spacelift // The Most Flexible IaC Platform

Terraform Essential Components

Whether you're just starting with Terraform or looking to brush up on the basics, this cheatsheet optimizes your workflow and helps you prepare for the certification.

Provider

- Plugins that allow Terraform to interact with specific infrastructure resources.
- Act as an interface between Terraform and the underlying infrastructure translating the Terraform configuration into the appropriate API calls
- Each provider has its own set of resources and data sources
- Providers exist even for applications that are not Cloud Vendor specific

```
provider "aws" {
 region = "us-east-1"
}

provider "aws" {
 alias  = "euwest"
 region = "eu-west-1"
}

provider "azurerm" {
 features {}
}

provider "kubernetes" {
 config_path = "~/.kube/config"
}
```

- You can define multiple providers of the same type by using the alias key word, in resources or datasources, you will have to specify either:
  - Resource: a provider parameter (example below)
  - Module: a provider block (example below)

```
# Alias Examples for Resources and Modules
resource "aws_instance" "this" {
 provider      = aws.euwest
 ami           = data.aws_ami.ubuntu.id
 instance_type = "t3.micro"
}

module "instance" {
 source = "../instance"
 providers = {
   aws = aws.euwest
 }
}
```

Resource

- Components of infrastructure that Terraform is able to manage: vms, vpcs, pods
- Each resource has a type, a name, some configurable arguments and some exposed attributes
- Terraform handles dependencies by default, but in order to create these dependencies between resources, you need to link the resources between them by using **type.name.attribute**

- How to handle the documentation
  - **Example Usage** → How to use the resource
  - **Argument Reference** -> What you can configure
  - **Attribute Reference** → What is the resource exposing
  - **Import** → What you can import

```
resource "azurerm_resource_group" "this" {
  name     = "rg1"
  location = "West Europe"
}

resource "azurerm_network_security_group" "this" {
  name                = "sg1"
  name                = azurerm_resource_group.this.location # ensures rg created before nsg and links to the above rg
  resource_group_name = azurerm_resource_group.this.name # ensures rg created before nsg and links to the above rg
}
```

Data Source

- Used to get different information that was created outside of Terraform
- Every provider has its own set of data sources
- Can be used inside of resources/locals/outputs
- Referenced with **data.type.name.attribute**

```
data "aws_ami" "ubuntu" {
  most_recent = true

 filter {
   name   = "name"
   values = ["ubuntu*"]
 }
}

resource "aws_instance" "this" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"
}
```

Output

- Used for exposing different attributes of a resource, variable, data source or local
- Exist outside of providers
- In Terraform Modules, they are used to expose some attributes that can be used in other configurations (either modules, resources, datasources, locals, providers)
- Supports three arguments, value (required), description and sensitive (whether or not the output is sensitive)

```
locals {
 a_list = [for i in range(10) : i]
}

output "a_list" {
 description = "This will print the elements of a_list"
 value       = local.a_list
}

# a_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Variable

- Declared with variable, referenced with var
- Simple types: number, string, bool
- Complex types: list, map, object, set → built from simple types
- Null types: used for omission
- 3 essential parameters when configuring a variable, all optional though: type, description and default (value)
- 3 other parameters: validation, sensitive, nullable

- Taking into account the definition precedence, you can assign values to variables:
  a. Inside of them
  b. In an environment variable
  c. In a terraform.tfvars file
  d. In a *.auto.tfvars file where * can be any name you want
  e. Using the -var option or -var-file option
- Terraform will use the last value it finds when you are declaring a variable value in multiple sources, but you will not be able to declare the same variable in the same source more than once, as this will result in an error

```
variable "a_string" {
 description = "This is a string"
 type        = string
 default     = "a_string"
}

variable "a_number" {
 description = "This is a number"
 type        = number
 default     = 10
}

variable "a_bool" {
 description = "This is a bool"
 type        = bool
 default     = false
}

variable "a_list" {
 description = "This is a list"
 type        = list(string)
 default     = ["elem1", "elem2"]
}
```

```
variable "a_map" {
 description = "This is a map"
 type        = map(string)
 default = {
   elem1 = "value1"
 }
}

variable "an_object" {
 description = "This is an object"
 type = object({
   name  = string
   power = number
 })
 default = {
   name  = "elem1"
   power = 5
 }
}
```

Local Variable

- Similar to a variable, but used to assign a name to an expression
- Exists in a locals block
- You can use constant values, attributes from resource, dynamically generate expression with loops or conditional statements

- Help with Do Not Repeat yourself paradigm, if overused will complicate your configuration, making it hard to maintain
- Referenced with **local.local_variable_name**

```
locals {
 expression1         = format("Hello %s!", "Mike")
 ternary_expression  = 3 > 2 ? 1 : 4              # If 3 is greater than 2 which it is, it will use value 1, otherwise value 4
 for_expression_list = [for i in range(5) : i]    # This will create a list of numbers from 0 to 4
 for_expression_map  = { for i in range(10) : format("Number_%d", i) => i } # This will create a map with key value pairs in the format Number_0
=> 0
}
```

Provisioner

- Used for doing operations outside Terraform's lifecycle
- Exist only inside of a resource, there is also a special type of resource called null_resource that can be leveraged to work with providers (these resources aren't doing anything on their own)
- Support **when** (this refers to when do you want them to be applied **on_create** which is default or **on_destroy**) and **on_failure** (which refers to the behavior they have when they encounter an error: **continue** to ignore the error, **fail** which is the default to behave exactly like any other statement in Terraform)
- 3 available provisioners, local-exec (runs a local command), remote-exec (runs a remote command), file (copies a file from local to remote)
- Remote provisioners need a connection block to identify where they should connect
- Should be used only as a last resort - you should use configuration management tools (Ansible, Chef, Puppet) to achieve these types of tasks

```
resource "null_resource" "this" {
 provisioner "local-exec" {
   command = "ls -l"
 }

 connection {
   type        = "ssh"
   user        = "ubuntu"
   private_key = file("~/.ssh/id_rsa")
   host        = "your_host"
 }
provisioner "remote-exec" {
  inline = [
    "echo hello"
  ]
 }

 provisioner "file" {
   source      = "./file.yaml"
   destination = "./file.yaml"
 }
}
```

spacelift

The Most Flexible IaC Platform