**GETTING STARTED WITH TERRAFORM**

**Infrastructure as Code**

* provisioning through software; not a manual process; achieve consistency and predictable deployments
* defined in code; that is stored in source control
* **benefits:** *automated deployment, repeatable process, creates multiple consistent environments,* ***DRY “don’t repeat yourself”*** *reusable components*
* **DEPLOYING YOUR FIRST TERRAFORM CONFIGURATION**

**Terraform** is ***push-type model***

* Automation tool
* Single binary compiled from Go
* Declarative syntax
* HashiCorp Config language or JSON format; .**tf extensions**

**Components of TF:**

* **Executable** single binary file to run terraform; scripts and provisioning
* **Configuration Files** config u deploy will be contain on one or more terra files
* **Provider Plugins** how terraform talks with other services
* **State data** keeps track; contains current info

**Provisioner**

* use to model specific actions on local machine or remote
* **file** copy files or dir from the machine exec
* **local-exec** invokes local exec after resource is created
* **remote-exec** invokes a scripts on a remote resource after it is created

**terraform** will list out all the main commands; help

***Object Types:***

* *Providers – provides info u are using*
* *Resources – things u want to create in a target environment; u’ll be writing*
* *Data Sources – query info from a provider*

**Block** used to provision ur component; container for other content

**Provider Block**

* plugins for terraform to interact with remote systems
* requires their own configurations
* plugins were automatically downloadable when u run **terraform init**

**Resource Block <Meta Arguments:>**

* Use with any resource types
* **depends\_on** specifies hidden dependencies
* **count** create multiple resource instances
* **for\_each** create multiple instances according to a map
* **provider** selecting non-default provider
* **lifecycle** lifecycle customization
* **provisioner and connection** taking extra actions after resource creation

**Data Source Block**

* allow data to be fetched or computed for use
* supports the same meta-args of resource
* use to avoid putting static content on u scripts
* can be use in querying

**Variable block**

* serves as a parameter
* *arguments:*

**default** a def value makes the var option

**description** specifies input variable’s documentation

**validation**

**sensitive** limits tf ui when var used in config

**type** string, number, bool <these are **constraints>;** list,set,map, object and tuple <these are **constructors**>

**Output block**

* helps to output the specific variable that u wanted to see after the resource created

***YOU CAN SEE THIS ON terraform.io WEBSITE:***

**MODULE** containers for **multiple** resources that are **used together**

**EXPRESSION** refer to compute values within a config

***>>For expression*** creates a complex type value by transforming another complex in a **list**

***>>Splat expression*** more concise wat to express common operation

***>>Conditional expression*** uses value of bool exp to select one of two values

**FUNCTIONS** terraform language includes several built-in functions that u can call from within expression to transform and combine values

**STATE** use to map real world resources

**WORKSPACE** persistent data stored in the backend is belong to this; set using TF CLI

Terraform Object Reference

<resource\_type>.<name\_label>.<attribute>

**terraform init**

**terraform plan** will look at your current

**terraform plan -out <**filename>

**terraform apply** <filename> will just create or confirm the changes u created

**terraform destroy** if u want to destroy; no longer need the environment

* **USING INPUT AND OUTPUT VARIABLES**

Working Data with Data in Terraform

* **Input variables** use to pass information to terraform configuration

U can have variable with no arguments; but not preferrable

**var.<name\_label>** variable reference

**{type,description,senstive}**

* **Local values** computed values inside config that can be referenced; inside the block has **key value pair**

***Locals Reference***

**locals {common\_tags = }**

**local.<name\_label>{com}**

* **Output values** can be constructed from one or more elements; **how we get information out of terraform**

**output “name\_label”{value, description, sensitive}**

**sample variable at main.tf**

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

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence//will now prompt for the value; pass the value

***Data types in Terraform:***

* ***Primitive*** *basic; primitive, string, number, and Boolean*
* ***Collection*** *grouping of primitive of types; list, set and map*
* ***Structural*** *allow u to mix data types; object and tuple*

***Syntax Validation***

**terraform validate** command that make sure ur config is correct; verifying contents

but first run **terraform init**

***Supply Variable Values***

**terraform plan -var=<variable.name>=<what u wanted to supply;value>** this is so long way;

***we have shortcut:***

***New File*** *just create*

**<variable name> = <**its value>

***To your main:***

**export TF\_VAR\_<variable name>=<its value>; and all of values u want to export**

u need to run that on a terminal so that values will not be able to display and can be seen by other; u can now run:

**terraform plan** will check first more likely dry run

**terraform plan -out <ur *New File>*** only because the values were initialized earlier

**terraform apply** will now run the code

**terraform apply <*New File****>* update changes

**terraform destroy** delete resources

+ added/uploaded/create

* Delete resource

~ modifying

Graphical user interface

Description automatically generated with medium confidence

//1st is the one u launch in AWS; 2nd u created from terraform (terminal); check the files created for reference

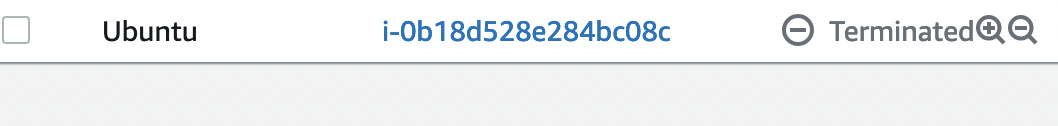
* **MODIFY RESOURCES**

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

Description automatically generated

//na-rename siya into “Ubuntu” using tags; check the file

* **DELETE RESOURCE (DESTROY)**

****

**terraform destroy**

* **REFERENCE RESOURCE**

**Graphical user interface, application

Description automatically generated**

* **TERRAFORM FILES**

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

Description automatically generated**

**.terraform** gets created whenever we initialized plugins

**terraform.tfstate** represents all state of terraform; all resources created are here; will update and keep track of our files

***PROJECT PRACTICE:***

***CHECK FILE Terraform > TerraProj1 > U CAN NOW SEE IN VSCODE ITS RULE AND SOME OF THE COMMANDS + NOTES***

* **TERRAFORM STATE COMMANDS**

**terraform state list** see more info abt list of resources; help for terraform

**terraform state show <one of the list u see>** want to know something specific; extract

* **TERRAFORM OUTPUT**

**U’ll add this in a specific resource u want to print out (*in ur own terraform file*); u can do this to any resources u wanted to automatically print out**

**Graphical user interface, text, application

Description automatically generated**

**output “server\_public\_ip” {**

**Value**

**}**

Automatically print out;

*After u terraform apply, u’ll now see the printed output*

***Text

Description automatically generated with medium confidence***

**terraform refresh** another way to update; or just refresh

**TARGET RESOURCES** u’ll just destroying specific resource u applied

***Command-line argument:***

**terraform apply -var “subnet\_prefix=10.0.100.0/24”** variable assignment; hindi kana pahihirapan using this command (not best practice) BUT TO HAVE SEPARATE FILE

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

Description automatically generated*

*//created new file: terraform.tfvars; like this, u’ll just save and apply*

**terraform apply -var-file terraform.tfvars** multiple application

BUT U CAN USE **default** if u don’t want to supply a value or have a separate file for variable

Text

Description automatically generated