**tfsec**

工程師應該注意 infrastructure 的安全性，然而並非人人都是資安專業背景，不一定都能捉到設定上資安風險。這時就要依賴外部的檢查資料庫，根據常見的安全性錯誤進行檢查。tfsec 是一個很好的免費開源工具，針對 terraform 的 .tf 檔案，針對 plan 直接進行分析，挑出安全性錯誤。底下介紹如何搭配 terragrunt 使用 tfsec。

* <https://tfsec.dev/>
* <https://github.com/aquasecurity/tfsec>

**Install tfsec**

sudo port install tfsec

sudo brew install tfsec

tfsec *--version*

0.57.0

**Run**

找尋 root module 直接運行 tfsec

* 然而由於我們有使用 terragrunt 做一層 wrapper，在執行 terragrunt 直接執行 tfsec 的話會缺乏許多參數跟檔案
* 會掃描到 .terragrunt-cache 的檔案，這些檔案是外部下載的 module
* 使用 --exclude-downloeaded 也只會 exclude .terraform

tfsec .

tfsec --exclude-downloaded-modules

為了避免以上的問題，可以使用 [terragrunt before hook](https://terragrunt.gruntwork.io/docs/features/before-and-after-hooks/)。這裡我們隨意拿一個 root module 作為範例

# azure/foundation/compute\_network/terragrunt.hcl

terraform {

...

before\_hook "tfsec" {

commands = ["apply", "plan"]

execute = ["tfsec", "."]

}

}

實際的效果

*# azure/foundation/compute\_network/terragrunt.hcl*

cd azure/foundation/compute\_network

terragrunt apply

Initializing modules...

Initializing the backend...

Initializing provider plugins...

Terraform has been successfully initialized!

INFO[0010] Executing hook: before\_hook prefix=[/Users/che-chia/my-workspace/terraform-30-days/azure/foundation/compute\_network]

times

------------------------------------------

disk i/o 2.952161ms

parsing HCL 29.157µs

evaluating values 1.307087ms

running checks 1.486024ms

counts

------------------------------------------

files loaded 7

blocks 8

evaluated blocks 26

modules 1

module blocks 18

results

------------------------------------------

critical 0

high 0

medium 0

low 0

ignored 0

No problems detected!

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration

and found no differences, so no changes are needed.

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

只要每一次 apply 前，都會進行 tfsec

**Move hooks to parent directory**

如果希望所有

* Review: 已具我們[前幾章的 terragrunt 設定](https://ithelp.ithome.com.tw/articles/10262695)操作 root module 時都會 terragrunt 向上尋找 import {}，並 import 上層的 terrgrunt.hcl。我們可以將 hook 寫在上層 terragrunt.hcl，所有底下的 root module 都會生效。

使用注意 hook 與 git pre-commit hook，會需要時間，影響開發效率，可以依據團隊的狀況做調整

* 例如頻繁開發的功能，還需要大量的 plan 除錯，就不需要每次 plan 都檢查
* 如果不放在 terraform hook 裏，而是移到 CI/ CD 上執行 tfsec，可以保持開發進度
  + 壞處就是在 CI / CD 上掃出安全性漏洞的話，就要拉回來重新執行 PR
* 放不放 hook？放 terragrunt hook，或是 pre-commit hook，或是 CI 中檢查，團隊需要多加溝通，多嘗試，持續調整，才能達到團隊最佳效益。

範例是 after hook，只要每一次 apply 完成後，做一次 tfsec，讓工程師測試時就可以檢查安全性問題

# azure/terragrunt.hcl

terraform {

...

after\_hook "tfsec" {

commands = ["apply", "plan"]

execute = ["tfsec", "."]

}

}

**tfsec checks**

tfsec 檢查清單可以到 [tfsec.dev](https://tfsec.dev/) 查閱。所有的檢查內容都有附上原因，風險說明，問題範例以及改進範例。例如 [azure storage 設定風險](https://tfsec.dev/docs/azure/storage/use-secure-tls-policy/#azure/storage)

tfsec 就是社群維護的安全守則，善用 tfsec 可以實現安全最佳實踐，也同時提升自己的資安知識。

**ignore warnings**

當掃出問題時，我們不一定能馬上解決，也許是排時程稍後再修理，也許是被其他因素影響，暫時無法改正。然而每次掃瞄 tfsec 還是會跳出警告。收到警告，但大家又不會馬上修改，就會無謂的警告，浪費團隊的精神能量，消耗無謂的注意力。[tfsec 提供 ignored 標記](https://github.com/aquasecurity/tfsec#ignoring-warnings)

* 掃描發現問題
* 把已知問題紀錄 Issue Tracking 系統（Github Issue 或 Jira）
* 在 .tf 程式碼中可以做標記，打上 Issue number / url 以供雙向追蹤

**Terragrunt multple workspaces**

在 Terraform 中，我們會 change directory 到一個一個 root module 中去執行 init, plan, apply 等工作。當 root module 數量很多的時候，這件事就變得很複雜。這時可以利用 [Terragrunt 提供同時多 workspace 執行](https://terragrunt.gruntwork.io/docs/features/execute-terraform-commands-on-multiple-modules-at-once/) 的功能，一次控制多個 root module。例如以 azure/dev 為例，資料夾樹狀結構如下

tree azure/dev

.

├── env.tfvars

├── japanwest

└── southeastasia

├── container\_registry

│ └── terragrunt.hcl

└── env.tfvars

cd foundation

terragrunt run-all init

*#terragrunt run-all init --reconfigure*

INFO[0000] Stack at /Users/che-chia/my-workspace/terraform-30-days/azure/foundation:

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/foundation/compute\_network (excluded: false, dependencies: [])

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/foundation/service\_principal (excluded: false, dependencies: [])

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/foundation/southeastasia/terraform\_backend (excluded: false, dependencies: [])

Initializing modules...

Initializing the backend...

Initializing the backend...

Initializing the backend...

也可以進行 run-all plan

* 每個 module 都會執行 plan

terragrunt run-all plan

INFO[0000] Stack at /Users/che-chia/my-workspace/terraform-30-days/azure/dev:

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/dev/japanwest/container\_registry (excluded: false, dependencies: [])

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/dev/southeastasia/container\_registry (excluded: false, dependencies: [])

...

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:

+ registry\_login\_server = (known after apply)

...

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:

+ registry\_login\_server = (known after apply)

然而，run-all 搭配 apply，要特別注意

* 這邊是先跳出確認，讓你盲眼 apply，在顯示 plan 時 -auto-approve
* 這邊是先跳出確認，讓你盲眼 apply，在顯示 plan 時 -auto-approve
* 這邊是先跳出確認，讓你盲眼 apply，在顯示 plan 時 -auto-approve
* 工作流程與 terraform 有所不同，需要使用者自行 run-all plan 檢視變更
* 注意不要不小心 apply 錯誤的 .tf 檔案

terragrunt run-all apply

INFO[0000] Stack at /Users/che-chia/my-workspace/terraform-30-days/azure/dev:

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/dev/japanwest/container\_registry (excluded: false, dependencies: [])

=> Module /Users/che-chia/my-workspace/terraform-30-days/azure/dev/southeastasia/container\_registry (excluded: false, dependencies: [])

Are you sure you want to run 'terragrunt apply' in each folder of the stack described above? (y/n)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

**Terragrunt module dependency**

有些 module 其實是有依賴性，需要依照先後順序 apply 到公有雲上，後面的 module 才能夠正常 provision。例如 compute VM 其實依賴 compute network 的 subnet id，才能把 compute VM 的 ip 分配到 subnet 上。實際的例子請見 azure/dev/southeastasia/chechia\_net/compute

*# azure/dev/southeastasia/chechia\_net/compute/terragrunt.hcl*

dependency "network"{

config\_path = find\_in\_parent\_folders("azure/foundation/compute\_network")

}

inputs = {

...

vnet\_subnet\_id = dependency.network.outputs.vnet\_subnets[0] *# dev-1*

}

inputs 中有個參數是 vnet\_subnet\_id 意思是這台 VM 應該使用指定的 subnet

* 細節 [可以參考 Azure: Linux VM private ip](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-static-private-ip-arm-pportal?context=/azure/virtual-machines/context/context)
* 如果是透過 web portal 操作，我們會先去 networking 新增 network -> subnet，取得 subnet id，建立 VM 的時候指定給 VM

這一個先後的動作，就是元件的依賴性

* 依賴性不只在於 provision 時有先後順序，所有的改動都應注意是否會影響下游依賴的服務，有可能不小心改了 network，network 正常，但是底下 VM 卻有功能損壞。所謂牽一髮而動全身
* 透過 portal 操作，我們自然就使用工人智慧，手動的維持依賴上游元件的穩定

使用 terraform ，[terraform 有提供 module depends\_on 的 meta-argument](https://www.terraform.io/docs/language/meta-arguments/depends_on.html)，讓 .tf 中可以描述 module 與 module 之間的關係

terragrunt 中提供的 dependency，進一步將這層依賴性，推廣到 root module 之間也能建立依賴性，[Terragrunt modules dependency](https://terragrunt.gruntwork.io/docs/features/execute-terraform-commands-on-multiple-modules-at-once/#dependencies-between-modules)，具體是有什麼差異：

* 在 terragrunt.hcl 中 dependency{} 宣吿有依賴別的 module，並把 config\_path 指向
  + Review: 這邊使用 terragrunt built-in function find\_in\_parent\_folders 來尋找
* 在 inputs = {} 中，使用 dependency 的 attibute 來取得 azure/foundation/compute\_network module 中的 output
  + 使用 output 來作為 input，vnet\_subnet\_id
  + 前提是 azure/foundation/compute\_network module 要能 output 需要的值

建立起這層依賴後對於 terragrunt 工作流程的影響是

* plan chechia\_net/compute 前，會先去取得 foundation/compute\_network state 中的 output
  + 一方面確認 network 的狀態
  + 也將 output 跨越 root module 傳遞
* 如果取得 foundation/compute\_network state 中有發現問題，則 chechia\_net/compute 的 plan 與 apply 會終止
  + 這符合預期，上游依賴的服務有問題，依賴的服務再加上去往往無法正常運作

**Pros & Cons**

Pros

* 明確宣告 module 之間的依賴性
* 動態取得其他 root module 的最新參數，而不是 hard-code 在 input
  + 對於可能遠端變動的 output 非常好用

Cons

* 運算每次 plan 前要先去取得其他 module 的 state，數量多時 plan 就會拖慢
* 提升穩固性，限制人為更改的程度，某方面也犧牲改動的彈性

實務建議：dependency 請適量使用，不用全部有關的依賴都套用上去，也不要都不用

* 重要的核心依賴（例如：一出問題底下就爆炸的）依賴可以加在 dependency 中，每次 apply 多做檢查
* 不重要但是相關的依賴，可以直接使用 hard-code 寫在 inputs 中
  + 雖然違反 clean code，但卻維持住 root module 間的 loose coupling，彼此改動不會被限制住

**References**

* <https://github.com/aquasecurity/tfsec>