**Reviews**

前六堂課程，我們簡單認識 Terraform 的核心觀念，包括 State & Backend，以及Module 使用，已經可以在工作上實際使用 Terraform。然而，隨著使用時間越長，使用的 module 越多，開始會發現許多 Terraform 的程式碼會不斷的重複，違反 DRY 原則，例如：

* provider.tf 每個 root module 都存在，而且內容幾乎一樣
  + 其中 backend 都是使用 azurerm，使用 storage container，設定只有 key 不同
  + 還記得 \_poc/container\_registry/ 這邊的範例，一堆 soft link provider.tf 嗎
* variables.tf 有許多重複的參數
  + 許多 module 都需要傳入 resource group 參數，而本篇所有的 example 都使用相同的 resource group
  + location 參數在相同 location 路徑下都是一樣的，ex. 都是"southeastasia"

Ex. Registry 的 input arguments 中許多參數，跟別的 root module 重複

*# \_poc/container\_registry/provider.tf*

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~> 2.65.0"

}

}

required\_version = ">= 1.0.1"

*# remote Backend*

backend "azurerm" {

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

storage\_account\_name = "tfstatee903f2ef317fb0b4"

container\_name = "tfstate"

key = "container\_registry.tfstate" *# 唯一不一樣的設定*

}

Ex. Registry 的 input arguments 中許多參數，跟別的 root module 重複

*# \_poc/container\_registry/registry.tf*

location = "southeastasia"

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

Don't Repeat Youself 是一個軟體工程開發的基本原則，不斷重複的程式碼往往代表未來維護的困難，違反 DRY 不一定代表不好，在某些情形工程師可能會選擇更好的可讀性，而犧牲 DRY。Terraform Lanuguage 由於語言特性，有一部份重複的程式碼。

這部分重複的程式碼，會隨團隊使用 Terraform 的規模線性成長，對於管理大量 Terraform 的維護人員造成困擾，也拖慢開發進度。

因此，在開始接觸大量複雜範例之前，我們選擇先導入 Terragrunt 這個工具，來精簡程式碼。

**Terragrunt**

[Terragrunt](https://terragrunt.gruntwork.io/) 是 gruntwork 推出的一個 Terraform thin wrapper，在執行 Terraform 前可以先"調整" root module 內的 .tf 檔案，保持程式碼的精簡，並提供許多而外工具，加速開發

[這裡附上 Gruntwork 官方的 Get Started Guide](https://terragrunt.gruntwork.io/docs/#getting-started)

**Install Terragrunt**

可以直接到 [Github release 頁面](https://github.com/gruntwork-io/terragrunt/releases" \t "_blank) 下載 binary

wget https://github.com/gruntwork-io/terragrunt/releases/download/v0.31.3/terragrunt\_darwin\_amd64

chmod +x terragrunt\_darwin\_amd64

sudo mv terragrunt\_darwin\_amd64 /usr/local/bin/terragrunt

如果使用 pkg manager 或其他工具，也可以直接使用

brew install terragrunt

安裝完可以使用 terragrunt

terragrunt --help

VERSION:

v0.31.3

**Config Terragrunt**

接下來我們要設定 terragrunt。由於Terragrunt 有非常多的功能，這邊我們先專注在兩個需求：

* provider / variable 程式碼精簡
* Backend / State 設定是否可以自動化

這邊我們直接看範例，首先要說明整體資料夾結構：

tree -L 1 azure

azure

├── \_poc

├── foundation

├── dev

│ └── southeastasia

├── modules

├── prod

├── stag

├── test

├── terragrunt.hcl

└── env.tfvars

* azure/\_poc 是使用 terragrunt 之前我們所有的範例
* 由於之後會介紹多環境的管理，所有的 .tf 都可以在不同環境，產生一模一樣的 resources，這邊先開啟四個環境，[可以參考 terrgrunt 文件：多環境的支援](https://terragrunt.gruntwork.io/docs/getting-started/configuration/" \l "formatting-hcl-files" \t "_blank)
  + foundation
  + dev
  + stag
  + prod
* modules 是本地維護的 local module
* test 是 .tf 的測試檔案，我們之後會講解如何為 .tf 撰寫測試
* dev/southeastasia 存放 dev 環境的 southeastasia
  + 許多 resource 放置在公有雲不同的 location
  + 實務也常見，在不同 location 產生相同 resource 做跨區 replicas 以支撐可用性
  + 接下來範例主要會使用 azure/dev/southeastasia

那剩下兩個東西是什麼？

* terragrunt.hcl
* env.tfvars

這是與 terragrunt 設定有關，請見底下的說明。

**Example: Terraform Backend**

首先我們可以在產生一組 terraform backend

* resource group: terraform-30-days
* 之前使用的可以選擇整個刪除掉 (resource group: terraform-30-days-poc)
* 這樣可以確保 resource gropu 內的資源都是 terragrunt 產生的，會比較乾淨
* 選擇要沿用 terraform-30-days-poc resource gropu 也不是不行

到 azure/foundation/southeastasia/terraform\_backend，一看，裡面只剩下一個檔案

# terragrunt.hcl

# TERRAGRUNT CONFIGURATION

terraform {

source = "../../../..//azure/modules/terraform\_backend"

}

# dependency cycle: terraform\_backend is provisioned before all terragrunt usage. There is no terragrunt.hcl at that time.

#include {

# path = "${find\_in\_parent\_folders()}"

#}

inputs = {

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

location = "southeastasia" # Or use japaneast

}

* source 大家應該很熟悉了，告訴 terragrunt 我們的 root module 路徑
  + Review: 在使用 terraform 而非 terragrunt 時，我們可以指定 local module path 來使用  
    odule，這邊原理類似
  + 要注意這邊的是 terragrunt 的 source 參數，雖然原理相同，但不等於 terraform 的 source
    - 要分清楚 .hcl 內部的是 terragrunt 的 config
    - 要分清楚 .tf 內部的是 terraform 的 config
* 由於使用 local module，這邊也透過 inputs 傳入

接下來進行 init 與 plan。我們把新的 terraform backend 使用 terragrunt 產生出來

terragrunt init

terragrunt plan

terragrunt apply

到這邊，使用起來跟直接使用 terraform 應該沒有差異，上面這個例子並沒有使用 terragrunt 的額外功能。terragrunt 單純把 terraform 的 command 傳遞下去，底下還是執行 terraform。

terragrunt 的功能，下個例子就會展現。

**Example: compute network**

網路 / VPC 網段的管理，是公有雲的必要工作，這邊以 provision 新的網段為例

到 azure/foundation/compute\_network，裡面有兩個檔案

tree

.

├── compute\_network.tf

└── terragrunt.hcl

provider.tf 在這邊已經消失了

* 由於幾乎每個 root module 的 provider.tf 都長一樣，我們希望可以讓 terragrunt 在執行 terraform 前動態載入
* 附上[terragrunt 關於 Keep your provider DRY 文件](https://terragrunt.gruntwork.io/docs/getting-started/quick-start/#keep-your-provider-configuration-dry)

看一下 terragrunt.hcl 的設定，重點在 include {} 這個 code block

* find\_in\_parent\_folders() 是 [terragrunt 提供的 function](https://terragrunt.gruntwork.io/docs/reference/built-in-functions/" \l "find_in_parent_folders" \t "_blank)
  + 會一路像上層資料夾，搜尋 terragrunt.hcl，並回傳絕對路徑
  + 這個例子就會變成：~/terraform-30-days/azure/terragrunt.hcl
* [include code block](https://terragrunt.gruntwork.io/docs/reference/config-blocks-and-attributes/#include) 是 terragrunt 繼承其他的 terragrunt.hcl 設定
  + 我們希望重複使用 provider.tf 的設定，所以把他放到上層 terragrunt.hcl 內部
  + 然後再用 include ，在 terragrunt command 時 (terraform command 前）動態載入

# terragrunt.hcl

# TERRAGRUNT CONFIGURATION

# use terragrunt function to include .hcl file

# in this case, will find azure/terragrunt.hcl

include {

path = find\_in\_parent\_folders()

}

terraform {

source = "../../..//azure/foundation/compute\_network"

# use double-slash (//) after repository root path to avoid

# - WARN[0000] No double-slash (//) found in source URL

#source = "."

}

底下的 terraform {} code block 則跟上一個例子一樣，指向 root module

* 仔細一看，../../../..//azure/modules/terraform\_backend 其實就是 . 也就是當前所在路徑
* 之所以使用較長的路徑，是為了幫助 terraform 找到 git repository 的 root path
* Review: Git remote module
  + 使用 local module，相對路徑不會影響 terraform 找尋 local module
  + 如果使用 git remote module，有沒有 double slash 就會影響 terraform 能否順利找到 module 路徑

接下來看上層 include 的 ~/terraform-30-days/azure/terragrunt.hcl 內容

首先是 provider.tf，這邊使用 [generate {} code block](https://terragrunt.gruntwork.io/docs/reference/config-blocks-and-attributes/#generate) 來產生 provider.tf

* 會在 source 的目錄（也就是執行 terraform 的目錄）下產生 provider.tf
* 如果要調整 provider.tf 的參數，這裡也支援使用 terragrunt 的 function 與變數，這邊先不使用

# azure/terragrunt.hcl

generate "provider" {

path = "provider.tf"

if\_exists = "overwrite\_terragrunt"

contents = <<EOF

provider "azurerm" {

features {}

}

EOF

}

再來產生的事 backend.tf，使用 [remote\_state {} code block](https://terragrunt.gruntwork.io/docs/reference/config-blocks-and-attributes/" \l "remote_state" \t "_blank) 設定 remote backend

* Review: 我們使用 azurerm + storage container
* 這邊使用 generate，原理與 generate block 相同，在 terraform 的 root module 內產生 backend.tf
* 在 backend.tf 內設定 storage container 的參數

# azure/terragrunt.hcl

remote\_state {

backend = "azurerm"

generate = {

path = "backend.tf"

if\_exists = "overwrite\_terragrunt"

}

config = {

key = "${path\_relative\_to\_include()}/terraform.tfstate"

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

storage\_account\_name = "tfstate445d2966b56b5d05"

container\_name = "tfstate"

}

}

[path\_relative\_to\_include()](https://terragrunt.gruntwork.io/docs/reference/built-in-functions/#path_relative_to_include) 是另一個 terragrunt function

* 與 include {} 搭配使用，回傳"從目前的 terragrunt.hcl 到 include{} terragrunt.hcl 路徑的相對路徑"
* 目前 azure/foundation/compute\_network/terragrunt.hcl
* include azure/terragrunt.hcl
* 這個範例 path\_relative\_to\_include() = foundation/compute\_network

├── terragrunt.hcl

├── env.tfvars

├── foundation

│ ├── compute\_network

│ │ ├── compute\_network.tf

│ │ └── terragrunt.hcl

所以整個效果等同於產生一個 backend.tf

*# backend.tf*

terraform {

backend "azurerm" {

key = "foundation/compute\_network/terraform.tfstate"

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

storage\_account\_name = "tfstate445d2966b56b5d05"

container\_name = "tfstate"

}

}

為何 key 要設為 foundation/compute\_network/terraform.tfstate

* 我們希望 terraform.tfstate 放到 azure storage blob 中，也能按照一定的邏輯存放，方便管理
* 所以使用 terragrunt.hcl 彼此的相對位置
* foundation/compute\_network/terragrunt.hcl
* 產生的 state 就在 blob//foundation/compute\_network/terraform.tfstate

最後一段 [terraform {} block](https://terragrunt.gruntwork.io/docs/reference/config-blocks-and-attributes/#terraform)，可以在 terragrunt 驅動的 terraform command 做許多調整，例如

* 這邊增加 extra\_arguments
  + [get\_terraform\_commands\_that\_need\_vars()](https://terragrunt.gruntwork.io/docs/reference/built-in-functions/#get_terraform_commands_that_need_vars)，回傳一串接受 -var 與 -var-file 參數的 terraform command
  + required\_var\_files 參數把 env.tfvars 檔案，作為 terraform -var-file 的參數
  + 搭配 [get\_parent\_terragrunt\_dir()](https://terragrunt.gruntwork.io/docs/reference/built-in-functions/" \l "get_parent_terragrunt_dir" \t "_blank) 使用，拿到上層 terragrunt.hcl 的絕對路徑
  + 然後讀取這個檔案 ~/terraform-30-days/azure/env.tfvars，作為 -var-file 的參數

*# azure/terragrunt.hcl*

terraform {

extra\_arguments "env" {

commands = get\_terraform\_commands\_that\_need\_vars()

required\_var\_files = [

"${get\_parent\_terragrunt\_dir()}/env.tfvars",

]

}

}

效果等同在 terraform plan 與 apply （以及其他 command) 執行時，多加參數

* env.tfvars 裡面的 variable 就會是所有 root module 中執行 terraform command 時都吃得到
* 只需要維護一組 env.tfvars

terraform plan -var-file ~/terraform-30-days/azure/env.tfvars

terraform apply -var-file ~/terraform-30-days/azure/env.tfvars

**Cache**

terragrunt 會將 terraform module cache 一分在本目錄，[cache](https://terragrunt.gruntwork.io/docs/features/caching/" \t "_blank)

如果是在本地開發中的 module，有可能會 cache 到錯誤的 module，請把本地 cache 清除再重新 init

rm -rf .terragrunt-cache

terragrunt init

**Summary**

terraform 的命令在 terragrunt 上完全都能使用，所以才說 terragrunt 是一層 wrapper，意思是

* terragrunt 只是在執行 terraform command 前，先對 .tf 檔案動一些手腳
* 對 terraform command 動一些手腳

**Pros & Cons**

使用 Terragrunt 是一個額外的選擇，團隊可以依據狀況去選擇

Pros

* 精簡程式碼，包含 provider / backend ...
* runtime 注入變數
* 在 terraform 的 lifecycle 之前，與之後執行額外的程式

Cons

* 程式碼會變得更複雜
* 需要額外注意 terragrunt 與 terraform 之間的 lifecycle
* 不熟悉時 debug 可能造成一些麻煩