**MODULE - 01 INTRODUCTION**

Use same project for deploy on different cloud providers (resources on AWS and Azure).

Plan before applying changes (what if) – not possible with AWS Cloudformation.

**MODULE - 01 CONFIGURING TERRAFORM WITH AWS**

Create AWS account (Free tier).

Create TerraformUser on IAM Users with full access.

Install Terraform (<https://www.terraform.io/>) and configure System variables.

Create System Variable for AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY.

**MODULE - 02 CREATING AN AWS RESOURCE WITH TERRAFORM**

Code samples: <https://github.com/kevholditch/terraform-course-examples>

Sample code to create S3 bucket

Go to sample code folder (main.tf file) and Terraform init (Example01)

Terraform plan

Terraform apply

Created S3 bucket rodolfomarra-myfirst-bucket on us-east-1

Terraform destroy

Deleted S3 bucket rodolfomarra-myfirst-bucket on us-east-1

**MODULE – 02 TERRAFORM RESOURCES**

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

Resources have arguments: required and optional

Resources could have Exported Attributes (ie. [ARN returned after having created S3 bucket](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket#attributes-reference))

Interpolation Syntax: replace information generated after creating AWS object (ie. pass ARN from S3 Bucket to IAM Role).

Interpolation syntax format:

“${<resource\_type>.<resource\_name>.<exported\_attribute>}”

Example:

**resource** “aws\_s3\_bucket” “my\_bucket” {

bucket = “rodolfomarra-myfirst-bucket”

}

Reference using: “$(aws\_s3\_bucket.my\_bucket.arn}”

Data types for resources attributes:

Int – defined using – port = 21

String – defined using – host = “localhost”

List – defined using – security\_groups = [“abc”,”def”]

Bool – defined using – enabled = false

Sample code to create S3 bucket and IAM Policy to list Bucket (Example02)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply

Created S3 bucket rodolfomarra-myfirst-bucket on us-east-1

Created IAM Policy my-bucket-policy on us-east-1

Terraform destroy

Deleted S3 bucket rodolfomarra-myfirst-bucket on us-east-1

Deleted IAM Policy my-bucket-policy on us-east-1

**MODULE – 03 TERRAFORM DATA SOURCES**

Data sources allow to reference objects from external Terraform projects or to resources already created on AWS

This will return many attributes from AWS resources to be reused on TF. To reuse attributes, you will need to use:

${**data**.<resource\_type>.<resource\_name>.<exported\_attribute>}

Example:

**data** “aws\_s3\_bucket” “my\_bucket” {

bucket = “rodolfomarra-myfirst-bucket”

}

Reference using: “$(**data**.aws\_s3\_bucket.my\_bucket.arn}”

Created S3 bucket rodolfomarra-already-here in us-east-1 **manually**

Sample code to reference S3 bucket that already exist and create IAM Policy to list Bucket (Example03)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply

Created IAM Policy my-bucket-policy on us-east-1

Terraform destroy

Deleted IAM Policy my-bucket-policy on us-east-1

Deleted S3 bucket rodolfomarra-already-here on us-east-1 **manually**

**MODULE – 04 TERRAFORM LOCALS**

Locals allow to assign a name to an expression (like variable)

Example of single local:

locals {

bucket\_name\_prefix = “rody-”

default\_instance\_tag = “my-instance”

}

Example of multiples locals:

locals {

bucket\_name\_prefix = “rody-”

}

locals {

default\_instance\_tag = “my-instance”

}

To reference locals you need to use the interpolation syntax:

“${local.<variable\_name>}”

Example:

locals {

bucket\_name\_prefix = “rody-”

default\_instance\_tag = “my-instance”

}

To reference bucket\_name\_prefix, you will need to use: “${**local**.bucket\_name\_prefix}”

Locals values can be combined to make more local values, example:

locals {

first = “rody”

last = “marra”

name = “$(local.first}-${local.last}”

}

The result of local.name will be “rody-marra”.

Locals can be a value of exported attribute, example:

resource “aws\_s3\_bucket” “my\_bucket” {

bucket = “rodolfomarra-myfirst-bucket”

}

locals {

bucket\_arn = “${aws\_s3\_bucket.my\_bucket.arn}”

}

Sample code to create S3 bucket using locals as S3 bucket name prefix

Go to sample code folder (main.tf file) and Terraform init (Example04)

Terraform apply

Created S3 bucket rodolfomarra-myfirst-bucket on us-east-1

Modified main.tf changing local used to S3 bucket name prefix (from rodolfomarra to rodolfosoares)

Terraform apply

Deleted S3 bucket rodolfomarra-myfirst-bucket on us-east-1

Created S3 bucket rodolfosoares-myfirst-bucket on us-east-1

Terraform destroy

Deleted S3 bucket rodolfosoares-myfirst-bucket on us-east-1

**MODULE – 05 TERRAFORM OUTPUTS**

Tell TF which values are important so TF can output them on screen when we run “Terraform apply”

Example:

output “my\_value” {

value = “hello kevin”

}

Will give the output: my\_value = hello kevin

Output can be result of expressions:

output “my\_value” {

value = “${aws\_s3\_bucket.my\_bucket.arn}”

}

Output can be values of locals:

output “my\_value” {

value = “${local.bucket\_name}”

}

Outputs can be used to return values from a module

Sample code to print on screen a value of local (Example05)

**MODULE – 06 TERRAFORM TEMPLATES AND FILES**

Terraform allow to use a file as a parameter to a resource (ie. a block of JSON for IAM Policy)

To use files, you will need to use: “${file(“<path\_to\_file>”)}”

Example:

resource “aws\_iam\_user\_policy” “my\_bucket\_policy” {

name = “my-bucket-policy”

user = “rodolfo-marra”

policy = “${file(“policy.json”)}”

}

<https://registry.terraform.io/providers/hashicorp/template/latest/docs>

To replace some value (ie."${bucket\_arn}") inside some TF template files (ie. policy.json) you need to use:

data "template\_file" "bucket\_policy" {

template = "${file("**policy.json**")}"

vars {

**bucket\_arn = "${aws\_s3\_bucket.my\_bucket.arn}"**

}

}

Every line on **policy.json** that is **"${bucket\_arn}"** will be replaced by S3 Bucket ARN.

To get and use the result of this replacement, you need to use: “${**data.template\_file.**<name>.**rendered**}”

Example:

“${**data**.**template\_file**.**bucket\_policy**.rendered}”

Sample code to create S3 bucket, IAM User and IAM Policy using external JSON template rendered with S3 bucket ARN (Example06)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply

Created S3 bucket rodolfomarra-bucket on us-east-1

Created IAM user Rodolfo-Marra on us-east-1

Created IAM User Policy my-policy on us-east-1

Terraform destroy

Deleted S3 bucket rodolfomarra-bucket on us-east-1

Deleted IAM user Rodolfo-Marra on us-east-1

Deleted IAM User Policy my-policy on us-east-1

**MODULE – 07 TERRAFORM PROVIDERS**

Terraform provider enables TF to talk to an API to manage resources (ie. aws, azure, googlecloud, etc).

Allow to manage resources on multiples clouds (or multiple regions of same cloud provider) in the single project.

You can specify the region and also access\_key and secret\_key when you define a provider (but is not recommended because it will be available on code), example:

provider "aws" {

region = "ca-central-1"

alias = "canada"

access\_key = “AAAAA”

secret\_key = “aASdasg”

}

We can pin a provider to certain version or add version requirements for a provider, example:

provider "aws" {

region = "ca-central-1"

alias = "canada"

version = “1.8”

}

provider "aws" {

region = "ca-central-1"

alias = "canada"

version = “~> 1.8”

}

<https://www.terraform.io/docs/configuration/providers.html>

If we don’t specify a provider on resource creation, default provider will be used

To specify another provider for specific resources, need to use: <provider>.<alias>

resource aws\_s3\_bucket "canada\_bucket" {

bucket = "rodolfomarra-canada"

**provider** = "**aws**.**canada**"

}

Sample code to create S3 bucket using default provider and another provider (Example07)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply

Created S3 bucket rodolfomarra-us-bucket on us-east-1

Created S3 bucket rodolfomarra-ca-bucket on ca-central-1

Terraform destroy

Deleted S3 bucket rodolfomarra-us-bucket on us-east-1

Deleted S3 bucket rodolfomarra-ca-bucket on ca-central-1

**MODULE – 08 TERRAFORM VARIABLES**

Variables serve as parameters to a TF module

When used at top level they enable to pass parameters into TF project

Properties of variables:

type (optional) – string, list or map

default (optional) – Default value

description (optional) – Variable description

Default variables need to be a String (ie. Foo) or List (ie. [“a”,”b”]). Can’t be interpolation syntax.

Examples of variable:

variable “**key**” {

type = “**string**”

}

To use this string variable, you need to: “{$var. **key**}”

variable “**instance\_size\_map**” {

type = “**map**”

default = {

**dev** = “t2.micro”

**test** = “t2.medium”

**prod** = “m4.large”

}

}

To use this map variable, you need to: “${lookup(var.**instance\_size\_map**,**dev**)}”

variable “**zones**” {

type = “**list**”

default = [“us-east-1a”,”us-east-1b”]

}

Usage: sometimes we cannot store secret, password, access\_key externally from source codes. So for it, we use external variables with .tfvars files.

To set the value of a variable, you can use:

Command line:

variable “**my\_name**” {}

terraform apply --var **my\_name**=kevin

Environment variable:

variable “**my\_name**” {}

env **TF\_VAR\_my\_name**=kevin terraform apply

Using a file:

Create .tfvars file with “variable = value”

Sample code to create SQS queue using environment variable (Example08)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply --var env\_type=dev --var queue\_name=myqueue

Created SQS Queue myqueue on us-east-1 tagged with dev-queue

Terraform destroy

Deleted SQS Queue myqueue on us-east-1 tagged with dev-queue

On cmd, ran:

SET TF\_VAR\_env\_type=test

SET TF\_VAR\_queue\_name=secondqueue

Terraform apply

Created SQS Queue secondqueue on us-east-1 tagged with test-queue

Terraform destroy

Deleted SQS Queue secondqueue on us-east-1 tagged with test-queue

Go to sample code folder and created **variables.auto.tfvars** file (Example09)

Terraform apply

Created SQS Queue from\_file on us-east-1 tagged with prod-queue

Terraform destroy

Deleted SQS Queue from\_file on us-east-1 tagged with prod-queue

Terraform loads variables in the following order, **with later sources taking precedence over earlier ones**:

1. **Environment** variables
2. The **terraform.tfvars** file, if present.
3. The **terraform.tfvars.json** file, if present.
4. **Any \*.auto.tfvars** or **\*.auto.tfvars.json** files, processed in lexical order of their filenames.
5. Any **-var** and **-var-file** options on the command line, in the order they are provided. (This includes variables set by a Terraform Cloud workspace.)

**MODULE – 02 PROJECT LAYOUT**

Terraform uses .tf that is located on root folder and ignore all other files (ie. txt) or .tf files that is stored on subfolders

All .tf files that is stored on root folders are joint together as one single .tf file when we run terraform apply

Sample code to create SQS Queue using more than one .tf files

Go to sample code folder (aws.tf file) and Terraform init (Example10)

Terraform apply

Created SQS Queue rody\_queue on us-east-1

Terraform destroy

Deleted SQS Queue rody\_queue on us-east-1

**MODULE – 03 TERRAFORM CUSTOM MODULES**

Template that package some useful configurations (ie. Configs reused to create EC2 instances ou SQS queues)

Standard layout:

main.tf – terraform configuration (resources, data sources, etc)

variables.tf – define variables (inputs to module)

output.tf – define outputs (returned by module)

To create a module an identify module folder, we need to declare module and fill source path, for example:

**module** “queue1” {

**source = “./queue”**

queue\_name = “my\_queue”

}

As we cannot use files on subfolders because TF are assumed to be on project root folder, when using module we can reference files on module folder using: ${path.module} – for example:

resource “aws\_iam\_user\_policy” “my\_bucket\_policy” {

name = “my-bucket-policy”

user = “Rodolfo-Marra”

policy = “${file(**“${path.module}”**/policy.json”)}”

To reference an output from a module, we need to use: ${**module**.<module\_identifier>.<output\_name>}

module “queue1” {

source = “./queue”

queue\_name = “my\_queue”

}

“${**module**.**queue1**.**queue\_arn**}”

Sample code to create two S3 bucket using custom module (Example11)

Go to sample code folder (main.tf file) and Terraform init

Terraform apply

Created S3 bucket rodolfomarra-normal on us-east-1

Created S3 bucket rodolfomarra-versioned on us-east-1

Terraform destroy

Deleted S3 bucket rodolfomarra-normal on us-east-1

Deleted S3 bucket rodolfomarra-versioned on us-east-1

**MODULE – 04 TERRAFORM REGISTRY MODULES**

Terraform registry modules is a module that someone else has written and uploaded to the Terraform registry (community)

<https://registry.terraform.io/>

To use modules registered by another person, you need to point the source and fill the required inputs:

module “user\_queue” {

**source = “terraform-aws-modules/sqs/aws”**

name = “user”

tags = {

Service = “user”

Environment = “dev”

}

}

**MODULE – 02 TERRAFORM PLANS**

Terraform apply run in multistage: first does a plan and give the chance to review plan and apply

Terraform plan: does a plan only

Terraform apply --auto-approve (apply without run a plan – not recommended)

Terraform actions:

Create (new resource) – identified by “+”

Change (change resource without destroy) – identified by “~”

Change with force recreate (destroy then create) – identified by “-/+”

Destroy (destroy resource) – identified by “-“

Sample code to create one S3 bucket

Go to sample code folder (main.tf file) and Terraform init

Terraform plan

Terraform apply

Created S3 bucket rodolfomarra on us-east-1

Terraform destroy

Deleted S3 bucket rodolfomarra on us-east-1

**MODULE – 03 TERRAFORM REMOTE STATE**

Keep track of infra it has created in a state file

By default terraform store state locally in file “terraform.tfstate”

To view terraform state need to run: terraform state list

We can have remote state to be able to share with team. To configure remote state, we need:

terraform {

backend “s3”{

bucket = “terraform-state-bucket”

key = “project.state”

region = “us-east-1”

}

}

Sample code to create one S3 bucket (example12a and example12b)

Created S3 Bucket “rodolfomarra-terraform-state” on us-east-1 **manually**

Go to sample code folder (Example12a - main.tf and state.tf file) and Terraform init

Terraform apply

Created “rodolfomarra-example” on us-east-1 and stored tfstate file on S3 “rodolfomarra-terraform-state”

Go to sample code folder (Example12b - main.tf and state.tf file) and Terraform init

Terraform destroy

Deleted “rodolfomarra-example” on us-east-1 and stored tfstate file on S3 “rodolfomarra-terraform-state”

Deleted “rodolfomarra-terraform-state” on us-east-1 **manually**

**MODULE – 04 MANAGING TERRAFORM STATE**

We can add/import and remove resources from TF state. That means TF will manage or not manage these resources

Steps to add an existing resource into TF project

Create a resource in TF project

Import resource into TF state

Run TF plan to check if TF doesn’t want to change the resource

In general, the cmd to import resource is: terraform import <resource\_path>.<id>

To remove resource is: terraform state rm <resource\_path>.<id>

Created SQS queue “rody\_queue” on us-east-1 **manually**

Go to sample code folder (Example13 - main.tf) and Terraform init

Terraform plan

Add SQS queue

Terraform state list

No state file found

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sqs_queue#import>

Terraform import aws\_sqs\_queue.queue <https://sqs.us-east-1.amazonaws.com/667084926673/rody_queue>

Terraform plan

No changes

Terraform state rm aws\_sqs\_queue.queue

Removed aws\_sqs\_queue.queue

Deleted SQS queue “rody\_queue” on us-east-1 **manually**

**MODULE – 02 TERRAFORM WORKSPACES**

Create multiple instances a single project using WF workspaces (dev, staging and production)

You need to be using one of state backends:

AzureRM

Consul

GCS

Local (default)

Manta

S3

To create a new TF workspace: terraform workspace new <name>

To list TF workspace: terraform workspace list

To switch TF workspace: terraform workspace select <name>

To delete TF workspace: terraform workspace delete <name>

To reference to workspace name, need to use: ${terraform.workspace}

resource "aws\_instance" "web" {

ami = data.aws\_ami.image.id

instance\_type = "t2.micro"

tags = {

Name = "WebServer**-${terraform.workspace}**"

}

}

Go to sample code folder (Example14 - main.tf) and Terraform init

Terraform apply

Created EC2 instance using AMI "ami-00eb20669e0990cb4" with type “t2.micro” tagged “**WebServer-default**”

Created EBS Block device

Created EBS Root Block device

Created Network interface

Terraform workspace list

\* default

Terraform workspace new qa

Terraform workspace list

default

\* qa

Terraform apply

Created EC2 instance using AMI "ami-00eb20669e0990cb4" with type “t2.micro” tagged “**WebServer-qa**”

Created EBS Block device

Created EBS Root Block device

Created Network interface

Terraform workspace select qa

Terraform destroy

Deleted EC2 instance using AMI "ami-00eb20669e0990cb4" with type “t2.micro” tagged “**WebServer-qa**”

Deleted EBS Block device

Deleted EBS Root Block device

Deleted Network interface

Terraform workspace select default

Terraform workspace delete qa

Terraform destroy

Deleted EC2 instance using AMI "ami-00eb20669e0990cb4" with type “t2.micro” tagged “**WebServer-default**”

Deleted EBS Block device

Deleted EBS Root Block device

Deleted Network interface

**MODULE – 03 RESOURCE META PARAMETERS**

All TF resources have 4 parameters defined:

count

depends\_on

provider

lifecycle

Count: tell TF how many of resource to create. Example:

resource “aws\_instance” “web” {

ami = “data.aws\_ami.image.id

instance\_type = “t2.micro”

**count = 3**

}

There are some resources that count doesn’t works (ie. S3 bucket – because the name is unique). To get around, need to use **${count.index}** on unique parameters. For example:

resource “aws\_s3\_bucket” “bucket” {

bucket = “rodolfomarra**-${count.index}**”

**count = 2**

}

You can specify count = 0 to not create a resource under certain conditions (ie. don’t create db backup for dev environment).

Depends\_on: allows to specify which resource depends on another resource.

resource “aws\_s3\_bucket” “bucket2” {

bucket = “rodolfomarra-depends”

**depends\_on = [“aws\_s3\_bucket.bucket”]**

}

When you use interpolation syntax, TF will create dependencies automatically.

Provider: allows to specify which cloud provider will be used to create resources.

provider “aws” {

region = “ca-central-1”

**alias = “canada”**

}

resource “aws\_s3\_bucket” “bucket” {

bucket = “rodolfomarra-bucket”

**provider = “aws.canada”**

}

Lifecycle: change the way TF updates, destroys, or changes resource.

create\_before\_destroy – create new DNS record before removing old one

prevent\_destroy – extra safety guard to prevent destruction of resource

ignore\_changes -

Go to sample code folder (Example15 - main.tf) and Terraform init

Terraform apply

Created S3 bucket “rodolfomarra-1” in “us-east-1”

Created S3 bucket “rodolfomarra-2” in “us-east-1”

Created S3 bucket “rodolfomarra-3” in “us-east-1”

Created S3 bucket “rodolfomarra-next-bucket” in “us-east-1”

Terraform destroy

Deleted S3 bucket “rodolfomarra-1” in “us-east-1”

Deleted S3 bucket “rodolfomarra-2” in “us-east-1”

Deleted S3 bucket “rodolfomarra-3” in “us-east-1”

Deleted S3 bucket “rodolfomarra-next-bucket” in “us-east-1”