**Terraform Module**

module 在 Terraform 中的定義很簡單，就是一個 container，裡頭有一組一起使用的 resource .tf。然而除了容納一組 resource 以外，module 還有須多額外的功能。

* directory 中至少一個包含一個 root module
* 可以調用其他的 module
* 可以透過 module input / output 傳值

**Example**

我們看一下 \_poc/container\_registry\_module 這邊的範例

module "test" {

source = "../../..//azure/modules/container\_registry"

registry\_name = "chechiatest"

resource\_group\_name = local.resource\_group\_name

location = local.location

public\_network\_access\_enabled = true

}

module "registry" {

for\_each = toset(local.environments) *# convert tuple to set of string*

source = "../../..//azure/modules/container\_registry"

registry\_name = "chechia${each.value}"

...

}

這裡使用 module block，來宣告一組 child module

* module 的程式碼來源，使用 source argument
* 其他的 input argument (例如 registry name)，則在底下宣告
* 由於有重複的 arguments，使用 locals block 宣告參數，然後使用 local. reference 到 local variable

**Module meta-argument**

使用另外一組 module block，來宣告另一組 child module

* 使用 for\_each [meta-argument](https://www.terraform.io/docs/language/modules/syntax.html#meta-arguments)，來宣告這組 module 裡面，由多個 instance 組成
* 使用 ${each.value}，將 registry name eval 成為 "chechiadev", "chechiastag", "chechiaprod" 三個名稱，給三個 instance 使用

**Module Init**

執行 Terraform init，會在 init

* 掃描 module 與 source 內容，檢查有無 module 設定錯誤（ex. 路徑錯誤找不到 source），
* 初始化 Backend，我們這邊是 remote Backend，會對遠端進行初始化
* 根據 modules 內部的內容，計算 module 需要的 providers 與 dependency
  + 由於我們這邊全部都是使用 azurerm provider，.terraform/providers 中只有 azurerm
  + 如果有多組 provider，會一併下載到 .terraform/providers 中

terraform init

Initializing modules...

- registry in ../../../azure/modules/container\_registry

- test in ../../../azure/modules/container\_registry

Initializing the backend...

Initializing provider plugins...

- Reusing previous version of hashicorp/azurerm from the dependency lock file

- Installing hashicorp/azurerm v2.65.0...

- Installed hashicorp/azurerm v2.65.0 (signed by HashiCorp)

...

我們可以看一下 .terraform 內容

* 多了一個 .terraform/modules 資料夾
* 多了一個 .terraform/modules/modules.json

cat .terraform/modules/modules.json | jq

{

"Modules": [

{

"Key": "",

"Source": "",

"Dir": "."

},

{

"Key": "registry",

"Source": "../../..//azure/modules/container\_registry",

"Dir": "../../../azure/modules/container\_registry"

},

{

"Key": "test",

"Source": "../../..//azure/modules/container\_registry",

"Dir": "../../../azure/modules/container\_registry"

}

]

}

* 第1個 module 是 "Dir": "."，也就是 \_poc/container\_registry\_module root module 本身
* 第2, 3個 module 是 ../../../azure//azure/modules/container\_registry module
  + 附上解析出來的相對路徑

如果 module 有任何新增刪除，或是設定改動，都需要重新 init，因為 module 的初始化在 init 步驟處理。

初始化完成，進行 plan，看看是否如我們預期

terraform plan

# module.registry["dev"].azurerm\_container\_registry.acr will be created

+ resource "azurerm\_container\_registry" "acr" {

+ location = "southeastasia"

+ name = "chechiadev"

# module.registry["prod"].azurerm\_container\_registry.acr will be created

+ resource "azurerm\_container\_registry" "acr" {

+ location = "southeastasia"

+ name = "chechiaprod"

# module.registry["stag"].azurerm\_container\_registry.acr will be created

+ resource "azurerm\_container\_registry" "acr" {

+ location = "southeastasia"

+ name = "chechiastag"

# module.test.azurerm\_container\_registry.acr will be created

+ resource "azurerm\_container\_registry" "acr" {

+ location = "southeastasia"

+ name = "chechiatest"

Plan: 4 to add, 0 to change, 0 to destroy.

plan 結果

* 產生一組 module.test.azurerm\_container\_registry.acr
* 產生一組 module.registry [object](https://www.terraform.io/docs/language/expressions/type-constraints.html#structural-types)，內容為三個 instance，依字母排序分別為
  + module.registry["dev"]
  + module.registry["prod"]
  + module.registry["stag"]

Terraform apply，terraform 便會平行化處理產生這四個 registry

terraform apply

...

module.registry["stag"].azurerm\_container\_registry.acr: Creating...

module.registry["dev"].azurerm\_container\_registry.acr: Creating...

module.registry["prod"].azurerm\_container\_registry.acr: Creating...

module.test.azurerm\_container\_registry.acr: Creating...

...

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

**root module**

[Terraform 官方文件](https://www.terraform.io/docs/language/modules/syntax.html) 對 module 的說明

* 每組資料夾中的 terraform configuration 至少有一個 module，意思是在 \_poc/container\_registry 中操作，本身便是一組 root module。

換句話說，當資料夾中只有一組，便不會產生 .terraform/modules/ 資料夾及檔案。這裡跟程式碼的實作比較有關，使用上不用額外注意。

**為何需要 Module**

隨著 Terraform 使用越久，我們會開始產生越來越複雜的 .tf 檔案，以滿足我們的需求，而這些需求有可能是類似的功能，不斷重複。例如：

\_poc/container\_registry 中，我們設定了一組 container registry。在現實的應用中，我們可能會產生多組 Registry，給不同服務使用。一個實際的例子：開發流程中產生多組功能相同的 infrastructure resources，作為測試環境與生產環境，除了 poc-registry 外，我們可能會產生 dev-registry，stag-registry，prod-registry，內容完全相同，承載開發流程中不同用途的軟體環境。

使用截至目前所學，可能是把 .tf 檔案複製多份，更改成名稱等參數，apply 上去就獲得多組不同的 registry。

resource "azurerm\_container\_registry" "acr" {

name = "chechia-poc"

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

location = "southeastasia"

}

resource "azurerm\_container\_registry" "acr" {

name = "chechia-dev"

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

location = "southeastasia"

}

resource "azurerm\_container\_registry" "acr" {

name = "chechia-stag"

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

location = "southeastasia"

}

resource "azurerm\_container\_registry" "acr" {

name = "chechia-prod"

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

location = "southeastasia"

}

這個做法十分直觀，而且確實能用。Terraform 對相同 directory 中的 .tf 數量也沒有限制，也就是說我們可以使用無限的複製貼上，來解決。

可以嘗試更改 \_poc/container\_registry 的資料夾，試著 apply。完全沒問題，是吧？

**Don't Repeat Yourself**

我們很快發現

* 上面這組 registry 除了檔案名稱以外，其他部分的內容都相同，導致 .tf 內容都是重複的
* 如果未來想要修改（ex. azure 發布新功能）我必須重複修改多次，浪費時間
* 另外一個 repository 也想使用相同的 resource block，複製過去等於是 hard fork，不會跟上這邊的更新

Don't Repeat Yourself (DRY)，是軟體工程中不同領域共通的最佳實踐。不斷重複的代碼，本身就代表而外的維護成本。那在 Terraform 中我們有沒有可能重複使用 .tf 中的內容？

**Issues**

使用的本地 module 會有一些問題

* module 程式碼重用
* module 版本鎖定

**Issues: module sharing**

開頭提到 module 可以方便程式碼重用，可以使用社群維護的 module，然而到目前我們還是無法使用社群維護的 terraform module，例如：

我想使用 [Azure 維護的 AKS module](https://github.com/Azure/terraform-azurerm-aks)，可以讓我直接建立 Azure Kubernetes Service  
目前使用本地 module 的話，我就要把整個 AKS module 裡面的 .tf copy 到本地 repository 中使用。如此確實可以運作，但 copy 等於失去遠端的 reference，如果後續遠端有更新，本地也很難使用

另外，本地的 module 很難給另外一個 repository 使用，例如：

我又開一個 github.com/chechiachang/terraform-30-weeks 的 repository，那要如何使用 terraform-30-days 中的 module？

我們寫成 module 是希望盡可能量重複使用相同程式碼，但希望是類似 git submodule 的方式，保留對遠端的 reference，有更新可以 git pull 下來使用

**Issues: module version locking**

module 會隨著使用持續修改，如果我今天希望修改 module 內容，但 module 已經在使用中了，該如何處理？例如：

開發 //azure/modules/container\_registry 中的新功能

* 希望在 module "test" 中測試，但
* 不要影響 module "registry" 上面的功能，特別是 stag / prod 環境，可能有其他團隊在測試

module "test" {

source = "../../..//azure/modules/container\_registry"

...

}

module "registry" {

for\_each = toset(local.environments) # convert tuple to set of string

source = "../../..//azure/modules/container\_registry"

...

}

以上面的 .tf 檔案，改了 //azure/modules/container\_registr 的話，所有使用到的內容都會一起改變。實務上會希望可以把現有的程式碼鎖在舊的版本，新的程式碼使用本地 module 繼續開發新功能。例如打 version tag：

module "registry" {

for\_each = toset(local.environments) # convert tuple to set of string

source = "../../..//azure/modules/container\_registry?ref=v0.1.0"

...

}

事實上，Terraform 不只支援本地的 module，還有許多類型的 module 可以解決上述問題（請見下章）

**Source code**

對於 terraform init module 的程式碼，有興趣請見 Terraform Github

* [terrafomr init](https://github.com/hashicorp/terraform/blob/24ace6ae7d68a7430a47d1d5d7991b5a1984ea97/internal/command/init.go#L307)
* [install module](https://github.com/hashicorp/terraform/blob/24ace6ae7d68a7430a47d1d5d7991b5a1984ea97/internal/command/meta_config.go#L186)
* [install module](https://github.com/hashicorp/terraform/blob/24ace6ae7d68a7430a47d1d5d7991b5a1984ea97/internal/initwd/module_install.go#L78)

**Homework**

* 閱讀 [Module 官方文件](https://www.terraform.io/docs/language/modules/index.html)
* 修改 \_poc/container\_registry\_module，使用其他的 module argument count，取代 for\_each，達成一樣的效果
* 完成一組 module，push 到 Github 上，使用 ssh 方式使用 remote module

**Remote module**

至此，我們已經學會使用 module 來重用(reuse) .tf 檔案，這邊只完成課程目標的一半。熟悉開源專案，我們很自然會想到，有沒有可以使用社群維護的 module，來讓我們使用。下堂課將分享，module 各種不同的 ㄩodule remote source，以及如何分享及使用遠端的 module？