**debug**

在 terraform 過程中，我們會遇到許多錯誤

最多的是 hcl 設定，也就是編輯 .tf 內容與 terraform 互動時，所遇到的錯誤

* 可能是 .tf 寫錯導致錯誤，或是結果不如預期

其次是 state 的錯誤

* 由於 terraform 仰賴 state 來做 .tf resource 與 remote resource 的對照，state 錯誤會導致 terraform 行為不正常
* state 偶爾會因為其他錯誤原因導致 out of sync，可能導致 terraform 錯誤的產生或刪除 resource

terraform core error

* 指的是 terraform 本身的程式碼出錯了，這時就需要到 terraform github 上提交 issue，讓 terraform core 團隊來進行修復
* 從使用者的角度， 使用穩定版本比較不容易遇見 bug，然而使用的是 edge 版本就有機會遇到
* 當 terraform 行為怪怪的，我們也需要進行錯誤排除，才能確定是 terraform core 的錯誤，而不是 hcl 寫錯

provider error

* terraform core 本身提供抽象的運作邏輯，實際與公有雲 api 互動的是各家的 provider，provider 本身有會有 bug
* 以筆者經驗，provider 還蠻容易有機會遇到 bug，特別是公有雲剛出的新功能，provider 剛支援時
* 這時我們一樣要走 debug workflow，才能確定是 provider 的問題
* 以及在 provider 尚未修復前，我們要如何繼續正常使用 terraform

更多細節可以參考[hashicorp learn 文件中建議的 trouble shooting](https://learn.hashicorp.com/tutorials/terraform/troubleshooting-workflow)

**terraform debug steps**

以上面這篇文章作為範例，hashicorp 建議的 debug 步驟

* terraform fmt
* terraform validate
* fix terraform version
* fix .tf code
* debug variable / for\_each

由於 hcl 的錯誤是最常見的，也就是使用者 .tf 寫錯，因此 debug 時我們會先 validate .tf code

* terraform fmt 將 .tf format 統一
* terraform validate 驗證 .tf context 是不是符合 provider 的規格
* terraform version，一般來說我們使用的 terraform 版本會固定，例如本 repo 所有的範例都使用相同的 terraform 版本，降低遇見新版 terraform 錯誤的機率

其中 fmt, validate 很常使用，因此可以整合到工作流程中

* 例如 terragrunt before & after hook
* 至少務必整合到 git pre-commit hook，確保 PR 與 master code 的整潔

**Terragrunt console**

有些時候，terraform 可以正常運作不出錯，可是產生結果不如我們預期，可能就是 .tf 參數寫錯。然而，複雜module 中的參數十分複雜很難除錯，這時我們會使用 terraform console 協助

我們使用用到爛的 azure/foundation/compute\_network 做範例，如果今天 subnet 建立出來後，發現參數怪怪的（ex. address space 很怪）

* 我們要如何檢查 terraform expression 實際 evaluating 後，各個 variable 的參數？
* 由於我們使用 terragrunt，一樣先要使用 terragrunt 帶入 console 指定
* 附上[terraform console 的官方文件](https://www.terraform.io/docs/cli/commands/console.html)

cd azure/foundation/compute\_network

terragrunt console

>

> help

The Terraform console allows you to experiment with Terraform interpolations.

You may access resources in the state (if you have one) just as you would

from a configuration. For example: "aws\_instance.foo.id" would evaluate

to the ID of "aws\_instance.foo" if it exists in your state.

Type in the interpolation to test and hit <enter> to see the result.

To exit the console, type "exit" and hit <enter>, or use Control-C or

Control-D.

進入 console 後，顯示互動式令令列

* 此時的 .tf expression 已經 evaluated，也就是參數都有靜態數值可以顯示
* 除了 apply 之後的才會確定的參數，其他參數都可以顯示
* 會讀取已經存在的 state

可以印出帶入的 input 參數

> module.network

{

"vnet\_address\_space" = tolist([

"10.2.0.0/16",

])

"vnet\_id" = "/subscriptions/6fce7237-7e8e-4053-8e7d-ecf8a7c392ce/resourceGroups/terraform-30-days/providers/Microsoft.Network/virtualNetworks/acctvnet"

"vnet\_location" = "southeastasia"

"vnet\_name" = "acctvnet"

"vnet\_subnets" = [

"/subscriptions/6fce7237-7e8e-4053-8e7d-ecf8a7c392ce/resourceGroups/terraform-30-days/providers/Microsoft.Network/virtualNetworks/acctvnet/subnets/dev-1",

"/subscriptions/6fce7237-7e8e-4053-8e7d-ecf8a7c392ce/resourceGroups/terraform-30-days/providers/Microsoft.Network/virtualNetworks/acctvnet/subnets/dev-2",

"/subscriptions/6fce7237-7e8e-4053-8e7d-ecf8a7c392ce/resourceGroups/terraform-30-days/providers/Microsoft.Network/virtualNetworks/acctvnet/subnets/dev-3",

]

}

可以印出 object 的 output，例如 module.network 的 output.vnet\_id

> module.network.vnet\_id

"/subscriptions/6fce7237-7e8e-4053-8e7d-ecf8a7c392ce/resourceGroups/terraform-30-days/providers/Microsoft.Network/virtualNetworks/acctvnet"

也可以在 console 中做實驗

* 在檢查複雜的參數很方便，例如 for expression，或是 for\_each count meta-argument 的產物

> 1+6

7

> [for k in [1,2,3,4,5] : k]

[

1,

2,

3,

4,

5,

]

結束後 exit console，釋放 state lock

> exit

Releasing state lock. This may take a few moments...

**terraform log level**

如果檢查變數都沒問題，但結果仍然有問題，我們就會懷疑是 provider，或是外部條件（網路環境）甚至是 core 的錯誤，這時可以打開 terraform debug log 來檢查

* TF\_LOG 支援的的 log level，TRACE, DEBUG, INFO, WARN or ERROR
* 由於 debug level 的輸出量已經很大，建議輸出到檔案，再用編輯器仔細檢視

cd azure/foundation/compute\_network

export TF\_LOG=DEBUG

TF\_LOG=DEBUG terragrunt plan | tee plan\_debug.log

...

vim plan\_debug.log

內容有 terraform 自身更核心的 log，以及與 public cloud 互動的資訊

**Terragrunt debug**

上面都是 terraform debug，由於使用 terragrunt 做了很多事情，我們有時也會因為錯誤的設定而需要 debug terragrunt

我們可以調整 terragrunt log level 或是啟用 debug mode

terragrunt apply --terragrunt-log-level debug --terragrunt-debug

細節請見 [terragrunt debugging 說明](https://terragrunt.gruntwork.io/docs/features/debugging/)