**test your code**

測試是程式碼開發的一環，對於程式碼的品質影響巨大，這裡不提測試的概念。總之，測試非常重要。

以 terraform 而言，有許多時候都會需要測試

* 測試一個 terraform module 功能
* 測試一個 root module 產生的 infrastructure 是否符合預期
* 測試一段 .tf 修改是否會破壞現有功能
* ...

那應該如何測試 terraform?

**testing for terraform**

[官方 blog 對於測試 terraform 的文件描述](https://www.hashicorp.com/blog/testing-hashicorp-terraform)

將團隊選擇測試策略，依照成本（時間成本與費用）排序，會是個階層金字塔

* unit test
* contract test
* integration test
* end-to-end test
* manual test

基於這個基礎想法，分別討論如何測試 terraform

**unit test**

terraform fmt -check

terraform validate

實作有幾個選項

* git pre-commit hook
* terragrunt [Before and after hook](https://terragrunt.gruntwork.io/docs/features/before-and-after-hooks/)
* terraform plan

**contract test**

檢查 module 的 input 與 input format

* 最基本的是給予 input variable type，而不要使用 any / object
* 接著是使用 variable block 中的 validation {}，為參數設定 validation，排除意外的參數

[terraform 在 variable 中提供 custom validation rule 的功能](https://www.terraform.io/docs/language/values/variables.html#custom-validation-rules)

* 絕大多數的公有雲 API 對於 request argument 都有限制，可以在官方文件查找，如果 terraform 使用不合法的參數，在 apply phase (api request 出去後) 會收到 error
* 依據 api 限制，與業務需求設置 validation 可以讓 validate / plan phase 就出錯
* 軟體工程中的fail-fast 原則

variable "listener\_rule\_priority" {

type = number

default = 1

description = "Priority of listener rule between 1 to 50000"

validation {

condition = var.listener\_rule\_priority > 0 && var.listener\_rule\_priority < 50000

error\_message = "The priority of listener\_rule must be between 1 to 50000."

}

}

contract test 可以延伸，所有在 terraform apply 之前，針對 content / format / input / output 的檢查都可以

**integration test**

整合測試針對 terraform apply 的結果做測試，也就是 terraform module 是否正確的產出 infrastructure

* 透過 test framework，實際對公有雲發出 api request
* 實際 apply，但是放在獨立的測試空間
* 使用測試用的 name 與 id，來與實際的環境隔離，(ex. 使用 dev / stag /prod 以外的環境產生 infrastructure)
* 使用測試用的參數可能是 testing name / id 或是 function 產生的 (ex. random name / id)
* 根據 apply 後的 result 做測試
* apply 後，destroy 產生的 infrasturcture

使用 terraform test framework

* [Terratest](https://terratest.gruntwork.io/)
* 這個下一堂會示範如何使用 terratest framework

**End-to-End test**

當 terraform apply 後，產生的 infrastructure，使用者是否能正常使用

這個層級的測試需要導入使用者的測試例，會需要 QA 團隊協助

* 如果 infrastructure 上部署 app，也需要 app 團隊

如何對 terraform 做 end-to-end test

* 可以鎖住穩定的 app 版本
* 使用新版的 terraform 在獨立的環境 apply
* 執行 QA team 的 end-to-end 測試，在 app 不變的狀況下，改變 infra 是否會改變測試結果

End-to-End 通常會花費需多時間，但對於整體環境是非常必要的

**Manual test**

要如何手動測試 terraform？

如果是 app，我們會手動測試其功能

* ex. 一個網站，連線到新版網站，並手動操作功能，來進行手動測試

回到 infrastructure，如何手動測試 infrastructure 的功能？

* 針對公有雲的 infrastructure，除非我們發現奇怪的錯誤訊息，不然一般都會選擇穩定的 infra 產品，並相信公有雲的文件，也相信產生出的 infra 品質
* ex. 把 aws ec2 生出來後，通常不會需要進去手動測試，他就是一個功能正常的 vm
* 更多時候 terraform 撐出來的 infrastructure 出錯，多半是 terraform resource argument 填錯，產生意料之外的 infra

因此對 terraform 而言，針對嵾數做 variable validation 是十分有效的

**summary**

筆者建議，做少少的努力就可以獲得明顯成效的方法

* terraform format + validation
* auto terraform plan + apply
* variable custom validation

app 產品穩定的話，應該有 end-to-end test，善用 QA 團隊既有的 end-to-end test

* 鎖住 app 在穩定版本
* 最好在上 stag / prod 之前，都能完成完整的 end-to-end test
* 如果產品規模太大， end-to-end 跑不完的話，可以針對修改部分進行測試
* 例如更改 network infra，就要求 QA 在新環境執行 networking 相關的 end-to-end test

完成以上內容，已經滿足一個『能夠正常乘載 app 的 infra 的需求了』。當然，實務上只是滿足需求不是終點，還有非常多可以調整優化的地方

為 terraform 額外寫 integration test，可以進一步提升 module 品質，降低成本與提升效能

* terratest integration test 可以達到 module 品管
* 可以針對複雜的 module 做，特別是 dependency module 多，階層複雜，參數複雜的 module
* 簡單的 module ex. 只有一個 resource，沒有複雜依賴關係的 module 就不需要寫 terratest，寫了變成測試比 code 多太多