

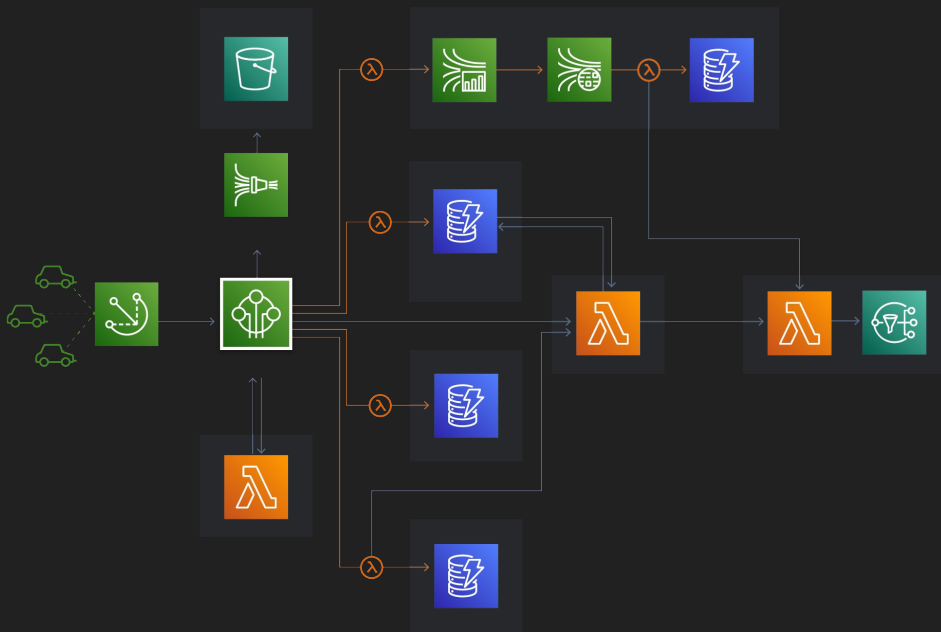
用程式碼管理你的雲端服務

Allen



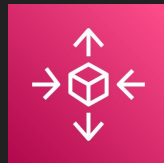
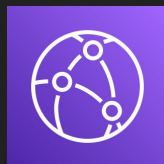
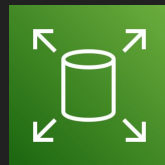
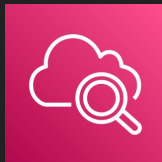
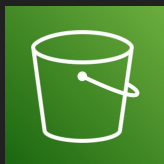
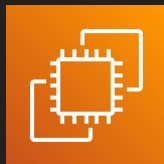
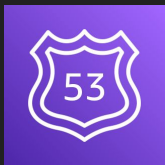
使用雲端服務可能會遇到的困擾

- 一個完整的架構可能會包含很多不同類型的雲端服務

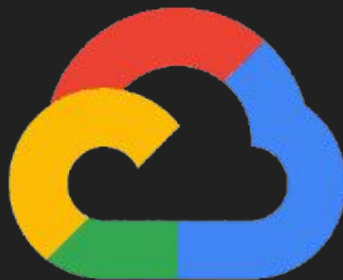
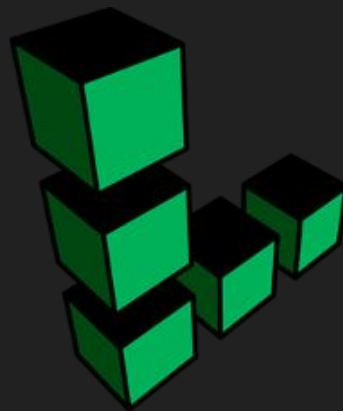


當你不再需要使用這些雲端服務時

- 你需要一個一個把這些服務刪除，不然這些服務會慢慢刪除你錢包中的錢
- 但你還記得你用了哪些服務嗎？



更別提如果你們公司用了不只一間雲端廠商



雲端服務是很貴的，某人曾說過 ...



雲端資源的花費可以分兩種

你開起來的

和你忘記關的

- 魯迅 (沒說過)

簡單介紹 Terraform

- 為了避免資源開了忘記關，你可以使用程式碼更輕鬆的管理你的雲端資源，即 IaC (Infrastructure as Code)



HashiCorp

Terraform

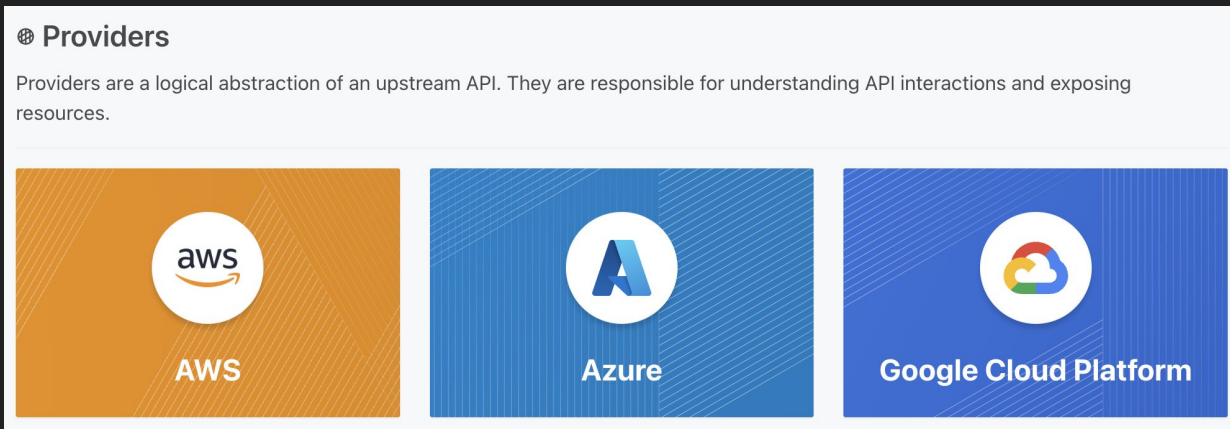
HCL (Hashicorp Configuration Language)

- 是 Terraform 使用的一種宣告式語言
- 你可以使用 HCL 去描述你想要建立的雲端資料
- 如果想要建立一個 VPC (Virtual Private Cloud), 你可以這麼寫



Provider

- Provider 提供了一個中介層。Terraform 可以透過這個中介層操作雲端的 API 來建立或刪除雲端資源
- 除了目前最知名的三雲 (AWS、Azure 與 GCP), Terraform 有許多 Provider 可以使用



根據 GitHub 2022 年度調查

- HCL 是目前成長最快速的語言

The fastest growing languages

The Hashicorp Configuration Language (HCL) saw significant growth in usage over the past year. This was driven by the growth in the popularity of the Terraform tool and IaC practices to increasingly automate deployments (notably, Go and Shell also saw big increases).

Additionally, Rust saw a more than 50% increase in its community, driven in part by its security and reliability. And Python continued to see gains in its usage across GitHub with a 22.5% year-over-year increase driven, in part, by its utility in data science and machine learning.

GROWTH IN PROGRAMMING LANGUAGES 2021-2022

01 HCL	56.1%
02 Rust	50.5%
03 TypeScript	37.8%
04 Lua	34.2%
05 Go	28.3%
06 Shell	27.7%
07 Makefile	23.7%
08 C	23.5%
09 Kotlin	22.9%
10 Python	22.5%

所以你應該要學 Terraform 嗎？



Terraform 的好處

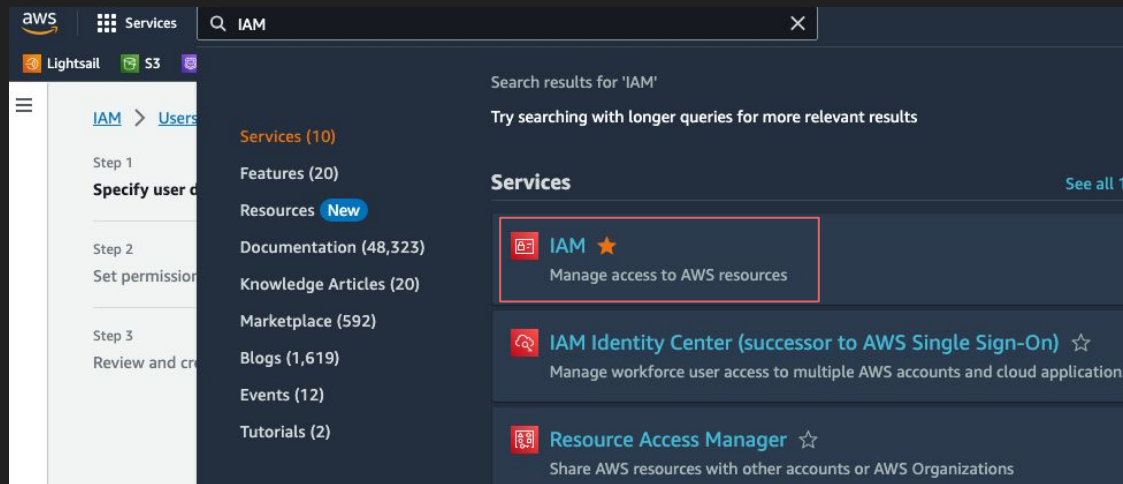
- 減少管理上的負擔與成本：確實記錄你建立的每一筆資源，不會等到帳單出來了才發現忘記關掉某個不需要的資源
- 跨平台：只要有 Provider，你就能透過 Terraform 操作不同平台的資源
- 自動化流程：使用 CI/CD 部署你的資源

在使用 Terraform 建立資源之前, 你必須先準備

- 一個可以讓 Terraform 操作資源的權限
 - 如果你是用 AWS, 需要建立一個 IAM, 然後用 AWS CLI 在本地設定好權限後給 Terraform 使用
 - 如果你是用 GCP, 需要使用 gcloud CLI 在本地建立一個 Credential 給 Terraform 使用
 - 如果你是用 Azure, 需要使用 Azure CLI 建立一個 Credential 給 Terraform 使用

建立一個 IAM 給 Terraform 使用 (Part 1)

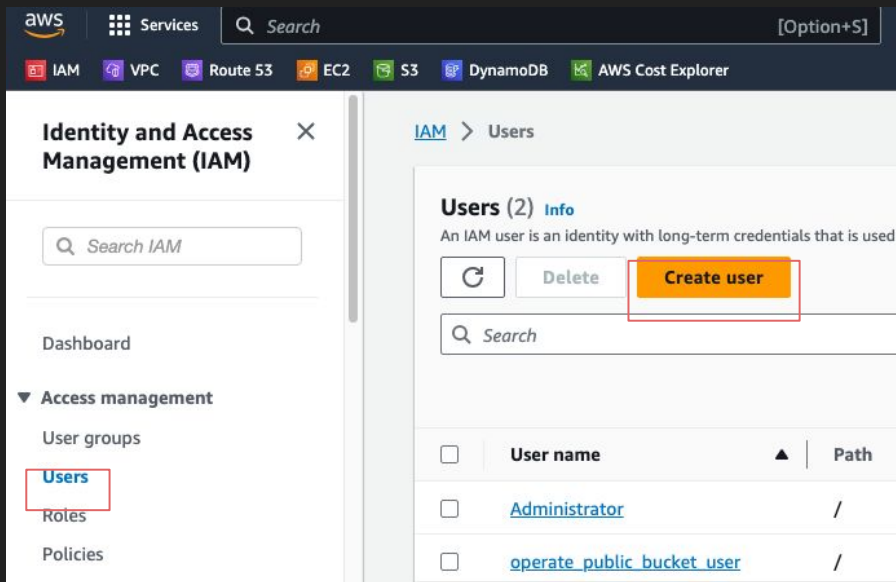
- 首先先申請一個 AWS 的帳號，登入後搜尋服務 IAM (Identity and Access Management)



* 新帳號有一年的 Free Tier, 部分服務在一定的用量 內會是免費, 可參考 [官方文件](#)

建立一個 IAM 給 Terraform 使用 (Part 2)

- 來到 IAM 的服務頁面後，開始來新增一個 IAM user



* IAM user 不會產生任何費用

建立一個 IAM 給 Terraform 使用 (Part 3)

- 給 IAM user 一個名稱, 然後 Next

Step 1

Specify user details

Step 2

[Set permissions](#)

Step 3

Review and create

Specify user details

User details

User name

terraform

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

i If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Cancel

Next

建立一個 IAM 給 Terraform 使用 (Part 4)

- 設定最高權限給 IAM user

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

☐ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

☒ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1123)

Choose one or more policies to attach to your new user.

Filter by Type All types 37 matches < 1 2 >

<input type="checkbox"/>	Policy name	Type	Attached entities
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	1

* 這邊設定最高權限只是為了方便，正式環境盡量別這麼做，請依照會開的資源類型去設定對應的權限

建立一個 IAM 給 Terraform 使用 (Part 5)

- 建立完成後，就可以在列表上看到我們剛剛建立的 user，請點進去開始申請 access key

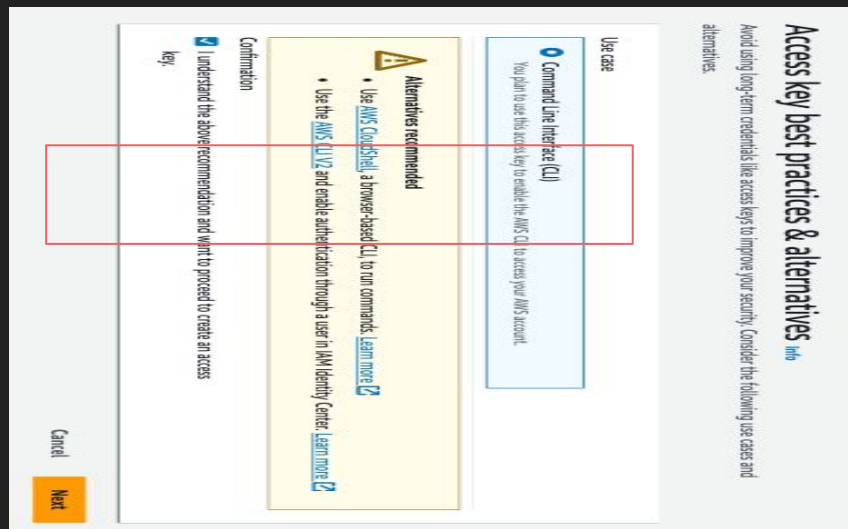
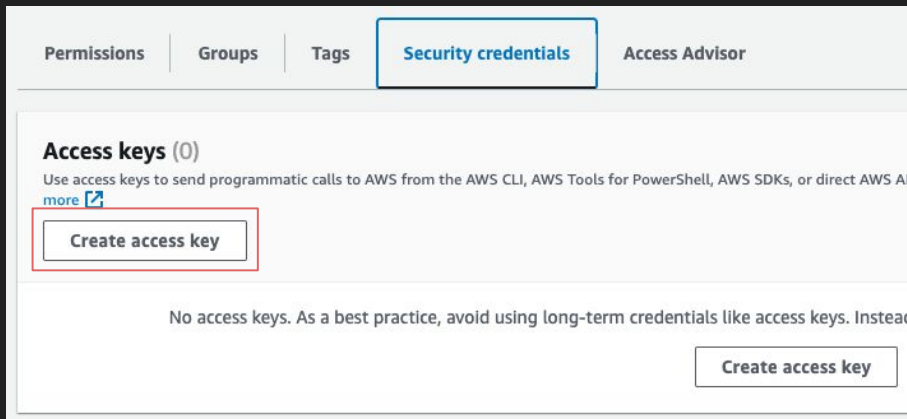
Users (3) [Info](#)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

<input type="checkbox"/>	User name ▲	Path ▼	Group: ▼	Last activity
<input type="checkbox"/>	Administrator	/	0	✓ 2 hours ago
<input type="checkbox"/>	operate_public_bucket_user	/	0	✓ Yesterday
<input type="checkbox"/>	terraform	/	0	-

建立一個 IAM 給 Terraform 使用 (Part 6, Final)

- 建立一個 access key 後，請記得把 key 記起來，等一下就會用到

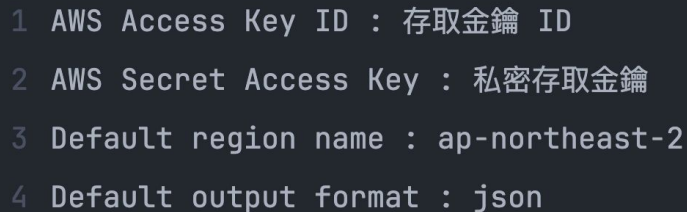


設定權限

- 安裝 AWS CLI 然後輸入 `aws configure` 在本地設定權限
- 依次輸入設定, 其中包含你剛剛建立的 IAM user access key



```
1 aws configure
```



```
1 AWS Access Key ID : 存取金鑰 ID  
2 AWS Secret Access Key : 私密存取金鑰  
3 Default region name : ap-northeast-2  
4 Default output format : json
```

* ap-northeast-2 是 AWS 的東京地區, 代表你建立的資源會放在 AWS 位在東京的資料中心

開始使用 Terraform

- 我們嘗試使用 Terraform 建立與剛剛一模一樣的 IAM user
- 建立一個新的資料夾 `aws-iam`, 然後在底下建立一個新的檔案 `main.tf`
- 在 `main.tf` 中, 寫上我們想建立的雲端資源

```
1 # main.tf
2
3 # set the provider
4 provider "aws" {
5 }
6
7 # create a iam user and create the access key
8 resource "aws_iam_user" "user" {
9   name = "another-terraform-user"
10 }
11
12 resource "aws_iam_access_key" "user" {
13   user = aws_iam_user.user.name
14 }
15
16 resource "aws_iam_user_policy_attachment" "attach" {
17   user      = aws_iam_user.user.name
18   policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
19 }
```

安裝 Provider

- 在 aws-iam 資料夾底下, 使用 `terraform init` 安裝 provider

```
terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.14.0...
- Installed hashicorp/aws v5.14.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

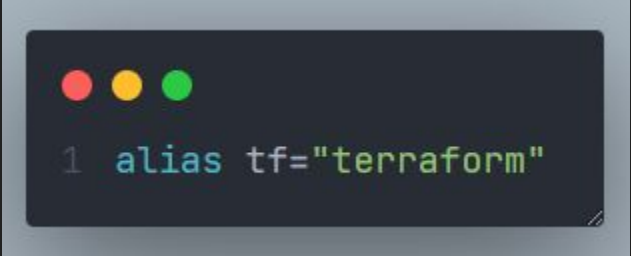
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

先等等

- 指令要打 `terraform` 真的太麻煩了, 讓我們用一下 `alias`

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `1 alias tf="terraform"` is displayed in a light green monospace font.

```
1 alias tf="terraform"
```

使用 Terraform 部署資源

- 在 aws-iam 資料夾底下, 使用 `terraform apply` 開始部署資源
- `apply` 會詳細列出 Terraform 要建立、修改、刪除哪些資源。務必確認清楚！
- 確認無誤後輸入 `yes`, Terraform 就會開始部署資源

tf apply

Terraform will perform the following actions:

```
# aws_iam_access_key.user will be created
+ resource "aws_iam_access_key" "user" {
  + create_date           = (known after apply)
  + encrypted_secret      = (known after apply)
  + encrypted_ses_smtp_password_v4 = (known after apply)
  + id                    = (known after apply)
  + key_fingerprint       = (known after apply)
  + secret                = (sensitive value)
  + ses_smtp_password_v4 = (sensitive value)
  + status                = "Active"
  + user                  = "another-terraform-user"
}

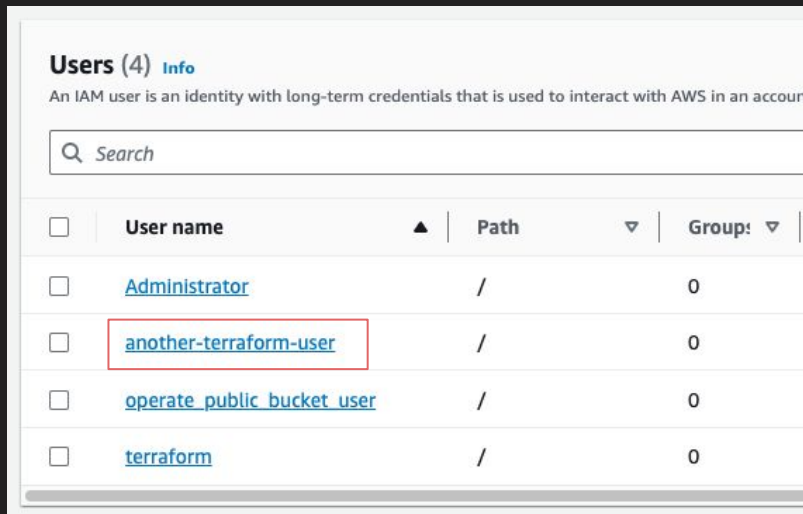
# aws_iam_user.user will be created
+ resource "aws_iam_user" "user" {
  + arn              = (known after apply)
  + force_destroy    = false
  + id              = (known after apply)
  + name            = "another-terraform-user"
  + path            = "/"
  + tags_all        = (known after apply)
  + unique_id       = (known after apply)
}

# aws_iam_user_policy_attachment.attach will be created
+ resource "aws_iam_user_policy_attachment" "attach" {
  + id              = (known after apply)
  + policy_arn      = "arn:aws:iam::aws:policy/AdministratorAccess"
  + user            = "another-terraform-user"
}
```

Plan: 3 to add, 0 to change, 0 to destroy.

查看資源是否建立成功

- 當 Terraform 執行完畢，可以上 AWS 的 IAM user 頁面上看一下是否建立成功
- 一行指令搞定我們剛剛 6 張投影片的內容，是不是很讚呢？



<input type="checkbox"/>	User name ▲	Path ▼	Groups ▼
<input type="checkbox"/>	Administrator	/	0
<input type="checkbox"/>	another-terraform-user	/	0
<input type="checkbox"/>	operate_public_bucket_user	/	0
<input type="checkbox"/>	terraform	/	0

Terraform State

- 建立完之後的, 你會資料夾底下多了一個 `terraform.tfstate` 檔案, 裡面記錄著你建立資源的詳細資料
- 其中也會包含敏感的資料, 例如剛剛你建立的 IAM user access key
- 這個檔案是不能進入版本控制的

想要多人協作？試試看 Remote State 吧

- 你可以將你的 `terraform.tfstate` 檔案放到雲端上讓所有人使用，並開啟協作
- Remote state 還會搭配 lock 的概念，確保同一時間只能有一位在修改資源

```
1 # in multi-person collaboration,
2 # terraform best practice recommends using state lock and remote state
3 terraform {
4   # ref: https://www.terraform.io/language/settings/backends/s3
5   backend "s3" {
6     bucket     = "terraform-state-files"
7     key        = "project-terraform.tfstate"
8     region     = "us-west-2"
9     dynamodb_table = "terraform-state-lock"
10  }
11 }
```

* 在 best practice 底下，都會建議使用 remote state，有興趣的朋友可以深入研究

刪除剛剛建立的資源

- 一樣一行指令搞定，確認後輸入 **yes** 就會將剛剛建立的資源全部刪除
- 請小心使用，確認前務必詳閱會刪除哪些資源



```
1 # destroy the resources !!! be careful !!!  
2 terraform destroy
```

Terraform 其實有些危險性，如果沒注意的話

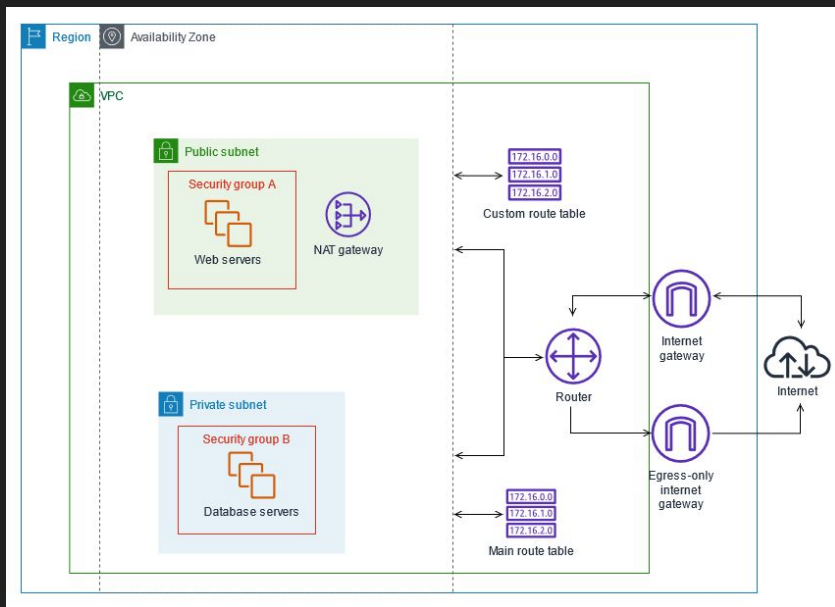
- 請仔細看 `apply` 與 `destroy` 的訊息，否則你重要的服務可能會 ...



* 如果對資源做出破壞性修改，`apply` 是會刪除現有資源，並重新建立一個新的，因此也要小心使用

接下來就是 Demo 的時間

- 讓我們使用 Terraform 建立一個 LEMP 的架構吧



八卦

Terraform 不再開源

- Terraform 的開發商 Hashicorp 宣布將旗下產品都轉為 BSL (Business Source License) 授權, 依然可以免費使用, 但不能把 Terraform 拿來商用, 推出與之相似的競品
- Hashicorp 此舉是為了打擊單純把 Terraform 拿來商用, 卻不做任何貢獻的廠商 (貢獻是指協助開發與維護 Terraform)
- 這引起很大的反彈聲浪, 因為 Terraform 發展到現在, 已經有了一個相當大的社群。不少公司也會擔心自己是否違反授權

新聞: [不堪雲端供應商濫用, HashiCorp 未來產品將改採 BSL 授權](#)

OpenTF

- 社群 fork 了一個版本, 建立了 OpenTF 專案, 並呼籲 Hashicorp 回心轉意改回開源授權, 但 Hashicorp 沒有正面回應



新聞: [社群分叉 Terraform 創建 OpenTF 專案, 準備申請進入 Linux 基金會](#)

影片: [The ruthless forking of Terraform](#)

等等，廠商濫用是？

- 有許多雲端廠商會將開源的專案，以自己的技術修改，並重新包裝成自己的產品
- 其中修改的部分，廠商會當作是自己的技術專利，並不會開源。
- 前幾年就有個雲端跟開源廠商鬧翻 ...



ElasticSearch VS OpenSearch

- ElasticSearch 非常不高興 AWS 用了他們家的專案卻不做貢獻，最後修改授權限制廠商商用
- 之後 AWS 在 ElasticSearch 的基礎上，開發了自己的搜尋引擎 OpenSearch

OpenSearch is not Elasticsearch

It is a common misconception that both Elasticsearch and OpenSearch are similar search engines. Since OpenSearch's general availability in 2021, Elasticsearch has outpaced OpenSearch by 4x in key innovations and features resulting in significantly [faster performance](#). More importantly, Elasticsearch accomplished this with significantly improved scalability while using far fewer resources. Elasticsearch and OpenSearch are clearly not the same.

Choose the [Elasticsearch service](#), the official offering from Elastic. It delivers Elasticsearch, Kibana, and all Elastic's turn key solutions like Search, Observability and Security, within AWS. Rest easy knowing that Elastic's search expertise is backing your deployments with security updates, solutions, and support.

ElasticSearch 網站還有某一頁特地拿 OpenSearch 比較，火藥味十足

新聞：[AWS 分叉 Elasticsearch 重新命名為 OpenSearch](#)

我想 Elasticsearch 是這麼看 Hashicorp 的



所以 Hashicorp 修改授權是有理的？

- 這也不一定，根據[新聞](#)，OpenTF 有一段聲明是這麼說的

同時，OpenTF宣言也認為Armon Dadgar在部落格文章中，對供應商濫用的HashiCorp開源專案的描述有誤，因為許多受BSL變更影響的供應商，同樣也對Terraform社群做出貢獻，文章舉例像是Terraform核心貢獻者有1,700位，Terraform AWS Provider則有2,800位貢獻者，Azure Provider也有1,300位貢獻者，而這些貢獻者絕大多數非HashiCorp員工。

懶人包：明明一堆廠商都有做貢獻，Hashicorp 卻獨攬全部的成果

也許我們該思考的

- 開源專案的背後，都是一群不求回報，默默付出的人
 - 萬人響應，一人開發
- 在我們享受開源的成果時，是否也該思考如何對開源作出貢獻
 - 有力出力，沒力出錢 (抖內)
- 沒錢養不起工程師

Thank You!