IaC - Terraform

Concepts:

1. **Infrastructure as Code (IaC) là gì?**
    - Viết code để mô tả và triển khai hạ tầng IT (máy chủ, mạng, DB, gateway...).
    - Giúp quản lý, backup, khôi phục hạ tầng dễ dàng, nhất là trên Cloud.

2. **Terraform là gì?**
    - Công cụ IaC phổ biến, open-source của HashiCorp.
    - Viết file cấu hình (HCL) để tạo, thay đổi, xóa hạ tầng tự động.
    - Lưu trạng thái hạ tầng bằng file `state`.

3. **Lợi ích khi dùng Terraform**
    - Dễ sử dụng, miễn phí.
    - Khai báo mong muốn, Terraform tự thực hiện.
    - Quản lý hạ tầng đa cloud (AWS, Azure, GCP...).
    - Tái tạo hạ tầng nhanh khi có sự cố.

Providers

1. **Provider là gì?**
    - Plugin giúp Terraform kết nối và quản lý hạ tầng (cloud, SaaS, API...).
    - Mỗi provider cung cấp các loại resource và data source mà Terraform có thể quản lý.

2. **Cách sử dụng Provider**

- Khai báo provider trong file cấu hình để Terraform biết cần cài đặt và sử dụng.
- Có thể cấu hình chi tiết (endpoint, region, credentials...).
- Nên giới hạn version provider để tránh lỗi khi có bản mới không tương thích.

3. **Provider đến từ đâu?**
    - Được phát hành riêng, có version riêng.
    - Được lưu trữ trên Terraform Registry, gồm:
        - Official (HashiCorp)
        - Partner (bên thứ ba, công ty)
        - Community (cộng đồng)
        - Archived (không còn duy trì)

Azure Provider

1. **Azure Provider là gì?**
    - Plugin giúp Terraform quản lý hạ tầng Azure qua Azure Resource Manager API.
    - Cung cấp nhiều loại resource và data source cho Azure.

2. **Cách xác thực với Azure**
    - **Azure CLI:** Dùng khi thao tác local.
    - **Managed Service Identity:** Dùng cho VM, App Service, AKS.
    - **Service Principal:** Dùng cho CI/CD, automation (qua client secret hoặc certificate).
    - **OpenID Connect:** Dùng cho pipeline, workload identity.
    - **Khuyến nghị:** Dùng Service Principal hoặc Managed Identity cho môi trường tự động.

3. **Khai báo provider**

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.0.0"
    }
  }
}

provider "azurerm" {
  features {}
  # resource_provider_registrations = "none" # Nếu không đủ quyền đăng ký resource provider
}
```

4. **Các tham số cấu hình quan trọng**

- `features` (bắt buộc): Tùy chỉnh hành vi provider.
- `subscription_id`, `client_id`, `tenant_id`, `client_secret`: Dùng cho Service Principal.
- `environment`: Chọn môi trường Azure (public, china, usgovernment...).
- `resource_provider_registrations`: Tùy chỉnh việc đăng ký resource provider (core, extended, all, none).
- Có thể dùng biến môi trường để truyền giá trị.

5. **Quản lý Resource Provider**

- Provider sẽ tự đăng ký các resource provider cần thiết trước khi chạy plan/apply.
- Nếu không đủ quyền, nên set `resource_provider_registrations = "none"` để tránh lỗi.

6. **Ví dụ tạo resource**

```
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "West Europe"
}

resource "azurerm_virtual_network" "example" {
  name                = "example-network"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  address_space       = ["10.0.0.0/16"]
}
```

Terraform

1. **Cấu trúc cơ bản của Terraform**
   - **Block:** Đơn vị cấu hình (resource, provider, variable, output...).
   - **Argument:** Thuộc tính trong block, gán giá trị cho resource.
   - **Identifier:** Tên định danh cho block/resource.
   - **Comment:** Dùng `#` hoặc để ghi chú.

2. **Provider & Backend**
   - Khai báo provider (AzureRM, AzureAD, Random...) trong block `terraform`.
   - Khai báo backend để lưu trữ state (nên dùng Azure Storage cho team).

3. **Resource**
   - Định nghĩa resource để tạo hạ tầng (ví dụ: resource group, vnet...).
   - Sử dụng biến để tái sử dụng cho nhiều môi trường.

4. **Input Variables**
   - Khai báo biến trong file `variables.tf`.

- Truyền giá trị biến qua:
  - `var` khi chạy lệnh
  - `var-file` với file `.tfvars`
  - Tên file `terraform.tfvars` hoặc `.auto.tfvars` sẽ tự động load
  - Biến môi trường: `export TF_VAR_<variable_name>=value`

5. **Output Values**
   - Khai báo output để hiển thị thông tin sau khi apply (location, id, name...).

6. **Quy trình làm việc**
   a. Khởi tạo: `terraform init`
   b. Kiểm tra cú pháp: `terraform validate`
   c. Xem trước thay đổi: `terraform plan`
   d. Thực thi: `terraform apply`
   e. Xem trạng thái: `terraform show`
   f. Định dạng file: `terraform fmt`

7. **Quản lý state**
   - State file (`terraform.tfstate`) lưu trạng thái hạ tầng.
   - Nên lưu state ở Azure Storage để tránh mất dữ liệu, hỗ trợ làm việc nhóm.
   - Di chuyển state từ local lên Azure Storage bằng backend `azurerm`.

8. **Terraform nó không có cơ chế tự rollback khi 1 flow chạy fail thì mình có 2 phương án để lựa chọn: vd flow Create cluster → Create Namespace(fail) → Deploy service:**
   a. Cluster đã được tạo sẵn mình chỉ cần fix lỗi ở create namespace sau đó chạy lại `terraform apply` thì terraform nó sẽ không tạo cluster nữa mà nó sẽ

tiếp tục tạo namespace rồi deploy service.

b. Nếu mình muốn dừng và xoá hết mọi thứ thì chạy `terraform destroy` thì nó sẽ xoá mọi thứ ở file state và hạ tầng luôn.

Nếu mình muốn chỉ destroy mọi thứ bị lỗi trong flow đó thôi thì mình phải tách riêng thư mục ra.Ví dụ `infra/flow1/main.tf` (Namespace payments + Payment API) vs `infra/flow2/main.tf` (Namespace orders + Order API).

Thì khi flow nào lỗi mình chạy `terraform destroy` trong thư mục đó thì nó chỉ xoá hạ tầng trong thư mục đó thôi .

Còn cả 2 flow đều tạo cluster thì mình sẽ để vào thư mục `infra/shared/main.tf` Dùng **remote state** để kết nối nó với flows. Và mình có thể tự động bằng cách setup CI/CD:

CI/CD orchestration

- Chạy `shared/` pipeline trước (tạo cluster).
- Sau đó mới chạy `flow1/`, `flow2/`.
- Nếu `flow1` fail → rollback(destroy) namespace/service A, cluster vẫn còn cho `flow2`.

Mình cũng có thể setup thêm nếu cả 2 flow đều lỗi thì xoá luôn cả cluster:

- Chạy `flow1` → nếu fail, đánh dấu flag `flow1_failed=true`.
- Chạy `flow2` → nếu fail, đánh dấu flag `flow2_failed=true`.
- Sau cả 2, nếu `flow1_failed && flow2_failed` → trigger job (xoá cluster)