**Terraform meta-argument: for each**

這邊細講 for\_each meta-argument 的相關使用

* 使用 for\_each 的 resource block 中，可以在使用 each object，來取得 for\_each 內的 key 與 value

以 node\_pools 這個變數為例，本身是 module 的 intput variable

* input type 定義是一個 map(any)
* 實際上 map of object({ name=string, ... })
* 使用 any 只是偷懶，terraform 不會 validate input 內部 object 的 type constraint
* 由於 validate 時沒有檢查，就是看 runtime 的時候，去 input variable 內部取值有無錯誤
* 使用 any 太偷懶了，應該改成明確的 type constraint
* [type 的細節，請見後面的 Terraform Type 說明](https://ithelp.ithome.com.tw/articles/10264034)

# modules/kubernetes\_cluster/variables.tf

# var.node\_pools is a map of any

variable "node\_pools" {

type = map(any)

default = {}

}

實際 node\_pools input variable 內容物可能漲這樣

* 現階段先看這個 variable map 的 member，有兩個 member
* spot: {...}
* on-demand: {...}

node\_pools = {

spot = {...},

on-demand = {...}

}

然後回頭看 for\_each

* for\_each = var.node\_pools 指的是，為每個 node\_pools 的 member 產生一組 resource

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

for\_each = var.node\_pools

name = each.value.name

...

}

將 meta-argument 展開後，實際上會是

* resource block {} 原本應該 evaluate 對應一個 azurerm\_kubernetes\_cluster\_node\_pool resource
* 使用 for\_each meta-argument，讓 terraform 知道
* each.key 會變成 map member 的 key，ex spot, on-demand, ...
* each.value 會變成 map member 的 value，以目前這個 input 的 type，member value 會是 {...} 仍是一個 map
* each.value.name 會變成取得 member value，然後在嘗試 value 內部，取得 name 這個 field 的值

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

name = var.node\_pools.spot.name

...

}

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main[on-demand]" {

name = var.node\_pools.on-demand.name

...

}

至於實際 each.value.name 會取到什麼值，就依照各個 member value 去尋找

node\_pools = {

spot = {

name = "spot"

...

},

on\_demand = {

name = "on-demand"

...

}

}

最後變成，兩個 resource block

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

name = "spot"

...

}

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main[on-demand]" {

name = "on-demand"

...

}

**for each meta-argument pros & cons**

比較兩種寫法

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

}

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

name = "spot"

...

}

resource "azurerm\_kubernetes\_cluster\_node\_pool" "main[on-demand]" {

name = "on-demand"

...

}

好處

* 語法精簡
* 同類型的參數可以使用 default，不是重要的 input 不用給就可以使用 default
* 維護更新時，只要改一個 resource block，就全部的 resource 都更新了
* 使用 meta-argument 才可能管理更大量的 resource，ex. 10 個或 100 個相似的 resource

壞處

* 可讀性下降，需要人腦
* 參數取值複雜，需要一層一層 map 下去找參數
* 增加 debug 的難度

實務上我們都會選擇使用 for\_each meta-argument，犧牲可讀性換取精簡的程式碼

* 精簡的程式碼，維護上還是會有極大的好處
* 不用擔心 spot resource block 與 on-demand block 寫法不同，造成額外的問題

**For each limitation**

[Terraform for each meta-argument 也有許多限制](https://www.terraform.io/docs/language/meta-arguments/for_each.html#limitations-on-values-used-in-for_each)

* for\_each 的 variable 必須是 deterministic，意思是必須是定值
* 不能使用 conditional expression （ex. if else 或是三元判斷 ? ）
* 也不能倚賴不定值 function 的 results (ex. uuid, bcrypt, or timestamp...)，這些 function 的結果，會在 main evaluation 時延後計算，導致進入 main evaluation 時 for\_each 的參數其實仍是 undefined
* 原因也很簡單，需要在計算 meta-argument 時決定最終會有幾個 resource，如果不確定 resource block 數量，便無法計算下個 workflow 的內容

sentisive 的參數也無法使用在 for\_each 上

* for\_each 需要的可見度，會無法取得 sensitive 數值

**for each chaining**

* complex syntax [for each chaining](https://www.terraform.io/docs/language/meta-arguments/for_each.html#chaining-for_each-between-resources)

variable "vpcs" {

type = map(object({

cidr\_block = string

}))

}

resource "aws\_vpc" "example" {

# One VPC for each element of var.vpcs

for\_each = var.vpcs

# each.value here is a value from var.vpcs

cidr\_block = each.value.cidr\_block

}

resource "aws\_internet\_gateway" "example" {

# One Internet Gateway per VPC

for\_each = aws\_vpc.example

# each.value here is a full aws\_vpc object

vpc\_id = each.value.id

}

output "vpc\_ids" {

value = {

for k, v in aws\_vpc.example : k => v.id

}

# The VPCs aren't fully functional until their

# internet gateways are running.

depends\_on = [aws\_internet\_gateway.example]

}