terraform-0-13>

CLI Environment Variables

* The flag TF\_CLI\_ARGS affects all Terraform commands. If you specify a named command in the form of TF\_CLI\_ARGS\_name then it will only affect that command. As an example, to specify that only plans never refresh, you can set TF\_CLI\_ARGS\_plan="-refresh=false"
* <https://www.terraform.io/docs/commands/environment-variables.html>

When you create a generic compute resource in Terraform, your virtual machine (VM) may not have much capability because it is a "fresh" install and needs to be provisioned with the software you want to use. Manually installing the necessary software and its respective dependencies on each VM is time consuming and difficult to maintain at scale.

[**cloud-init**](https://cloudinit.readthedocs.io/en/latest/) is a standard configuration support tool available on most Linux distributions and all major cloud providers. cloud-init allows you to pass a shell script to your instance that installs or configures the machine to your specifications.

**CLI Commands**

<https://www.terraform.io/docs/commands/>

Tutorials

<https://learn.hashicorp.com/terraform>

<https://www.youtube.com/playlist?list=PL8HowI-L-3_9bkocmR3JahQ4Y-Pbqs2Nt>

<https://learn.hashicorp.com/collections/terraform/certification>

<https://github.com/zealvora/terraform-beginner-to-advanced-resource>