**lecture retrospective**

寫到現在，應該都可以寫出能夠工作的 terraform。

到目前為止，課程對學生的期待是『能夠 google 到社群分享的 .tf，並且會正確的複製貼上，轉化成自己的 .tf 內容』。

做到現在，想必也累積了一些問題。之前的內容，或許有些邏輯還不是很清楚，課程都簡單帶過，許多複雜的部分略過不提;目的先讓學生對語言有個基礎的了解，有實際操作的經驗跟手感，之後做深入的討論時，更能理解內容。

接下來要繼續深入，我們會花一點篇幅，回頭細講細節，把關防文件細細地走過

**Terraform syntax**

先從這篇出發[Terraform Syntax](https://www.terraform.io/docs/language/syntax/index.html)，介紹 Terraform syntax。這裡又分為兩部分：一個是[Configuration Syntax](https://www.terraform.io/docs/language/syntax/configuration.html)，另一個是[Json Configuration Syntax](https://www.terraform.io/docs/language/syntax/json.html)。乍看之下有點疑惑，json 是從哪跑出來的？我們可以從幾個角度看這件事：

Terraform 底層 low-level syntax 是由 [Hashicorp Configuration Language](https://github.com/hashicorp/hcl) 定義的

* HCL 又是什麼東西？不精確的說，他就是過去幾堂課我們寫的 .tf 的內容語法
  + 精確的說應該反過來，.tf 的語法底層是由 hcl 定義
* [Hashicorp 當初推出 HCL 的目的](https://github.com/hashicorp/hcl#why)，而不直接用 json 或 yaml
  + 許多高階程式語言(ex. ruby, golang, c...) 不易描述多階層結構資料(ex. json)，與宣告式(declarative)的資料內容
    - 而這兩者在定義 infrastructure 等 configuration / state 時很重要
    - 這也是許多應用的 conf 不會用高階語言來直接定義，例如：
      * kubernetes 與 CNCF 專案使用 yaml / json
      * nginx.conf 的語法很有趣，對 hcl 的設計影響很大
  + [json 作為 data-interchange format](https://www.json.org/json-en.html)，設計本身已經是人類可讀，語法結構非常清晰，也是泛用的 api 溝通格式
    - 然而太複雜的結構可讀性變差。例如 json key 深度越多層，可讀性大幅下降，必須使用外部 tool parse （ex. 本課程愛用的 jq）
    - json 不易描述更多增益功能性語法，例如：沒有 native comment，loop，data type，escape...等
    - yaml 也是有同樣的問題
* HCL 依循 key-value 與階層 block {} 設計(類似 json)
* 並增加許多功能性的語法定義，讓使用 HCL 語言的應用（ex. terraform ）

這裡要講古一下：歷史來看， hashicorp 在 2014 年發表 terraform，同時也釋出 hcl 語言 spec

* hashicorp 2014 前的產品採用不同的 conf
  + vagrant 也是 configuration 工具採用 ruby
  + packer 使用 json。然而隨著 hcl 逐漸穩定，新版 packer 也支援 hcl 的 template
* terraform configuration 是第一個套用 hcl spec 的產品，hcl spec 也隨 terraform 的更新而演進
* 後面的產品（consul，vault）都變成 hcl 的形狀了

用範例講解一下上面一大段，回頭看 \_poc/security\_group/security\_group.tf

* 他是 terraform configuration syntax，底下是 hcl
* 支援 .tf native 格式，也支援 .tf.json json 格式

使用 cli 的 global option -json 來輸出 json

* 比較 hcl 與 json 的輸出差異
* json 乍看之下有點凌亂，使用 jq parse 後，會發現輸出內容相同
* double quote 的跳脫字元 (escape)

terraform plan

*# azurerm\_network\_security\_group.main will be created*

+ resource "azurerm\_network\_security\_group" "main" {

+ id = (known after apply)

+ location = "southeastasia"

+ name = "poc-chechia"

+ resource\_group\_name = "terraform-30-days"

+ security\_rule = (known after apply)

+ tags = {

+ "environment" = "poc"

}

}

*# azurerm\_network\_security\_rule.main["homeport22"] will be created*

+ resource "azurerm\_network\_security\_rule" "main" {

+ access = "Allow"

+ destination\_address\_prefix = "\*"

+ destination\_port\_range = "22"

+ direction = "Inbound"

+ id = (known after apply)

+ name = "Port\_22"

+ network\_security\_group\_name = "poc-chechia"

+ priority = 100

+ protocol = "\*"

+ resource\_group\_name = "terraform-30-days"

+ source\_address\_prefix = "17.110.101.57"

+ source\_port\_range = "\*"

}

terraform plan -json

{"@level":"info","@message":"azurerm\_network\_security\_group.main: Plan to create","@module":"terraform.ui","@timestamp":"2021-08-18T22:58:26.459366+08:00","change":{"resource":{"addr":"azurerm\_network\_security\_group.main","module":"","resource":"azurerm\_network\_security\_group.main","implied\_provider":"azurerm","resource\_type":"azurerm\_network\_security\_group","resource\_name":"main","resource\_key":null},"action":"create"},"type":"planned\_change"}

{"@level":"info","@message":"azurerm\_network\_security\_rule.main[\"homeport22\"]: Plan to create","@module":"terraform.ui","@timestamp":"2021-08-18T22:58:26.461178+08:00","change":{"resource":{"addr":"azurerm\_network\_security\_rule.main[\"homeport22\"]","module":"","resource":"azurerm\_network\_security\_rule.main[\"homeport22\"]","implied\_provider":"azurerm","resource\_type":"azurerm\_network\_security\_rule","resource\_name":"main","resource\_key":"homeport22"},"action":"create"},"type":"planned\_change"}

{"@level":"info","@message":"Plan: 2 to add, 0 to change, 0 to destroy.","@module":"terraform.ui","@timestamp":"2021-08-18T22:58:26.461227+08:00","changes":{"add":2,"change":0,"remove":0,"operation":"plan"},"type":"change\_summary"}

*# format with jq*

{

"@level": "info",

"@message": "azurerm\_network\_security\_rule.main[\"homeport22\"]: Plan to create",

"@module": "terraform.ui",

"@timestamp": "2021-08-18T22:58:26.461178+08:00",

"change": {

"resource": {

"addr": "azurerm\_network\_security\_rule.main[\"homeport22\"]",

"module": "",

"resource": "azurerm\_network\_security\_rule.main[\"homeport22\"]",

"implied\_provider": "azurerm",

"resource\_type": "azurerm\_network\_security\_rule",

"resource\_name": "main",

"resource\_key": "homeport22"

},

"action": "create"

},

"type": "planned\_change"

}

{

"@level": "info",

"@message": "Plan: 2 to add, 0 to change, 0 to destroy.",

"@module": "terraform.ui",

"@timestamp": "2021-08-18T22:58:26.461227+08:00",

"changes": {

"add": 2,

"change": 0,

"remove": 0,

"operation": "plan"

},

"type": "change\_summary"

}

.tf 與 .tf.json 的格式轉換，我們可以用一個[小工具 kvz/json2hcl](https://github.com/kvz/json2hcl)來做轉換

*# azure/\_poc/security\_group/provider.tf*

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~> 2.65.0"

}

}

required\_version = ">= 1.0.1"

*# remote Backend*

backend "azurerm" {

resource\_group\_name = "terraform-30-days-poc"

storage\_account\_name = "tfstate8b8bff248c5c60c0"

container\_name = "tfstate"

key = "\_poc/security\_group/terraform.tfstate"

}

}

provider "azurerm" {

features {}

}

安裝 json2hcl

# azure/\_poc/security\_group/provider.tf

curl -SsL https://github.com/kvz/json2hcl/releases/download/v0.0.6/json2hcl\_v0.0.6\_darwin\_amd64 \

| sudo tee /usr/local/bin/json2hcl > /dev/null && sudo chmod 755 /usr/local/bin/json2hcl && json2hcl -version

v0.0.6

json2hcl -reverse < azure/\_poc/security\_group/provider.tf

{

"provider": [

{

"azurerm": [

{

"features": [

{}

]

}

]

}

],

"terraform": [

{

"backend": [

{

"azurerm": [

{

"container\_name": "tfstate",

"key": "\_poc/security\_group/terraform.tfstate",

"resource\_group\_name": "terraform-30-days-poc",

"storage\_account\_name": "tfstate8b8bff248c5c60c0"

}

]

}

],

"required\_providers": [

{

"azurerm": [

{

"source": "hashicorp/azurerm",

"version": "~\u003e 2.65.0"

}

]

}

],

"required\_version": "\u003e= 1.0.1"

}

]

}

然而輸入 \_poc/security\_group/security\_group.tf 則會出錯

* json 沒有辦法辨識 hcl 的功能性語法
  + variable 如：local. each.，是 hcl 送進 terraform 後才能有效 evaluate
  + for\_each 在 json 中也不存在

json2hcl -reverse < azure/\_poc/security\_group/security\_group.tf

unable to parse HCL: At 2:25: Unknown token: 2:25 IDENT local.name

我們可以依據[hcl readme](https://github.com/hashicorp/hcl#information-model-and-syntax) 補上 double quote ""，讓格式符合 json 的格式，這樣就可以順利轉換

* 轉換成標準 json
* "${ }" 對 terraform 有額外意義，會自動 evaluate 變數值

cat \_poc/security\_group/security\_group\_json.tf

resource "azurerm\_network\_security\_rule" "main" {

for\_each = "${local.rules}"

name = "${each.value.name}"

priority = "${each.value.priority}"

direction = "${each.value.direction}"

access = "${each.value.access}"

protocol = "${each.value.protocol}"

source\_port\_range = "${each.value.source\_port\_range}"

destination\_port\_range = "${each.value.destination\_port\_range}"

source\_address\_prefix = "${each.value.source\_address\_prefix}"

destination\_address\_prefix = "${each.value.destination\_address\_prefix}"

resource\_group\_name = "${local.resource\_group\_name}"

network\_security\_group\_name = "${local.name}"

}

json2hcl -reverse < azure/\_poc/security\_group/security\_group\_json.tf

{

"resource": [

{

"azurerm\_network\_security\_rule": [

{

"main": [

{

"access": "${each.value.access}",

"destination\_address\_prefix": "${each.value.destination\_address\_prefix}",

"destination\_port\_range": "${each.value.destination\_port\_range}",

"direction": "${each.value.direction}",

"for\_each": "${local.rules}",

"name": "${each.value.name}",

"network\_security\_group\_name": "${local.name}",

"priority": "${each.value.priority}",

"protocol": "${each.value.protocol}",

"resource\_group\_name": "${local.resource\_group\_name}",

"source\_address\_prefix": "${each.value.source\_address\_prefix}",

"source\_port\_range": "${each.value.source\_port\_range}"

}

]

}

]

}

]

}

這也是舊版 terraform 常出現 "${}" 語法的原因。新版 terraform 已經能自動 parse 沒有 double quote 的 hcl 語法，並且 validate 時會對有不必要 double quote 的 "${}" 語法跳出 warning。

**Review terraform resource**

了解 terraform syntax 後，我們要回頭[重新檢視 hcl 中描述的 terraform resources](https://www.terraform.io/docs/language/resources/index.html)，以及[為什麼 resources 會有這些行為](https://www.terraform.io/docs/language/resources/index.html)

resource block {}

* hcl parse 後是一層 json node
* block 內有許多參數 Resource Arguments，包含：
  + resource 遠端物件的 arguments
    - 例如：resource.azurerm\_linux\_virtual\_machine 的 name, size ...等是雲端物件的參數
    - 這些物件與參數，透過 provider 實作，轉成公有雲 api 接受的 json payload，向 api 發出 request，變更遠端 resource
    - terraform 可以管理或變更這些雲端物件參數
  + 另外還有 [resource meta-argument](https://www.terraform.io/docs/language/resources/syntax.html#meta-arguments) （ex. provider, for\_each, lifecycle,...）
    - 這些 meta-argument 是 terraform 的參數，也就是上面提到的功能性參數，用來改變 resource 行為，方便使用者透過 terraform 做更高階的 resource 的管理
    - 例如：在 terraform 內操作迴圈控制 resource
    - meta-argument 只在 terraform 內部有效，作用時間與其他參數不同

我們在編寫 hcl 內容時應善用 meta-argument，應注意

* 哪些參數是 terraform 內參數，哪些是 meta-argument
* terraform cli workflow 的各個階段，argument 會作用
* module 間的 depends\_on 與 variable + argument 的使用，會造成 module 彼此的依賴性
  + 造成 apply 時候，terraform 送出 request 的先後順序有差異
* for each 與 count 是強大的工具，但使用上需要注意的地方，我們下一章節會說明

**Source code**

如果對 Terraform 原始碼有興趣，可以看與 hcl config parse

* [internal/configs/parser.go](https://github.com/hashicorp/terraform/blob/main/internal/configs/parser.go)

**Homework**

官方文件閱讀測驗

* [Terraform: configuration syntax](https://www.terraform.io/docs/language/syntax/configuration.html)
  + 了解平時使用的 .tf 內容底層是如何定義的
  + 自己的理解，與官方文件描述內容是否相符
  + 被一些單字，官方是如何用英文描述功能，描述元件，描述語法
    - 之後遇到問題 google 找問題時，使用對的單字問對的問題，才搜尋的到解答

**References**

* [HCL Native syntax specification](https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md)

**Next session**

meta-argument 只在 terraform 內部有效，細節範例我們下一章節會說明