# 

# **Read, Generate, Modify Configurations**

## **Module 1: Attributes and Output Values**

Terraform has the capability to output the attribute of a resource with the output values.

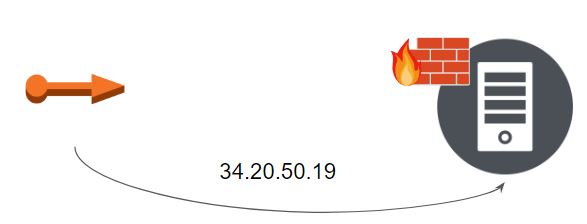
Example:

* ec2\_public\_ip = 35.161.21.197
* bucket\_identifier = terraform-test-kplabs.s3.amazonaws.com

An outputed attributes can not only be used for the user reference but it can also act as an input to other resources being created via terraform

Let’s understand this with an example:

After EIP gets created, it’s IP address should automatically get whitelisted in the security group.



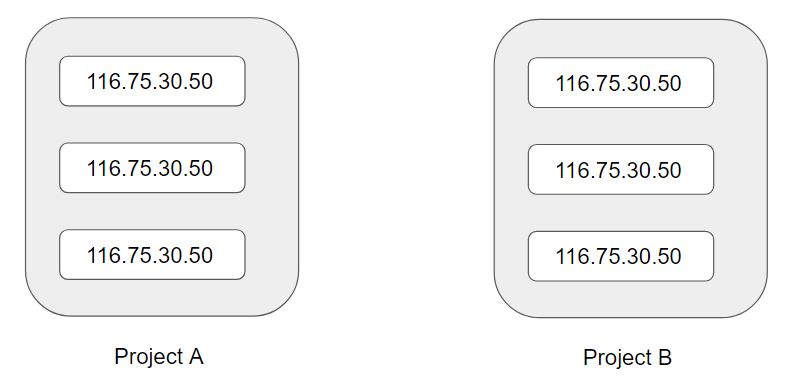
## 

## 

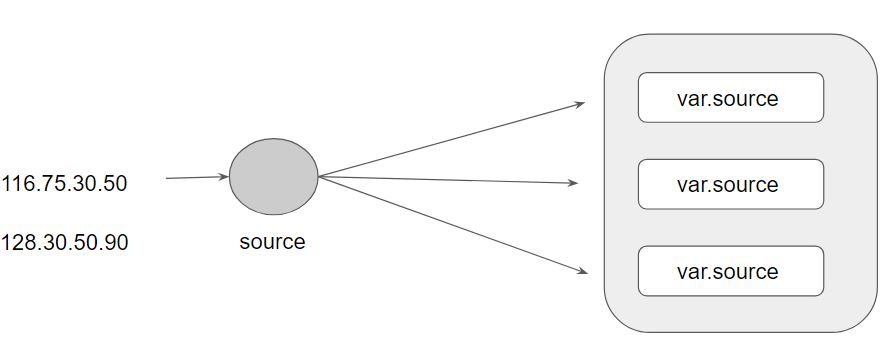
## 

## **Module 2: Terraform variables**

Repeated static values can create more work in the future.



Terraform Variables allows us to centrally define the values that can be used in multiple terraform configuration blocks.



## 

## 

## 

## 

## **Module 3: Approach for Variable Assignment**

Variables in Terraform can be assigned values in multiple ways.

Some of these include:

* Environment variables
* Command Line Flags
* From a File
* Variable Defaults

Sample Commands for the following:

i) Environment Variables:

export TF\_VAR\_instancetype="t2.nano"

echo $TF\_VAR

ii) Command Line Flags:

terraform plan -var="instancetype=t2.small"

terraform plan -var-file="custom.tfvars"

iii) From a File (terraform.tfvars):

instancetype="t2.large"

iv) Variable Defaults:

variable "instancetype" {

default = "t2.micro"

}

## 

## **Module 4: Data Types for Variables**

4.1 Overview of Type Constraints

The type argument in a variable block allows you to restrict the type of value that will be accepted as the value for a variable

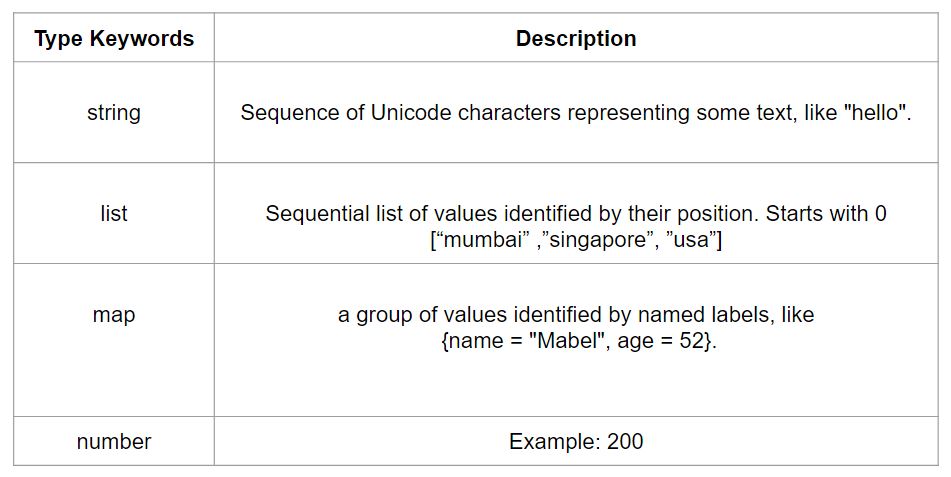
variable "image\_id" {

type = string

}

If no type constraint is set then a value of any type is accepted.

4.2 Overview of Data Types



## **Module 5: Count Parameter**

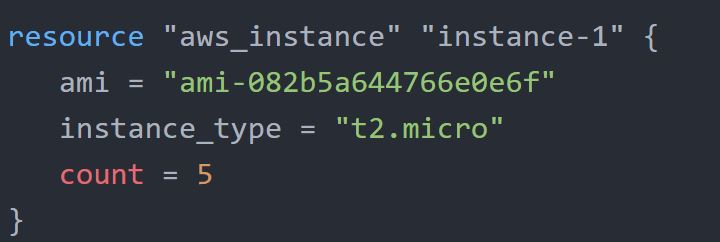
5,1 Overview of Count:

The count parameter on resources can simplify configurations and let you scale resources by simply incrementing a number.

Let’s assume, you need to create two EC2 instances. One of the common approaches is to define two separate resource blocks for aws\_instance.



With the count parameter, we can simply specify the count value and the resource can be scaled accordingly.



5,2 Count Index

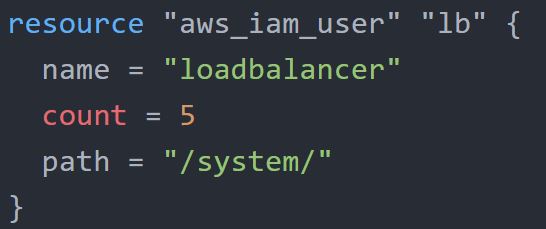
In resource blocks where the count is set, an additional count object is available in expressions, so you can modify the configuration of each instance.

This object has one attribute:

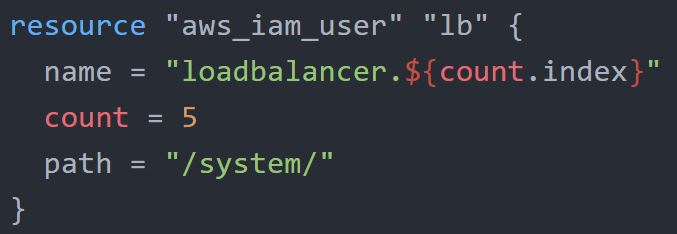
count.index — The distinct index number (starting with 0) corresponding to this instance.

5.3 Challenges with Count Parameter

With the below code, terraform will create 5 IAM users. But the problem is that all will have the same name.



count.index allows us to fetch the index of each iteration in the loop.

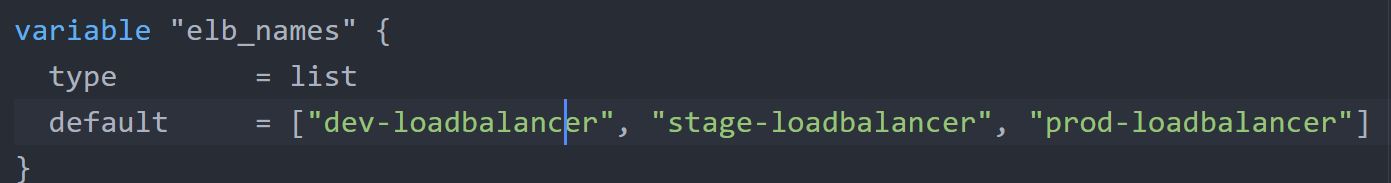


Understanding Challenge with Default Count Index

Having a username like loadbalancer0, loadbalancer1 might not always be suitable.

Better names like dev-loadbalancer, stage-loadbalancer, prod-loadbalancer is better.

count.index can help in such a scenario as well.



## **Module 6: Conditional Expression**

A conditional expression uses the value of a bool expression to select one of two values.

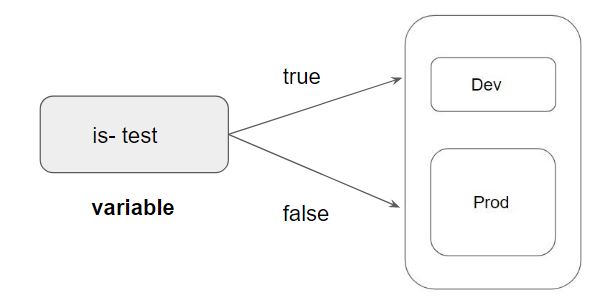
Syntax of Conditional expression:

condition ? true\_val : false\_val

If the condition is true then the result is true\_val. If the condition is false then the result is false\_val.

Let’s assume that there are two resource blocks as part of terraform configuration.

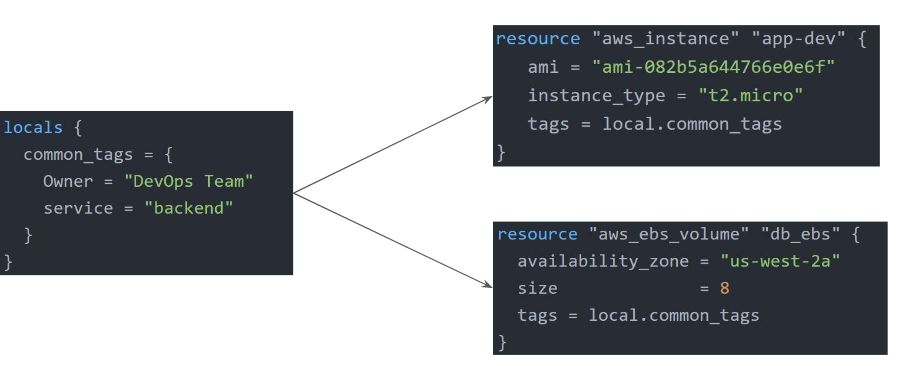
Depending on the variable value, one of the resource blocks will run.



## 

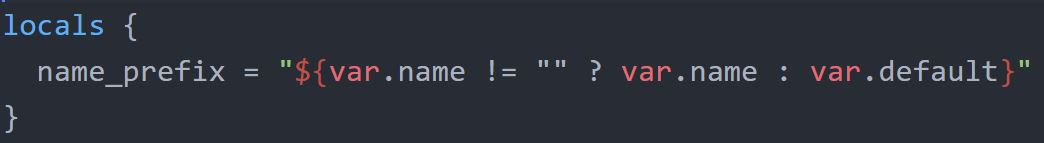
## **Module 7: Local Values**

A local value assigns a name to an expression, allowing it to be used multiple times within a module without repeating it.



Local Values Support for Expression

Local Values can be used for multiple different use-cases like having a conditional expression.



Important Pointers for Local Values:

Local values can be helpful to avoid repeating the same values or expressions multiple times in a configuration.

If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used

Use local values only in moderation, in situations where a single value or result is used in many places and that value is likely to be changed in the future.

## **Module 8: Terraform Functions**

The Terraform language includes a number of built-in functions that you can use to transform and combine values.

The general syntax for function calls is a function name followed by comma-separated arguments in parentheses:

function (argument1, argument2)

Example:

> max(5, 12, 9)

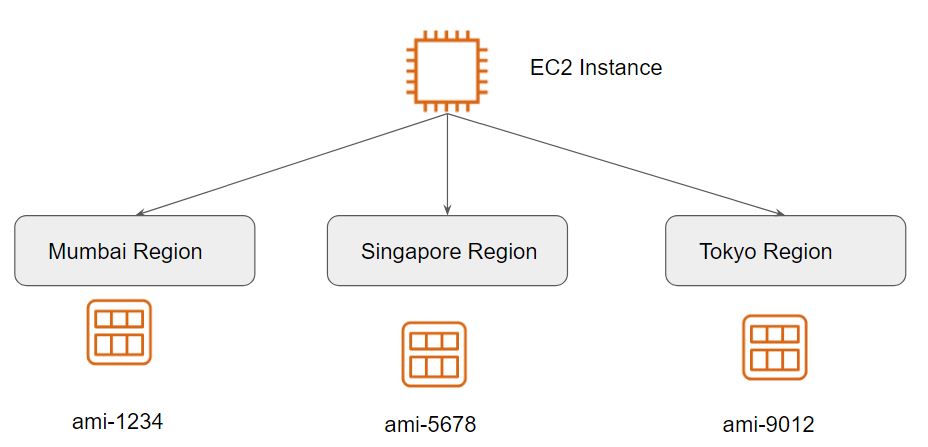
12

The Terraform language does not support user-defined functions, and so only the functions built into the language are available for use

* Numeric
* String
* Collection
* Encoding
* Filesystem
* Date and Time
* Hash and Crypto
* IP Network
* Type Conversion

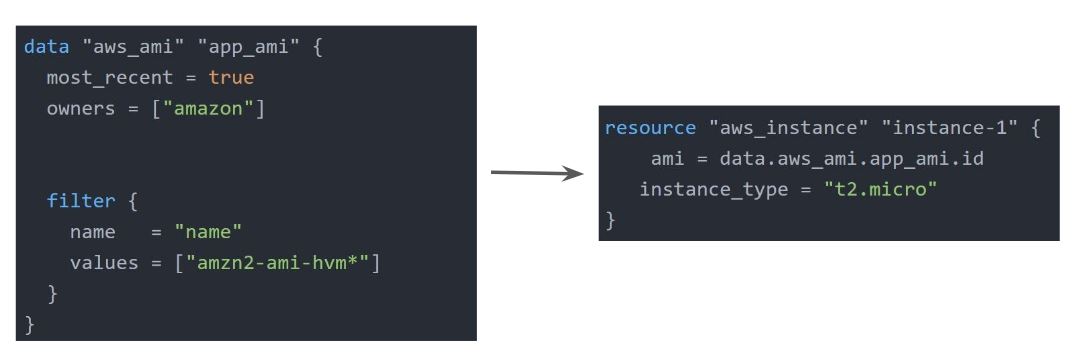
## **Module 9: Data Sources**

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.



A data source is defined under the data block.

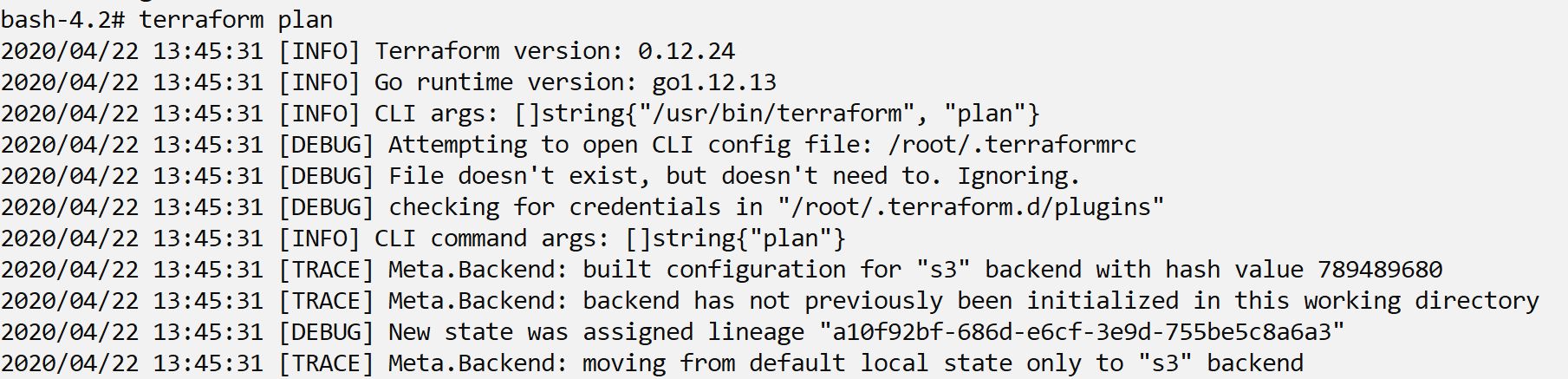
It reads from a specific data source (aws\_ami) and exports results under “app\_ami”



## **Module 10: Debugging in Terraform**

Terraform has detailed logs that can be enabled by setting the TF\_LOG environment variable to any value.

You can set TF\_LOG to one of the log levels TRACE, DEBUG, INFO, WARN or ERROR to change the verbosity of the logs



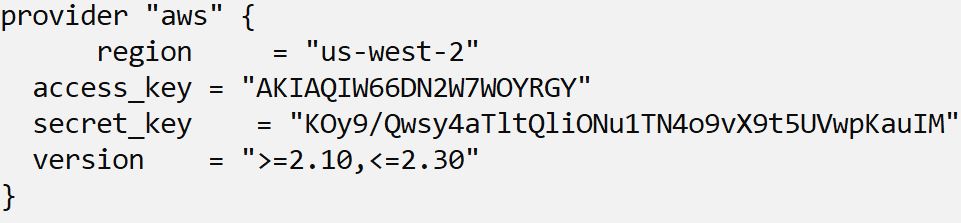
Important Pointers for Debugging:

TRACE is the most verbose and it is the default if TF\_LOG is set to something other than a log level name.

To persist logged output you can set TF\_LOG\_PATH in order to force the log to always be appended to a specific file when logging is enabled.

## **Module 11: Terraform Format**

Anyone who is into programming knows the importance of formatting the code for readability.



The terraform fmt command is used to rewrite Terraform configuration files to take care of the overall formatting



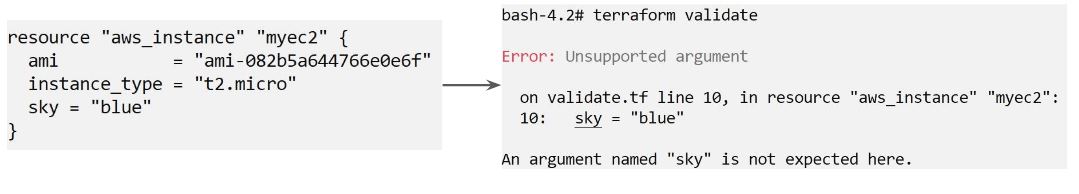
## 

## 

## **Module 12: Terraform Validate**

Terraform Validate primarily checks whether a configuration is syntactically valid.

It can check various aspects including unsupported arguments, undeclared variables, and others.

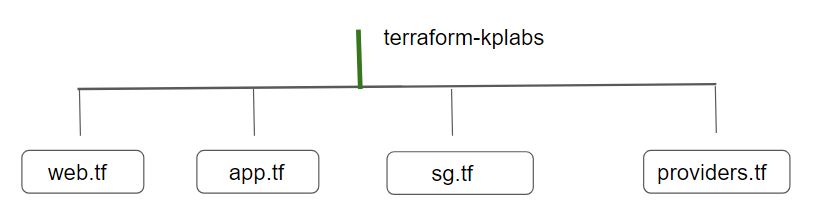


## 

## **Module 13: Load Order & Semantics**

Terraform generally loads all the configuration files within the directory specified in alphabetical order.

The files loaded must end in either .tf or .tf.json to specify the format that is in use.



## 

## 

## **Module 14: Dynamic Blocks**

14.1 Understanding the Challenge:

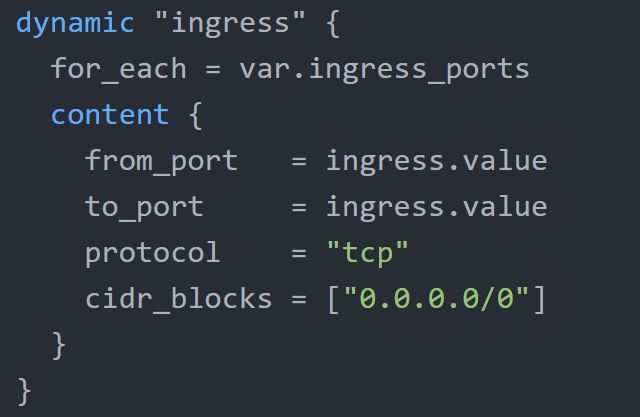
In many of the use-cases, there are repeatable nested blocks that need to be defined.

This can lead to a long code and it can be difficult to manage in a long time.



14.2 Overview of Dynamic Blocks

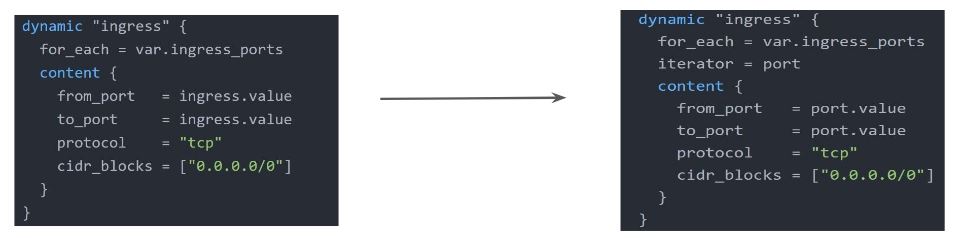
Dynamic Block allows us to dynamically construct repeatable nested blocks which is supported inside resource, data, provider, and provisioner blocks:



14.3 Overview of Iterators

The iterator argument (optional) sets the name of a temporary variable that represents the current element of the complex value

If omitted, the name of the variable defaults to the label of the dynamic block ("ingress" in the example above).



## 

## 

## 

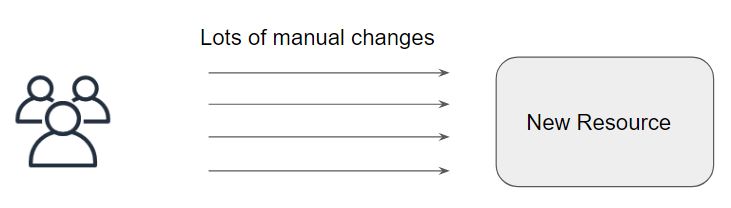
## **Module 15: Terraform Taint**

15.1 Understanding the Challenge:

You have created a new resource via Terraform.

Users have made a lot of manual changes (both infrastructure and inside the server)

Two ways to deal with this: Import The Changes to Terraform / Delete & Recreate the resource



15.2 Overview of Terraform Taint

The terraform taint command manually marks a Terraform-managed resource as tainted, forcing it to be destroyed and recreated on the next apply.



15.3 Important Pointers for Terraform Taint

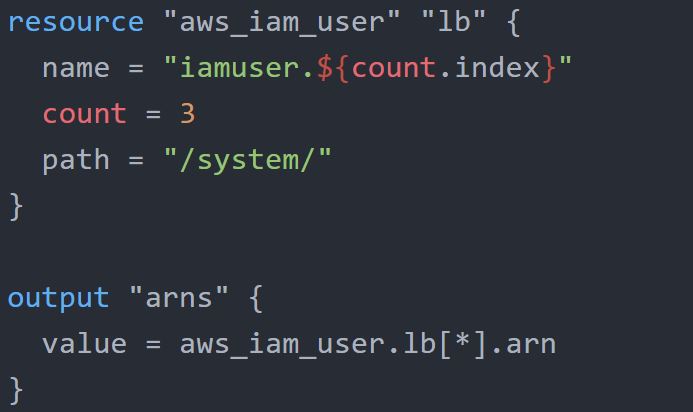
This command will not modify infrastructure but does modify the state file in order to mark a resource as tainted.

Once a resource is marked as tainted, the next plan will show that the resource will be destroyed and recreated and the next apply will implement this change.

Note that tainting a resource for recreation may affect resources that depend on the newly tainted resource.

## **Module 16: Splat Expression**

Splat Expression allows us to get a list of all the attributes.

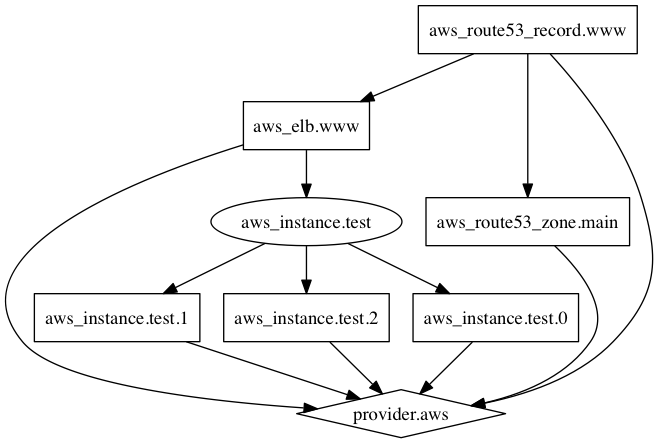


## 

## **Module 17: Terraform Graph**

The terraform graph command is used to generate a visual representation of either a configuration or execution plan

The output of terraform graph is in the DOT format, which can easily be converted to an image.



## 

## 

## **Module 18: Saving Terraform Plan to a File**

The generated terraform plan can be saved to a specific path.

This plan can then be used with terraform apply to be certain that only the changes shown in this plan are applied.

Example:

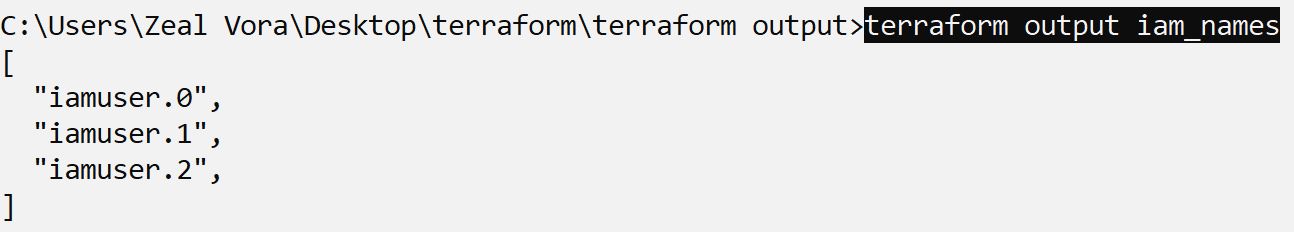
terraform plan -out=path

## 

## 

## **Module 19: Terraform Output**

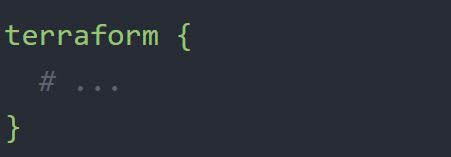
The terraform output command is used to extract the value of an output variable from the state file.



## **Module 20: Terraform Settings**

The special terraform configuration block type is used to configure some behaviors of Terraform itself, such as requiring a minimum Terraform version to apply your configuration.

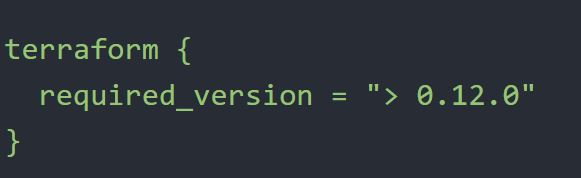
Terraform settings are gathered together into terraform blocks:



20.1 Setting 1 - Terraform Version

The required\_version setting accepts a version constraint string, which specifies which versions of Terraform can be used with your configuration.

If the running version of Terraform doesn't match the constraints specified, Terraform will produce an error and exit without taking any further actions.



20.1 Setting 2 - Provider Version

The required\_providers block specifies all of the providers required by the current module, mapping each local provider name to a source address and a version constraint.



## 

## 

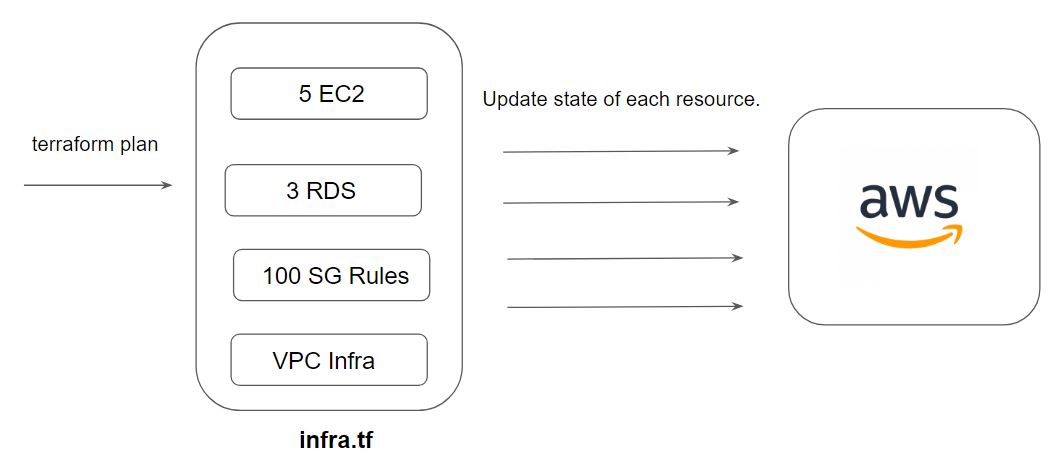
## 

## 

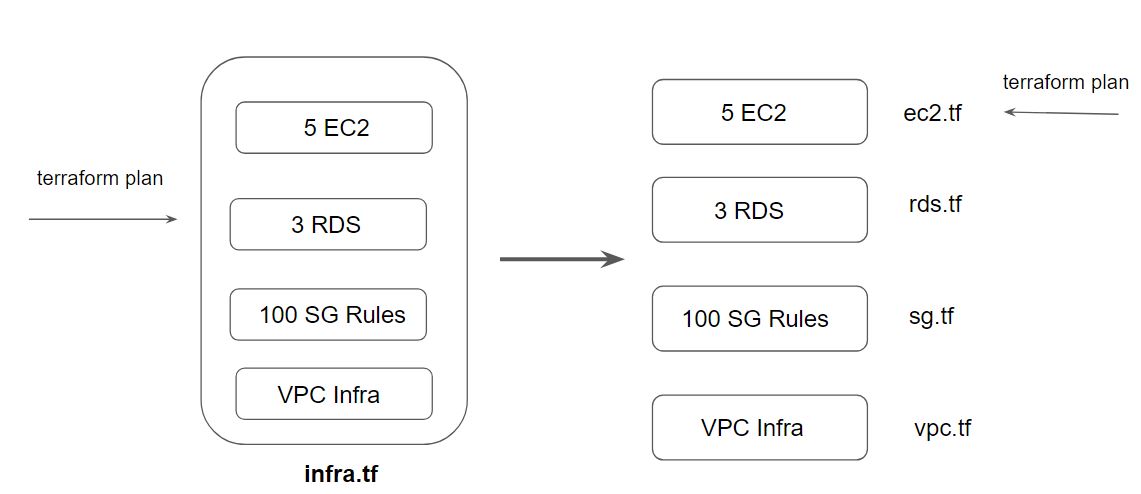
## 

## **Module 21: Dealing with Large Infrastructure**

When you have a larger infrastructure, you will face issues related to API limits for a provider.



It is important to switch to a smaller configurations were each can be applied independently.



21.1 Setting Refresh to False

We can prevent terraform from querying the current state during operations like terraform plan.

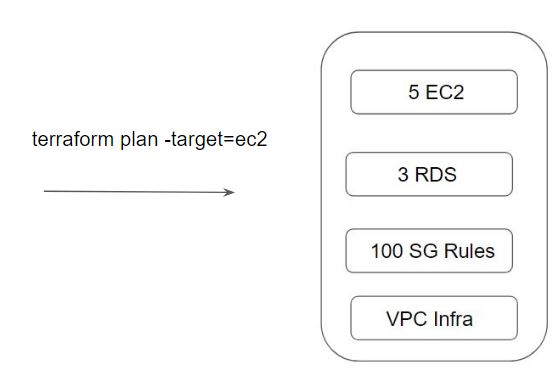
This can be achieved with the -refresh=false flag



21.2 Specify the Target

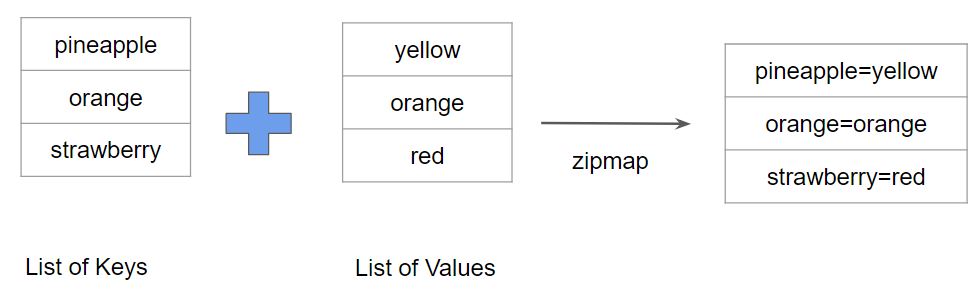
The -target=resource flag can be used to target a specific resource.

Generally used as a means to operate on isolated portions of very large configurations

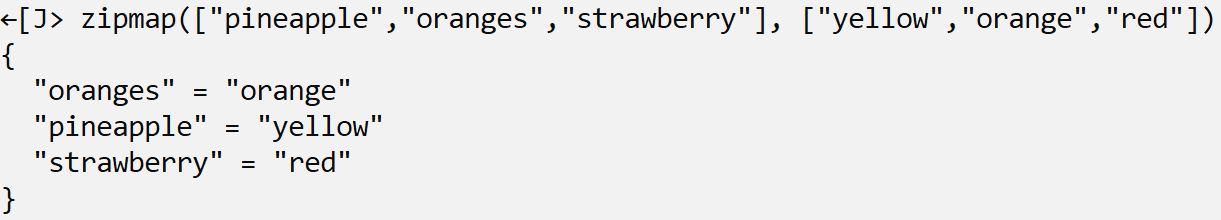


**Module 22 - Zipmap Function**

The zipmap function constructs a map from a list of keys and a corresponding list of values.



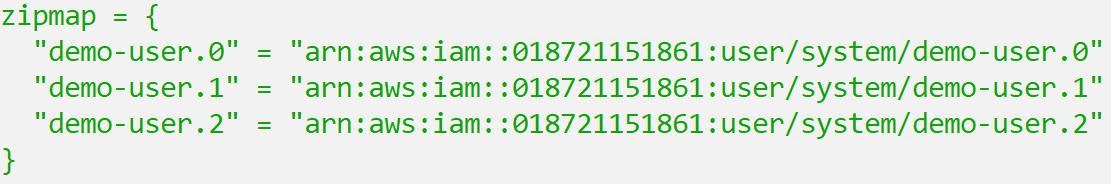
Following screenshot shows a sample output of Zipmap



Let us understand Zipmap with a sample use-case

You are creating multiple IAM users.

You need output which contains direct mapping of IAM names and ARNs



## 