**for Expression**

For expression 是 terraform configuration language (hcl) 內的 syntax 語法  
for each meta-argument 是

* for\_each 是在計算 resource block {} 的 meta 時使用
* for expression 是在處理 values 的運算

<https://www.terraform.io/docs/language/expressions/for.html>

poolNameTuple = { for p in var.node\_pools : name => p.name }

poolNameList = [

for p in var.node\_pools : "p.name"

]

output "pool\_name\_list" {

value = [for p in var.node\_pools : p.name]

}

兩個語法的定義位階不同

* 一個是 terraform 的功能
* 一個是 hcl 就定義的語法

**for expression as for each argument**

for\_each = [for p in var.node\_pools : p.name]

for 產出的是 list

for\_each = [

var.node\_pools["spot"].name

var.node\_pools["on-demand"].name

...

]

一樣需要考量 for each 的限制

* for expression evaluation 需要是決定性的，不能有 conditional expression

**Homework: for examples**

請依照[Terraform 官方文件 for example](https://www.terraform.io/docs/language/expressions/for.html" \t "_blank)，嘗試每個 example，以熟悉 for syntax 使用

**Dynamic block**

想要管理很多類似的多個 resource block，我們可以參考使用 meta-argument，來管理 resource block

有個時候，一個 resource block 中，會有 repeatable nested blocks arguments，例如

* [azurerm virtual machine 中](https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/virtual_machine#storage_data_disk)，可以定義多個 storage\_data\_disk

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

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

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"

storage\_os\_disk {

name = "myosdisk1"

caching = "ReadWrite"

create\_option = "FromImage"

managed\_disk\_type = "Standard\_LRS"

}

storage\_data\_disk {

name = "data1"

caching = "None"

create\_option = "Attach"

disk\_size\_gb = "100"

}

storage\_data\_disk {

name = "data2"

caching = "None"

create\_option = "Attach"

disk\_size\_gb = "200"

}

storage\_data\_disk {

name = "data3"

caching = "None"

create\_option = "Attach"

disk\_size\_gb = "300"

}

...

}

這時候，是想要管理一個 block 中的 field block，讓他一據 input 動態產生，這時可以使用 [Dynamic block](https://www.terraform.io/docs/language/expressions/dynamic-blocks.html)

以上面這個例子，可以改寫成 dynamic blocks

locals {

storage\_data\_disks = [

{

name = "data1"

disk\_size\_gb = "100"

},

{

name = "data2"

disk\_size\_gb = "200"

},

{

name = "data3"

disk\_size\_gb = "300"

}

]

}

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

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

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"

storage\_os\_disk {

name = "myosdisk1"

caching = "ReadWrite"

create\_option = "FromImage"

managed\_disk\_type = "Standard\_LRS"

}

dynamic "storage\_data\_disk" {

for\_each = var.storage\_data\_disks

content {

name = storage\_data\_disk.value["name"]

caching = "None"

create\_option = "Attach"

disk\_size\_gb = storage\_data\_disk.value["disk\_size\_gb"]

}

}

...

}

**Multi-level nested dynamic block**

許多高階程式語言有提供 nested loop，terraform dynamic block 也提供 nested

* 這邊範例使用的 variable 是 map of object
* object 中的 origins field type 是 set of object

variable "load\_balancer\_origin\_groups" {

type = map(object({

origins = set(object({

hostname = string

}))

}))

}

resource "load\_balancer" "main" {

dynamic "origin\_group" {

for\_each = var.load\_balancer\_origin\_groups

content {

name = origin\_group.key

dynamic "origin" {

for\_each = origin\_group.value.origins

content {

hostname = origin.value.hostname

}

}

}

}

}

這個範例，在複雜的網路設定相關的 resource 很有機會看到。double nested dynamic block 的問題

* 可讀性已經非常差了
* debug 的時候更痛苦
* 由於是 dynamic block，需要 evaluation 的時候才會展開這些 block
* 需要善用 console debug，才知道展開到底長什麼樣子

通常比較好的 provider （例如三大公有雲）會提供另外的 resource block 來管理這些 nested block

* 可以寫 load\_balancer resource
* 加上 origin\_group resource
* 後使用 attachment，把兩個 resource 關聯起來

**Note**

* for expression 是 hcl syntax，意思是只要以 hcl 為底層的 configutation language 都可以使用（ex. tcl，vault config，consul config，...）
* for each meta-argument 是 terraform 的 resource block argument，terraform 中才可以使用，並且用來操作 resource block
* dynamic block 是 terraform 的 in-resource argument，可以視作 resource block {} 中的一個特別的參數，用來動態產生 repeatable nested block

**Source code**

熟 golang 的不妨看一下 source code

* [For each evaluation](https://github.com/hashicorp/terraform/blob/c687ebeaf19c7c89188727ffc54b03bcc6e51a01/internal/terraform/eval_for_each.go)
  + for each evaluation 的流程
  + iterate set 的流程

**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}"

...

}

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 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