本章介紹 terraform 的 for\_each，以及如何運用 meta-argument 管理 resource

**Resource and meta-argument**

一般來說，每個 resource block {} 就代表一個 resource。然而在實務中，我們會有額外的需求

* 需要管理許多大量重複的 resource
* 需要管理重複的一組 resource

如果全部都寫成一個一個獨立的 resouce block

* 會有大量重複的程式碼，違反 DRY，且難以管理

因此 terraform 有提供 meta-argument。使用者可以在 resource 以外的地方，使用 meta-argument 改變 terraform evaluate resource block 的行為。

**About loop**

幾乎所有的高級程式語言都有提供 for, each, loop 等語法，來處理不斷重複的邏輯。能夠高頻率的執行重複工作，這對軟體產業的效能是至關重要的。

這也是 IaC 管理公有雲時，相對於 web console 操作有決定性的優勢，例如：

* 管理 1 台 VM 或是 10 台 VM，還可以使用 web console 手動控制
* 數量級再提升，例如 100 台 1000 台，沒有工具協助是做不到的
* 人腦很不擅長處理，不斷重複卻又有一定變化程度的工作，很容易導致  
  human error

Terraform 在管理大量類似的雲端 resource 是十分有效率的，底下為各位示範。我們可以先用自己熟悉的語言來做個想像

例如 Cloud Computing 最常用到的功能之一，使用 loop 快速 provision VM

* 同時要建立多個附屬 resource，例如取得 public IP
* 套用同一組防火牆規則，把 VM 加入管理
* ...

如果是熟悉的語言，大概會是長什麼樣子

*# sudo code*

createFirewallRule()

for i from 0 to 9 {

createPublicIP($i)

}

for j from 0 to 9 {

createVM(

name: $j

publicIP: $ip

firewallRule: $firewallRule

...

)

}

**Terraform meta-argument**

Terraform 提供兩個不同用途的

* [resource / module block meta-argument: for each](https://www.terraform.io/docs/language/meta-arguments/for_each.html)
* [resource / module block meta-argument: count](https://www.terraform.io/docs/language/meta-arguments/count.html)
* [for expression](https://www.terraform.io/docs/language/expressions/for.html) 是與其他高階程式語言相近的 for loop 語法
  + 要注意 type

**loop expression in Terraform**

範例：modules/kubernetes\_cluster/node\_pool.tf

# modules/kubernetes\_cluster/node\_pool.tf

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main" {

for\_each = var.node\_pools

name = each.value.name

kubernetes\_cluster\_id = azurerm\_kubernetes\_cluster.main.id

vm\_size = each.value.vm\_size

node\_count = each.value.node\_count

mode = each.value.mode

priority = each.value.priority

node\_labels = each.value.node\_labels

node\_taints = each.value.node\_taints

depends\_on = [

azurerm\_kubernetes\_cluster.main

]

}

resource.azurerm\_kubernetes\_cluster\_node\_pool.main 中使用兩個 meta-arguments

* depends\_on 建立依賴關係，確保 node\_pool 會在 azurerm\_kubernetes\_cluster.main 產生後才會產生
* for\_each 可以依據 var.node\_pools，為每個 var.node\_pools 的 element 產生一個 resource.azurerm\_kubernetes\_cluster\_node\_pool.main

**Depends On**

首先先看 [depends on terraform 官方文件](https://www.terraform.io/docs/language/meta-arguments/depends_on.html)

* depends\_on explicitly 建立依賴關係，確保 node\_pool 會在 azurerm\_kubernetes\_cluster.main 產生後才會產生
* 影響 provider 對 azure cloud 發出 api 的時機，i.e.
* 延伸問題；如果不使用 depends\_on 的話，plan 或 apply 仍然會成功嗎？
  + 在這個範例，沒有 depends\_on 的話不影響順序
  + 由於 node\_pool 的參數中， kubernetes\_cluster\_id = azurerm\_kubernetes\_cluster.main.id 就需要 cluster 的 id，而 id 是 kubernetes\_cluster 的參數，產生之後才會取得，terrafrom 會自動推測（automatically infer）兩者的 dependency
  + 如果是其他例子，並沒有透過參數引用建立關係，terraform 便無法判斷，可能造成 apply 失敗

提醒注意的地方

* 雖然 terraform 提供 depends\_on meta-argument，然而官方建議使用上應該作為最後手段(last resort)
* 使用參數
* depends\_on 無法使用 Arbitrary expressions，意思是不能使用可變的數值，而是必須靜態的值，舉例
  + 如果改成下面這樣
    - 希望判斷 var.node\_pools 內部 element 的長度，如果沒有 element 就不用建立 depends\_on
    - 這種寫法 validate 會直接判斷 Invalid Expression，無法 init 或 plan

depends\_on = [

length(var.node\_pools) > 0 ? azurerm\_kubernetes\_cluster.main : null

]

terragrunt init

There are some problems with the configuration, described below.

The Terraform configuration must be valid before initialization so that

Terraform can determine which modules and providers need to be installed.

╷

│ Error: Invalid expression

│

│ on node\_pool.tf line 16, in resource "azurerm\_kubernetes\_cluster\_node\_pool" "main":

│ 16: length(var.node\_pools) > 0 ? azurerm\_kubernetes\_cluster.main : null

│

│ A single static variable reference is required: only attribute access and

│ indexing with constant keys. No calculations, function calls, template

│ expressions, etc are allowed here.

╵

ERRO[0003] 1 error occurred:

\* exit status 1

**Expression evaluation**

為何無法使用，許多 meta-argument 也是必須使用靜態的參數：例如 depends\_on, provider, lifecycle

* 使用 provider 作為範例來說明可能比較單純
* terraform 首先需要 init，明確指定使用的 provider 版本與 source，或是讓 terraform 掃描並推測使用的 provider
* 沒有 provider，便無法 plan 各個 resource，包含內部的參數 validate
* 讀取 state 內容，plan 與 apply 都必須有靜態的 provider 存在，如果 provider 使是動態產生，有時存再有時不存在便會影響上述工作
* terraform 限制 provider 的使用，設定必須是靜態的
* 但在少數的情形，可以產生 provider 後，透過操作移除已經 init 的 provider，這時遠端 state 上已經存在，但本地沒有 provider 可以讀取與 plan，便會產生 orphan state
  + 沒有 provider 可以 plan，當然就無法透過 provider 修改或刪除，工作被卡住
* depends\_on 概念相同，如果依賴有時為 true，有時為 false 的判斷 expression，就無法建立靜態的依賴性
* lifecycle 可以改變 terraform plan, apply 中的行為，也需要靜態的 expression 設定

[Terraform reference to values](https://www.terraform.io/docs/language/expressions/references.html)