Terraform exam Prep

Part 1

① Providers in AWS

→ provider is responsible for understanding API interactions & exposing resources

tf init → automatically download & save locally to a .terraform directory

- provider's with versions, region must be declared so that terraform can download them but provider block may be omitted if its content would otherwise be empty.

- provider block's - ? mandatory ? - NO

Alias in Providers

→ can be used for using same provider

with tf config for tf resources

→ save the credentials outside tf config like as env variables

→ tf settings are used to configure project specific tf behaviours, such as min tf version to apply to your config

→ in required_provider {..} mention source, version but can't specify region or access/secret key

→ terraform init command initializes working directory

→ safe to run multiple times

→ initialization includes provider plugins, backend initialization, etc.

→ terraform init -upgrade installs the

latest module of provider versions allowed within configured constraints

→ tf plan command is used to create an execution plan

→ tf plan does not modify the state file even after checking drift

→ to save a plan use the "-out=FILE" flag which ensures to have a consistent image

→ name of state file created during apply is → terraform.tfstate

→ tf destroy is not only command to destroy infra

→ tf fmt to format code
- check / -recursive

→ terraform validate → validates the config files in a directory

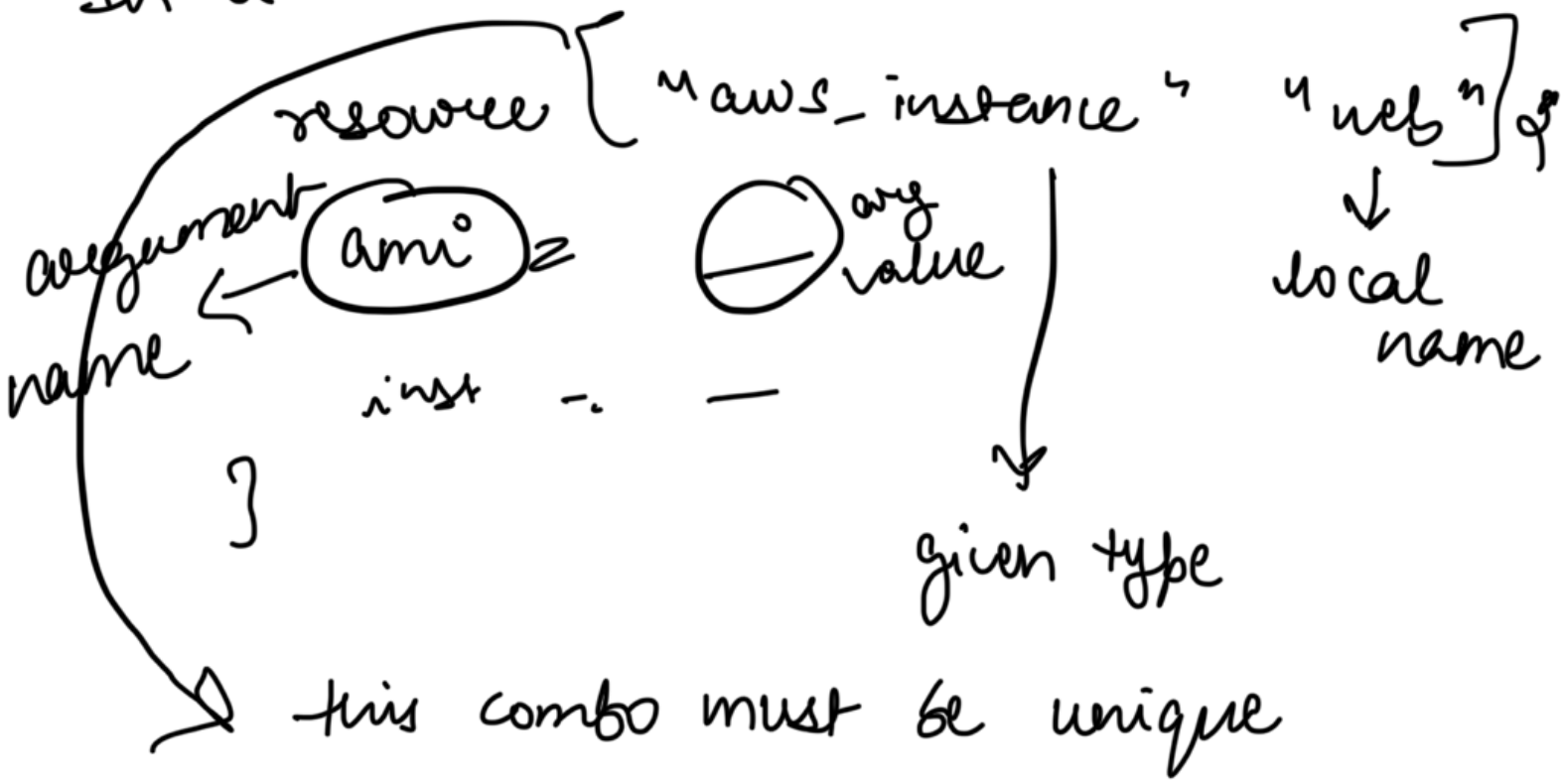
→ will not work without tf

→ terraform refresh → reads current settings from all managed remote objects & updates the tf state to match ↓

this will just modify tf state & not modify real remote objects

→ this is unsafe behaviour — deprecated

In a resource block :



data types in terraform :

string, number, bool, list, set, map, null

→ array not supported

→ terraform state command is used for advanced state management

tf state
sub commands

list → to list resources within tf state file

mv → moves item with tf state

pull → manually download tf o/p state from remote state

push → manually upload local state file to remote state

rm → remove items from tf state

show → show attributes of single resource in state

---

tfl logs can be enabled by setting TF_LOG env variable to any value

TRACE, DEBUG, INFO, WARN, ERROR

TF_LOG_PATH set to save logs

Allows importing existing infra to tf

Auto code generation for imported resources is supported

You can use import blocks to import more than one resource at a time

terraform workspace new <name>

" " select <name>

→ in the registry → a module address has syntax hostname/namespace/name/system

Optional : in default = registry.terraform.io

→ user defined func not supported in tf

→ Optional

→ Sentinel → embedded policy as code framework integrated with hashicorp enterprise products.

Sentinel is a proactive service

tf plan → sentinel checks → apply

→ terraform graph → refers to a visual representation of dependency relationships b/w resources defined in your tf config

→ O/P of tf graph is in DOT format, which can be easily converted to an img

→ terraform.tfvars file can be used to define value to all variables

→ terraform plan -var-file = "prod.tfvars"

variables → default val can be given
→ .tfvars

→ env vars

→ can set in cmd as well

tf plan -var "name = val"

Tf loads variables in following order with later sources taking precedance over earlier ones:

1. env variables
2. .tfvars if present
3. .tfvars.json "
4. any auto.tfvars
5. any -var & -var-file options in cli

some reserved keywords which can't be var names:

- count - depends_on - for_each
- lifecycle _ providers - source

→ terraform console command opens

interactive env to try func before using

→ Dependency lock file:

↓

→ allows us to lock specific version of provider

→ terraform.lock.hcl for tracking provider dependency

→ everytime that tf version of provider will be used tf to upgrade use tf init -upgrade

Dependencies:

i) implicit ⇒ tf can automatically find references of object & create an implicit ordering requirement b/w 2 resources

ii) explicit ⇒ Specifying a dependency is only necessary when a resource relies on some other resource's behaviour but doesn't

access any of that resource's data in its argument

→ use depends_on meta-argument

Extra features in tf enterprise:

- SSO
- auditing
- private data center n/w
- clustering

To recreate the resource, tf apply -replace will apply the resource again even if no config changes done to that resource

---

Splat expressions [*]

- allows us to get a list of attributes
- splat can't be used with for_each

Air Gapped env - n/w security measure employed to ensure that a secure computer n/w is physically isolated from

unsecured n/w such as public internet

Requirement for publishing a module in registry

core rep: github, named as terraform-<Provider>-<Name>

(5) Repo description

standard module structure

X.Y.2 tags for release

---

tf uses parallelism to create resources, 10 resource at a time

---

lec 134

---

(1) terraform validate can be used to validate config files in a directory & validates

syntax error

- will not access any remote state

(2) To deal with dfy dev values instead region we make use of data source block

(3) TF_LOG = TRACE to enable debug

(21) tf apply match the current state to desired state

- tf init downloads all necessary provider plugin

Medium corp has created infra using tf apply where is state file?

⇒ in terraform.tfstate.d directory

Provisioner block is defined inside resource block.

If a child module is called from ROOT module user can set the variable value. but you cannot override variable value if already defined in child

To import a manually created ec2 instance with id use : tf import aws_instance.<name> id

---

dynamic block allows you to generate a collection of resources based on a list of values

Useful for creating repetitive resources like sg rules

TF_<name> is a general env variable which may/maynot be used by tf

If TF_<name> is not explicitly defined as tf variable, setting env will have no impact

---

TF_VAR_<name> is used to interpret env variable as input

By default when you create a new WS, you are automatically switched to it

---

All tf supported backends DO NOT have the state locking functionality like (local) backend, S3, gcs, azurerm

---

Tf functions : max, min, element, join, concat, file

---

if output is defined in child & root module both, on tf apply only root value o/p will be displayed

---

Tf cloud will automatically run a tf plan operation whenever a new code change is commited to version control repo that's linked to WS

---

dynamic blocks allow constructing a set of nested config blocks

---

Tf requires each provider to be unique so if defining 2 provider of same type, use "alias"

If you want to install certain s/w packages to ec2 instances after creating an instance, make use of remote provisioners

dynamic blocks can be useful for managing complex firewall config in a more efficient & maintainable way

Remote backend overview

↓

(1) stores tf state & may be used to run operations in tf cloud

(2) tf cloud can also be used with local operations & only state is stored in cloud backend

(3) with remote backend, operations like tf plan or apply can be executed in tf cloud with log o/p streaming to local terminal

(4) you cannot have VCS/github connected to a WS if you are defining resources locally as well — "single source of truth" to be followed

---

tf taint → if want to delete a resource & recreate explicitly

---

To output something related to a

... already linked to the resource :

tflint expression :

aws_iam_user.lb[4].arn

If tf taint command is run for a resource, the resource will be re-created on next tf apply operation

---

To run a specific program locally after tf creates a resource

local-exec provisioner can be used to achieve this use-case

Not supported source type for tf module installer → FTP server

supported → github repo, tf registry, local path, generic git, mercurial repo

---

tf backend can be migrated anytime even if the resources are already created

---

~ symbol in tf plan means that a resource

will be updated in place

"connection block" allows setting credentials required while defining tf provisioners

---

Tf workspaces allow you to isolate infra into separate env → each WS has it's own state file

---

Tf WS feature in tf cloud is not same as Tf WS feature in open source version of tf

---

To write code in canonical format & style

⇓

tf fmt & not graph

. tfvars files & state files should not be committed to git

---

if "sensitive = true" in O/P → value will not show in tf apply O/P but will be saved in state file

remote exec provisioner requires a connection & supports both ssh & winrm

Vault provider allows you to read & write Hashicorp vault to terraform

Tf cloud always encrypts state at rest & protects it with TLS in transit

All API requests to Tf cloud must be authenticated with bearer token

In the `tf state rm` command for a specific resource → Tf will remove all records of that resource from tf state file so that it's no longer there to track
will this destroy the resource? ⇒ False / No

when `tf init` is run, source code for referenced modules is retrieved from specified location

Hashicorp `doesn't` verify all modules published in the

---

set has a collection of unique values that do not have any secondary identifier or ordering

---

tf state show "resource name" will show o/p

---

default value of variable & it's description in state file — it will only store actual value of variable

---

Tf refresh needs to be connected to cloud provider, credentials to modify state file

---

Tf cloud > Tf cli° 3 -> secure variable storage.

Revise remote state mgmt, Tf cloud, provisioners