**Count**

要管理重複的 resource block ，Terrafrom 還提供另一個 meta-argument count。範例

* 建立一組 azurerm\_virtual\_machine
* 使用 count meta-argument，告訴 terraform 這組 resource block 要有三個
* 希望 vm.name 是 unique 方便辨識，所以使用 ${count.index} 取得每個 count 產生的 resource 的 index

resource "azurerm\_virtual\_machine" "main" {

count = 3

name = "${var.prefix}-vm-${count.index}"

location = azurerm\_resource\_group.main.location

resource\_group\_name = azurerm\_resource\_group.main.name

network\_interface\_ids = [azurerm\_network\_interface.main.id]

vm\_size = "Standard\_DS1\_v2"

...

}

實際 output 會類似

* 產生一組 recource block
* state 路徑在 azurerm\_virtual\_machine.main，這邊會是一組 collection，可以使用 index 存取
* azurerm\_virtual\_machine.main[0]
* azurerm\_virtual\_machine.main[1]
* azurerm\_virtual\_machine.main[2]

resource "azurerm\_virtual\_machine" "main[0]" {

name = "${var.prefix}-vm-0"

...

}

resource "azurerm\_virtual\_machine" "main[1]" {

name = "${var.prefix}-vm-1"

...

}

resource "azurerm\_virtual\_machine" "main[2]" {

name = "${var.prefix}-vm-2"

...

}

Count 與 for each 是互斥的，意思是 resource block 中只能使用其中一個 meta-argument，一起使用的話會在 validate 出 syntax error

**Count index Issue**

for each 使用 input variable 的 key 作為 key，count 使用則搭配 count.index，在 collection 取得參數值

首先是 node pool 範例，使用 for each meta-argument

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

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

for\_each = var.node\_pools

name = each.value.name

...

}

* 我們可以更改 for each，改用 count meta-argument 來描述
* 使用 length() function 取得 map of object 的 member 數量，作為 count 參數
* each 也改用 count.index 來存取 var.node\_pools

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

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

count = length(var.node\_pools)

name = var.node\_pools[count.index].name

...

}

展開變成

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

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main[0]" {

name = var.node\_pools[0].name

...

}

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main[1]" {

name = var.node\_pools[1].name

...

}

注意：上面使用 count 與 for each 的取值方式不同，這裡會可能造成 count.index 的錯亂

* terraform 的 map 是 unordered map，本身沒有 index
* 使用 for each 時，map 是 order 是因為依照 key 的 alphabatical order，依據
* 使用 count.index 時，不保證 index 與 key 的順序相同
* 如果上層 var.node\_pools 有改變， plan 的時候重新計算 resource block，便有可能導致順序錯亂
* 加上由於這個例子中的 resource 沒有依賴性，是平行化產生的，本身不保證先後順序，可能會產生問題。

Terraform 官方在 [When to use count and for each](https://www.terraform.io/docs/language/meta-arguments/count.html#when-to-use-for_each-instead-of-count) 說明 count 與 for each 建議的使用時機，已經不建議如此使用 count 了

那為何 count 還是會存在？是歷史緣故 resource 中的 count 支援版本很早，for each 要到 0.12 之後的 terraform 版本才支援。也就是說，古人沒有 for each 可以用被迫使用 count + count.index

* 事實上，如果使用 terraform 久了，還是有機會在比較舊的 module 立面看到 count 的大量使用

**Count binding multiple variables**

count 歷史悠久，可以分享一些常見的用法

variable "vm\_names" {

default = ["vm-1", "vm-2", "vm-3"]

type = list

}

variable "vm\_sizes" {

default = ["Standard\_DS1\_v2", "Standard\_DS2\_v2", "Standard\_DS4\_v2"]

type = list

}

resource "azurerm\_virtual\_machine" "main" {

count = length(var.vm\_names)

name = var.vm\_names[count.index]

vm\_size = var.vm\_sizes[count.index]

}

上面這個用法的問題

* 使用兩個參數來定義同一個物件，很不直覺
* 這也是歷史緣故，舊版的 type constraint 並不像現在這麼完整，有辦法從 map of map 中輕易取值。可能會需要調用許多 function 來取得正確的值
* 最大的問題是上面提過的，count.index 不保證有序的問題(這個例子是安全的，因為 list 有 built-in index，list[index] 可以取得正確的值)
* 如果 variable type 是 map 或是 any 的話就會有問題

新版 terraform 請使用

* 將 vm\_names 與 vm\_sizes 合併成為單一 variable
* 使用 for each 來 iterate 上面這個 map

**Count conditional**

count 可以接受 0 為參數，意思是就產生 0 個 resource block

resource "azurerm\_virtual\_machine" "main" {

count = 0

name = "${var.prefix}-vm-${count.index}"

...

}

這又產生了另外一個用法，來有條件的控制 resource block 與 module

variable "enable\_azurerm\_virtual\_machine" {

type = bool

default = false

}

resource "azurerm\_virtual\_machine" "main" {

count = var.enable\_azurerm\_virtual\_machine ? 1 : 0

name = "${var.prefix}-vm-${count.index}"

...

}

上面這個範例

* variable enable\_azurerm\_virtual\_machine == true 的時候，會產生 count = 1，也就是產生一個 azurerm\_virtual\_machine
* variable enable\_azurerm\_virtual\_machine == false 的時候，會產生 count = 0，也就是產生一個 azurerm\_virtual\_machine
* 每一次 plan variable 都是定值，因此這樣的寫法，雖然看起來是 dynamic expression，但實際上以 root module 的角度看是 deterministic

實務上這樣子的使用情境還算蠻多的，一個 resource / module 啟用或不啟用

* for each 也能達成相同效果（如果 for\_each 的 argument literate 下去是 empty 的話，for each 出來就會產生 0 個 resource / module

**Count vs for each**

Terraform 官方在 [When to use count and for each](https://www.terraform.io/docs/language/meta-arguments/count.html#when-to-use-for_each-instead-of-count) 說明 count 與 for each 建議的使用時機

* 如果產生一組全部都相同的 resource block，可以使用 count
* 如果內部有變數處理，或是取用 input argument 的值，使用 for each 會比較安全

能用 for each 的時候就用 for each

* for each 能夠使用 map
* 使用 collection

**Source code**

熟 golang 的不妨看一下 source code

* [count evaluation](https://github.com/hashicorp/terraform/blob/c687ebeaf19c7c89188727ffc54b03bcc6e51a01/internal/terraform/eval_count.go)
  + 以及 sensitive input 的處理