



(Theory) Container and Infrastructure as Code

- [1. Containerization Concept](#)
- [2. Docker: Platform Containerization](#)
- [3. Kubernetes: Orkestrasi Container](#)
- [4. Docker Workflow](#)
- [5. Infrastructure as Code \(Terraform\)](#)

▼ 1. Containerization Concept

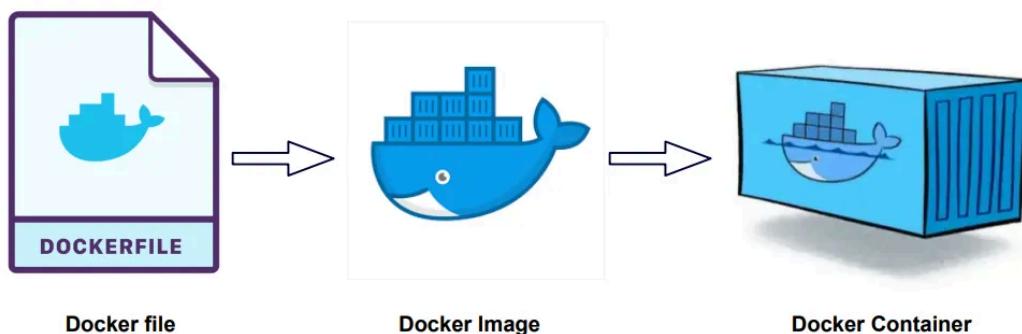
▼ Apa itu Containerization?

- **Containerization** adalah teknik virtualisasi di level sistem operasi yang mengemas aplikasi beserta seluruh dependensinya (seperti library, runtime, konfigurasi, dan file pendukung) ke dalam satu paket terisolasi yang disebut **container**. Dengan demikian, aplikasi yang telah di-container dapat dijalankan secara konsisten di berbagai lingkungan—baik itu di mesin pengembangan, server produksi, ataupun cloud—tanpa khawatir terjadi perbedaan konfigurasi.
- **Analogi:** Bayangkan **container** seperti **kardus yang berisi barang lengkap untuk pindah rumah**. Kardus ini terisolasi (tidak tercampur dengan barang lain), mudah dipindahkan, dan bisa langsung dibuka di rumah baru tanpa perlu menyusun ulang isinya.

▼ Keunggulan Containerization

- **Portabilitas:** Karena semua komponen aplikasi dikemas bersama, container dapat dipindahkan antar lingkungan dengan mudah dan dijalankan di lingkungan mana pun (laptop, server, cloud).
- **Efisiensi Resource:** Container berbagi kernel host, sehingga penggunaan memori dan CPU lebih efisien dibandingkan virtual machine. Tidak membutuhkan OS lengkap seperti VM (Virtual Machine), sehingga hemat sumber daya.
- **Konsistensi & Isolasi:** Setiap container berjalan secara terisolasi, mengurangi risiko konflik antar aplikasi yang berjalan pada sistem yang sama. Masalah di satu container tidak memengaruhi container lain.

▼ 2. Docker: Platform Containerization



▼ Apa itu Docker?

- <https://youtu.be/Fe0-jOsELJE?si=4DhAK8IEVhpq5sbu>
- Docker adalah platform open-source yang mempermudah pembuatan, pengemasan, dan pengelolaan container. Docker menggunakan file bernama **Dockerfile** untuk mendefinisikan lingkungan aplikasi, lalu membangunnya menjadi **image** (template container) yang bisa dijalankan di mana saja.
- **Analogi:** Docker seperti **pabrik kardus otomatis**.
 -
 - Dockerfile** = Resep untuk membuat kardus.
 -

Image = Kardus yang sudah jadi dan siap dikirim.

-

Container = Kardus yang sedang dibuka dan digunakan.

- **Contoh Studi Kasus:**

Sebuah tim developer mengembangkan aplikasi web dengan database PostgreSQL. Mereka membuat dua Dockerfile:

1. Untuk aplikasi web (berbasis Node.js).
2. Untuk database PostgreSQL.

Dengan Docker, mereka bisa menjalankan aplikasi dan database di container terpisah di laptop lokal, lalu mengirim image-nya ke server produksi tanpa khawatir perbedaan versi library atau OS.

▼ Kegunaan Docker

- **Membangun dan Mengemas Aplikasi:** Menggunakan *Dockerfile*, developer mendefinisikan bagaimana aplikasi dan dependensinya dikemas ke dalam sebuah *image*. *Image* inilah yang nantinya dijalankan sebagai container.
- **Distribusi:** Docker Hub menyediakan repositori untuk berbagi dan mendistribusikan image.
- **Kemudahan Penggunaan:** Dengan perintah sederhana, seperti `docker run`, aplikasi dalam container dapat dijalankan di mana saja selama Docker telah diinstal.

▼ Docker Compose

<https://docs.docker.com/compose/>

▼ What is Docker Compose?

Docker Compose is a tool for **defining and running multi-container applications** using a declarative YAML configuration file (`docker-compose.yml`). It automates container orchestration, enabling you to manage interconnected services (e.g., databases, APIs, frontends), networks, and volumes in one cohesive setup.

▼ Why Docker Compose is Important?

- **Simplifies Multi-Container Workflows:** Replace manual `docker run` commands and complex scripts with a single file.

- **Single Source of Truth:** All service configurations (ports, environment variables, dependencies) are centralized.
- **Environment Consistency:** Ensure identical setups across development, testing, and production.
- **Efficient Networking & Resource Sharing:** Automatically creates a **default network** for services to communicate, eliminating manual network setup.

▼ Setup Docker Compose

1. Install Docker (Compose is included in Docker Desktop for Windows/Mac).
2. For Linux:

```
sudo curl -L "<https://github.com/docker/compose/releases/l
atest/download/docker-compose-$>(uname -s)-$(uname -
m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

3. Verify:

```
docker-compose --version
```

▼ Struktur File Docker Compose

File `docker-compose.yml` terdiri dari:

1. `version`: Versi syntax Compose (e.g., `'3.8'`).
2. `services`: Daftar container/service yang akan dijalankan.
 - Setiap service mendefinisikan `image`, `ports`, `environment`, `volumes`, dan `depends_on` (urutan startup).
3. `networks` & `volumes` (opsional): Konfigurasi resource yang digunakan bersama.

Contoh:

```
services:
  postgres:
    image: postgres:14
```

```

environment:
  POSTGRES_PASSWORD: mypassword
volumes:
  - pg_data:/var/lib/postgresql/data

pgadmin:
  image: dpage/pgadmin4
  ports:
    - "8080:80"
  depends_on:
    - postgres

volumes:
  pg_data:

```

Keunggulan Jaringan Otomatis:

- Tanpa Compose, Anda harus membuat network secara manual dan menghubungkan container dengan `-network`.
- **Docker Compose membuat network default secara otomatis,** sehingga semua service dalam file yang sama dapat saling terhubung via nama service (e.g., `postgres://postgres:5432` dari container `pgadmin`).

Takeaway

Docker Compose mengotomatisasi orchestration container, termasuk jaringan, volume, dan dependency management. Dengan satu perintah (`docker-compose up`), seluruh aplikasi multi-container siap berjalan tanpa konfigurasi manual.

▼ 3. Kubernetes: Orkestrasi Container

▼ Apa itu?

Kubernetes (K8s) adalah sistem **orchestration** untuk mengelola container dalam skala besar. Kubernetes mengotomatiskan deployment, scaling, dan manajemen aplikasi yang terdiri dari banyak container.

▼ Fitur utamanya meliputi:

- **Manajemen Cluster:** Mengatur banyak node (mesin fisik atau virtual) yang menjalankan container.
- **Pod dan Service:** Container dikelompokkan dalam unit yang disebut *pod*. Kubernetes mengelola komunikasi antar pod dan memastikan layanan tetap tersedia.
- **Auto-Scaling & Load Balancing:** Secara otomatis menskalakan jumlah container berdasarkan beban kerja, serta mendistribusikan lalu lintas untuk menghindari kelebihan beban.
- **Self-Healing:** Jika ada container yang gagal, Kubernetes akan otomatis mencoba memulihkan dengan menjalankan container baru.

▼ Analogi:

Kubernetes seperti **pengawas bandara**.

- **Container** = Pesawat.
- **Node** = Landasan pacu.
- **Cluster** = Seluruh bandara.

Pengawas bandara (Kubernetes) memastikan:

- Pesawat (container) mendarat di landasan (node) yang tepat.
- Jika ada pesawat rusak, langsung diganti dengan cadangan.
- Menambah landasan jika lalu lintas padat (auto-scaling).

Kubernetes menjadi solusi andalan bagi perusahaan yang harus mengelola ratusan atau bahkan ribuan container sekaligus, sehingga memudahkan deployment dan pemeliharaan aplikasi secara terdistribusi.

Perbandingan Docker vs Kubernetes		
Aspek	Docker	Kubernetes
Fokus	Membuat dan menjalankan container	Mengelola banyak container di cluster
Skala	Cocok untuk lingkungan kecil	Dirancang untuk skala besar
Komponen	Dockerfile, Image, Container	Pod, Service, Deployment, Cluster

▼ 4. Docker Workflow

Docker adalah alat yang membantu dalam containerization, yaitu menjalankan aplikasi beserta semua dependensinya dalam lingkungan yang terisolasi. Berikut adalah penjelasan dari masing-masing konsep yang kamu tanyakan, disertai contoh dalam konteks **Data Engineering**.

1. Dockerfile

Dockerfile adalah file skrip yang berisi instruksi untuk membangun **Docker Image**. Dockerfile menentukan lingkungan yang dibutuhkan, paket yang harus diinstal, dan perintah yang akan dijalankan saat container berjalan.

Contoh: Membuat Dockerfile untuk pipeline data ETL

Misalnya, kita ingin membuat **ETL pipeline** sederhana menggunakan Python dan Pandas.

Dockerfile untuk ETL pipeline menggunakan Python:

```
# Menggunakan image dasar Python versi 3.9
FROM python:3.9

# Menetapkan working directory di dalam container
WORKDIR /app

# Menyalin semua file dari host ke dalam container
COPY ..

# Menginstal dependensi yang diperlukan
RUN pip install --no-cache-dir -r requirements.txt

# Menjalankan script ETL saat container dijalankan
CMD ["python", "etl_pipeline.py"]
```

2. Docker Image

Docker Image adalah blueprint atau template untuk membuat container. Setelah kita menulis Dockerfile, kita perlu membangun Docker Image

menggunakan perintah:

```
docker build -t etl_pipeline .
```

Image ini dapat digunakan untuk membuat container berkali-kali tanpa perlu menginstal ulang dependensi.

Analogi:

Docker Image seperti template sistem operasi + aplikasi yang siap dijalankan.

3. Container

Container adalah instansi dari Docker Image yang sedang berjalan. Setelah image dibuat, kita bisa menjalankannya sebagai container:

```
docker run -d --name etl_container etl_pipeline
```

- `-d` : Menjalankan container di background
- `--name etl_container` : Memberi nama container

Cek container yang berjalan:

```
docker ps
```

Contoh Workflow:

Misalkan kita punya **data transaksi** dalam format CSV dan ingin membersihkannya menggunakan **script Python dalam container**.

1. **Dockerfile** telah dibuat untuk menginstal Pandas dan dependensi lainnya.
2. **Docker Image** dibangun dari Dockerfile.
3. **Container** dijalankan dengan perintah `docker run`.
4. **Script ETL dijalankan dalam container** untuk membaca CSV, membersihkannya, dan menyimpan hasil ke database.

4. Docker Compose

Docker Compose adalah alat untuk mengelola beberapa container dalam satu konfigurasi menggunakan file `docker-compose.yml`.

Contoh Kasus: ETL dengan PostgreSQL

Misalkan kita ingin menjalankan pipeline ETL yang membutuhkan:

- **Python script (ETL process)**
- **PostgreSQL sebagai database penyimpanan hasil ETL**

Kita bisa membuat `docker-compose.yml`:

```
version: '3'
services:
  db:
    image: postgres:latest
    container_name: postgres_db
    restart: always
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: etl_db
    ports:
      - "5432:5432"

  etl:
    build: .
    container_name: etl_pipeline
    depends_on:
      - db
    environment:
      DB_HOST: postgres_db
      DB_USER: user
      DB_PASSWORD: password
      DB_NAME: etl_db
```

Cara menjalankan semua container:

```
docker-compose up -d
```

Dengan ini:

- Container **PostgreSQL** akan dijalankan.
 - Container **ETL Python** akan berjalan setelah PostgreSQL siap.
 - Data hasil ETL akan masuk ke PostgreSQL.
-

5. Docker Daemon

Docker Daemon adalah proses latar belakang yang menjalankan Docker di sistem kamu. Ia bertanggung jawab untuk:

- Membangun dan menjalankan container
- Mengelola image dan volume
- Mengatur jaringan antar container

Docker Daemon otomatis berjalan saat Docker aktif, dan kamu bisa memeriksanya dengan:

```
systemctl status docker # untuk Linux
```

Workflow dalam Data Engineering dengan Docker

Berikut adalah alur kerja Data Engineering menggunakan Docker:

1. Tulis kode ETL dalam Python

Misalnya, script untuk membaca CSV, membersihkannya, lalu menyimpan ke database.

2. Tulis Dockerfile

- Pilih base image (misalnya `python:3.9`)
- Instal dependensi (`pandas`, `sqlalchemy`, dll.)
- Copy file ETL ke container
- Tetapkan perintah untuk menjalankan ETL

3. Buat Docker Image

```
docker build -t etl_pipeline .
```

4. Jalankan Container Secara Manual (Opsional)

```
docker run -d --name etl_container etl_pipeline
```

5. Gunakan Docker Compose jika ada banyak service

- Database seperti PostgreSQL atau MySQL
- Kafka atau Airflow untuk orkestrasi

6. Jalankan Semua Service Sekaligus

```
docker-compose up -d
```

7. Cek Log atau Debug

```
docker logs etl_container
```

8. Stop dan Hapus Container Jika Tidak Diperlukan Lagi

```
docker-compose down
```

Kesimpulan

- **Dockerfile** → Skrip untuk membangun **Docker Image**
- **Docker Image** → Template sistem yang siap dijalankan
- **Container** → Versi berjalan dari image
- **Docker Compose** → Mengatur banyak container dalam satu konfigurasi
- **Docker Daemon** → Proses yang menjalankan Docker di sistem

Dengan Docker, kamu bisa menjalankan pipeline data **secara konsisten**, baik di lokal maupun di server tanpa perlu khawatir tentang perbedaan lingkungan. 🚀

▼ Studi Kasus: Membangun ETL Pipeline Sederhana dengan Docker dan WSL

Scenario:

Anda adalah seorang Data Engineer yang bertugas membangun ETL (Extract, Transform, Load) pipeline sederhana. Pipeline ini akan:

1. Mengekstrak data dari file CSV.
2. Melakukan transformasi data (misal: membersihkan data).
3. Memuat data ke database PostgreSQL.

Anda akan menggunakan Docker untuk mengisolasi lingkungan kerja dan WSL untuk menjalankan perintah Linux secara native di Windows.

Langkah-langkah dan Penjelasan

1. Persiapan Lingkungan

- **Instal Docker Desktop:**
 - Pastikan Docker Desktop sudah terinstal dan menggunakan WSL 2 sebagai backend.
 - Buka **Docker Desktop** → **Settings** → **General** → Pastikan **Use the WSL 2 based engine** dicentang.
 - Buka **Settings** → **Resources** → **WSL Integration** → Aktifkan integrasi dengan distribusi WSL (misal: Ubuntu).
- **Instal WSL dan Distribusi Linux:**
 - Jika belum, instal WSL 2 dan distribusi Linux (misal: Ubuntu) dengan perintah:

```
wsl --install
```

2. Membuat Dockerfile untuk Image ETL

- Buat direktori proyek di WSL (misal: `~/etl-pipeline`).
- Buat file `Dockerfile` untuk mendefinisikan image Docker:

```
# Gunakan image base Python
FROM python:3.9-slim

# Set working directory
WORKDIR /app
```

```
# Copy requirements file  
COPY requirements.txt .  
  
# Install dependencies  
RUN pip install --no-cache-dir -r requirements.txt  
  
# Copy script ETL  
COPY etl_script.py .  
  
# Run the ETL script  
CMD ["python", "etl_script.py"]
```

- Buat file `requirements.txt` untuk dependensi Python:

```
pandas  
sqlalchemy  
psycopg2-binary
```

- Buat file `etl_script.py` untuk logika ETL:

```
import pandas as pd  
from sqlalchemy import create_engine  
  
# Extract: Load data from CSV  
df = pd.read_csv('data.csv')  
  
# Transform: Clean data  
df['column_name'] = df['column_name'].str.strip() # Contoh transformasi sederhana  
  
# Load: Save data to PostgreSQL  
engine = create_engine('postgresql://user:password@db:5432/mydatabase')  
df.to_sql('my_table', engine, if_exists='replace', index=False)  
  
print("ETL process completed!")
```

3. Build Image Docker

- Di terminal WSL (Ubuntu), navigasikan ke direktori proyek:

```
cd ~/etl-pipeline
```

- Build image Docker:

```
docker build -t etl-pipeline .
```

4. Menjalankan Container dengan Docker Compose

- Buat file `docker-compose.yml` untuk mengatur container ETL dan PostgreSQL:

```
version: '3'
services:
  etl:
    image: etl-pipeline
    volumes:
      - ./app
    depends_on:
      - db

  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    ports:
      - "5432:5432"
```

- Jalankan container dengan Docker Compose:

```
docker-compose up
```

5. Peran WSL dalam Proses Ini

- **Terminal Linux:**
 - Anda menggunakan terminal WSL (Ubuntu) untuk menjalankan perintah Docker (`docker build` , `docker-compose up` , dll.).
 - WSL memungkinkan Anda menjalankan perintah Linux secara native di Windows tanpa perlu beralih ke sistem operasi lain.
 - **Integrasi dengan Docker Desktop:**
 - Docker Desktop menggunakan WSL 2 sebagai backend untuk menjalankan container. Ini memungkinkan Anda mengelola container dari terminal WSL atau Docker Desktop GUI.
-

6. Eksekusi Kode

- **Di Mana Kode Dieksekusi?:**
 - Kode Python (`etl_script.py`) dieksekusi di dalam container Docker yang berjalan di WSL 2.
 - Anda bisa menjalankan perintah Docker dari terminal WSL (Ubuntu) atau menggunakan Docker Desktop GUI.
 - **Docker Desktop vs VSCode:**
 - **Docker Desktop** adalah aplikasi untuk mengelola container dan image Docker. Ini bukan IDE seperti VSCode, tetapi lebih seperti alat untuk menjalankan dan mengelola container.
 - **VSCode** adalah IDE yang bisa digunakan untuk menulis kode. Anda bisa mengintegrasikan VSCode dengan Docker dan WSL untuk pengembangan yang lebih nyaman.
-

7. Contoh Integrasi VSCode dengan WSL dan Docker

- Buka VSCode dan instal ekstensi **Remote - WSL** dan **Docker**.
- Buka proyek Anda di WSL melalui VSCode:
 - Di VSCode, tekan `Ctrl + Shift + P` → Pilih **Remote-WSL: New Window using Distro**.
 - Pilih distribusi WSL (misal: Ubuntu) dan buka direktori proyek (`~/etl-pipeline`).

- Gunakan terminal terintegrasi di VSCode untuk menjalankan perintah Docker.
-

Kesimpulan

Dalam studi kasus ini:

- **WSL** digunakan untuk menjalankan perintah Linux secara native di Windows.
- **Docker** digunakan untuk mengisolasi lingkungan kerja dan menjalankan ETL pipeline.
- **Docker Desktop** adalah alat untuk mengelola container, sedangkan **VSCode** adalah IDE untuk menulis dan mengelola kode.

Dengan kombinasi WSL, Docker, dan VSCode, Anda bisa membangun dan menjalankan pipeline Data Engineering dengan efisien di Windows. Jika ada pertanyaan lebih lanjut, silakan beri tahu saya! 😊

▼ 5. Infrastructure as Code (Terraform)

▼ Pengenalan Terraform

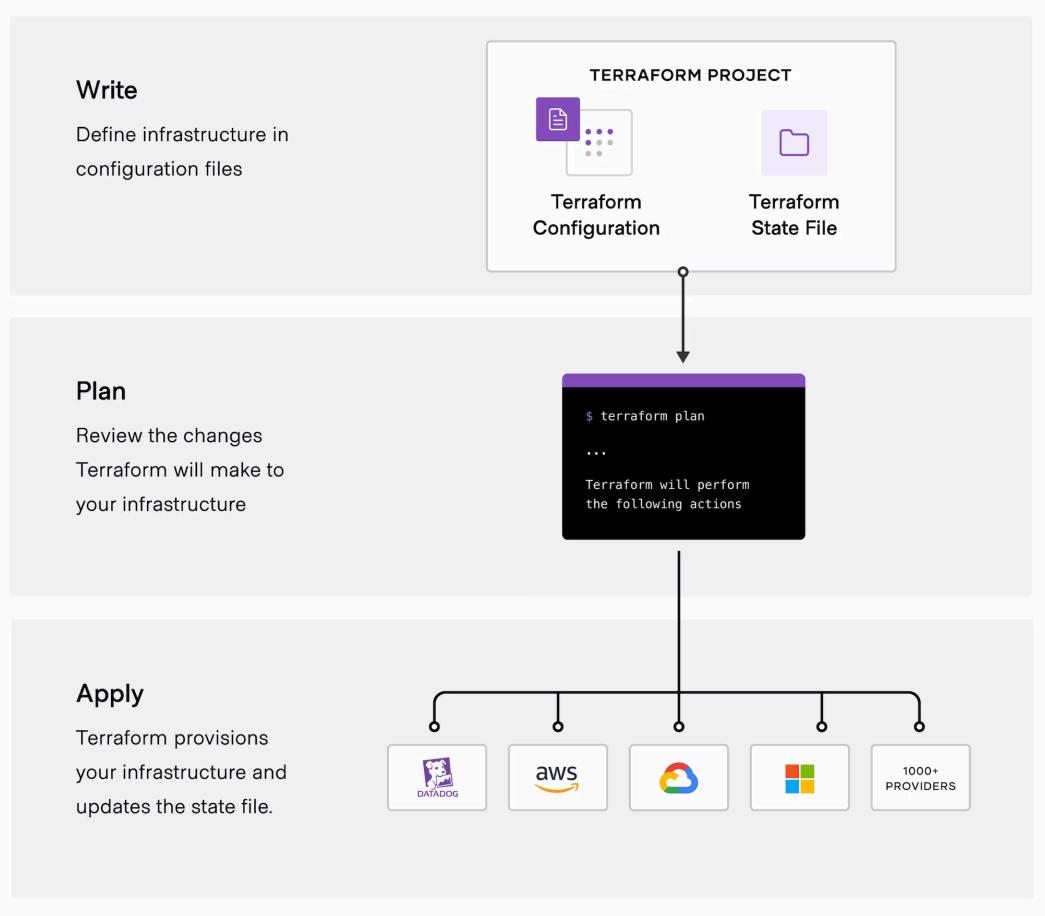
▼ Apa itu Terraform

You may find yourself using the graphical user interface of your favorite cloud provider clicking thousand different buttons to get your infrastructure configured properly but what happens when you need to reproduce the same thing again in the future. Terraform provides a way to represent your infrastructure and all that settings with the hashicorp configuration language. Think of it like a human readable blueprint that can execute and automate everything you do in the cloud.

Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It allows you to define, provision, and manage infrastructure resources in a declarative manner. These resources can include virtual machines, databases, networks, and much more across various cloud providers, on-premises data centers, or hybrid environments.

You describe the desired state of your infrastructure in .tf configuration files, and Terraform handles the provisioning and

management. Terraform supports a wide range of cloud providers (e.g., AWS, Google Cloud Platform, Azure) and on-premises solutions.



▼ Konsep Infrastructure as Code

Infrastructure as Code (IaC) adalah proses pengelolaan dan penyediaan infrastruktur komputasi melalui file konfigurasi kode, bukan melalui proses konfigurasi fisik perangkat keras atau alat konfigurasi interaktif. IaC memperlakukan konfigurasi infrastruktur sama seperti kode perangkat lunak, menggunakan teknik yang sama untuk mengembangkan dan mengujinya.

Konsep utama IaC meliputi:

- Deklaratif vs Imperatif** - Terraform menggunakan pendekatan deklaratif di mana Anda menentukan "apa" yang Anda inginkan, bukan "bagaimana" mencapainya.
- Idempotent** - Operasi dapat dijalankan berulang kali tanpa mengubah hasil di luar eksekusi pertama.

3. **Konsistensi** - Infrastruktur selalu sesuai dengan konfigurasi yang didefinisikan dalam kode.
4. **Pengelolaan Versi** - Perubahan infrastruktur dapat dilacak dan dikelola menggunakan sistem kontrol versi seperti Git.

▼ Apa Manfaat Terraform

1. **Pengelolaan Multi-Cloud** - Terraform dapat mengelola beberapa penyedia cloud dan layanan dalam satu konfigurasi.
2. **Otomatisasi** - Menghilangkan kesalahan manusia dan meningkatkan efisiensi melalui otomatisasi.
3. **Pengujian & Validasi** - Memungkinkan pengujian perubahan infrastruktur sebelum diterapkan.
4. **Modularitas & Reusabilitas** - Kode dapat dimodulkan dan digunakan kembali di berbagai proyek.
5. **Kolaborasi Tim** - Memudahkan kolaborasi dengan pengelolaan versi dan state bersama.
6. **Dokumentasi Imanen** - Konfigurasi bertindak sebagai dokumentasi "hidup" yang selalu akurat.
7. **Tracking Perubahan** - Perubahan infrastruktur dapat dilacak dan diaudit.
8. **Skalabilitas** - Memungkinkan pengelolaan infrastruktur berskala besar dengan efisien.

▼ Limitasi Terraform

1. **Kurva Pembelajaran** - Memerlukan waktu untuk mempelajari bahasa HCL dan konsep-konsep Terraform.
2. **Pengelolaan State** - Pengelolaan state dapat menjadi kompleks dalam lingkungan tim.
3. **Limited Debugging** - Kemampuan debugging yang terbatas dibandingkan dengan alat pengembangan tradisional.
4. **Fitur Provider yang Tidak Merata** - Tidak semua provider cloud memiliki dukungan fitur yang sama dalam Terraform.
5. **Performa** - Provisi infrastruktur besar dapat memakan waktu lama.

6. **Keterbatasan Konversi** - Mengkonversi infrastruktur yang ada ke dalam Terraform tidak selalu mudah (terraform import masih terbatas).

▼ Perintah Utama Terraform

1. **terraform init** - Menginisialisasi direktori kerja Terraform, mengunduh plugin provider yang diperlukan.
2. **terraform plan** - Menunjukkan perubahan yang akan dibuat pada infrastruktur tanpa menerapkannya.
3. **terraform apply** - Menerapkan perubahan yang didefinisikan dalam konfigurasi.
4. **terraform destroy** - Menghapus semua sumber daya yang dikelola oleh konfigurasi Terraform tertentu.
5. **terraform validate** - Memvalidasi file konfigurasi Terraform.
6. **terraform fmt** - Memformat file konfigurasi sesuai dengan konvensi standar.
7. **terraform state** - Melihat dan memanipulasi state Terraform.
8. **terraform import** - Mengimpor sumber daya yang ada ke dalam manajemen Terraform.
9. **terraform workspace** - Mengelola workspace Terraform (lingkungan yang berbeda).

▼ Terraform dalam Konteks Data Engineering

Dalam Data Engineering, Terraform sangat berguna untuk:

1. **Penyediaan Data Lakes** - Otomatisasi penyediaan bucket penyimpanan, IAM, dan konfigurasi keamanan.
2. **Pipeline Data** - Menyiapkan infrastruktur untuk ETL dan pipeline data.
3. **Data Warehousing** - Penyediaan dan pengelolaan warehouse seperti Snowflake, BigQuery, atau Redshift.
4. **Compute Resources** - Menyediakan cluster komputasi untuk pemrosesan data.
5. **Streaming Infrastructure** - Mengatur layanan streaming seperti Kafka, Kinesis, atau PubSub.

▼ Perbedaan utama Containerization (Docker) dengan Infrastructure as Code (Terraform)

Aspek	Docker	Terraform
Fokus Utama	Mengemas aplikasi dan dependensinya dalam wadah yang dapat dipindahkan	Menyediakan dan mengelola infrastruktur cloud dan lokal
Tingkat Abstraksi	Tingkat aplikasi	Tingkat infrastruktur
Pendekatan	Containerization	Infrastructure as Code
Bahasa Konfigurasi	Dockerfile, docker-compose.yml	HCL (HashiCorp Configuration Language)
State Management	Tidak menyimpan state secara eksplisit	Menyimpan dan mengelola state infrastruktur
Lingkup	Terutama pada runtime aplikasi	Seluruh infrastruktur cloud (VM, jaringan, load balancers, database, dll.)
Use Case dalam Data Engineering	Menjalankan alat data dan aplikasi (Spark, Airflow, dll.)	Menyediakan dan mengelola infrastruktur untuk data platform (VPC, S3, Dataproc, BigQuery, dll.)
Portabilitas	Sangat portabel antar lingkungan	Portabilitas bergantung pada provider cloud
Integrasi	Sering digunakan dengan orchestrator seperti Kubernetes	Sering digunakan dengan CI/CD dan alat DevOps lainnya

▼ Terraform untuk Google Cloud Platform

Analogi Keseluruhan: Membangun Rumah

Untuk memahami Terraform, bayangkan Anda sebagai kontraktor yang membangun rumah. Terraform adalah seperti sistem manajemen konstruksi otomatis yang memungkinkan Anda:

- Menggambar blueprint (kode)
- Memeriksa rencana sebelum pembangunan (plan)
- Membangun sesuai blueprint (apply)
- Membongkar jika diperlukan (destroy)

▼ 1. Mengunduh dan Menginstal Terraform

Langkah-langkah:

1. Kunjungi situs resmi Terraform:
<https://www.terraform.io/downloads.html>
2. Unduh versi Windows (ZIP file)
3. Ekstrak file `terraform.exe` ke folder tertentu (misalnya: `C:\terraform`)
4. Tambahkan lokasi folder ke PATH Windows:
 - Buka "Edit the system environment variables"
 - Klik "Environment Variables"
 - Edit variabel "Path"
 - Tambahkan `C:\terraform`
5. Buka Command Prompt baru dan ketik `terraform -v` untuk memverifikasi instalasi

Analogi:

Ini seperti membeli dan menyiapkan peralatan konstruksi Anda. Menambahkan Terraform ke PATH seperti menempatkan alat-alat di tempat yang mudah dijangkau dalam kotak perkakas sehingga Anda bisa mengaksesnya kapan saja.

▼ 2. Memahami Terraform Provider

Penjelasan:

```
terraform {  
  required_providers {  
    google = {  
      source = "hashicorp/google"  
      version = "6.17.0"  
    }  
  }  
}
```

Provider adalah plugin yang memungkinkan Terraform berinteraksi dengan platform cloud tertentu (seperti Google Cloud, AWS, Azure). Blok ini menentukan bahwa kita memerlukan Google Provider versi 6.17.0.

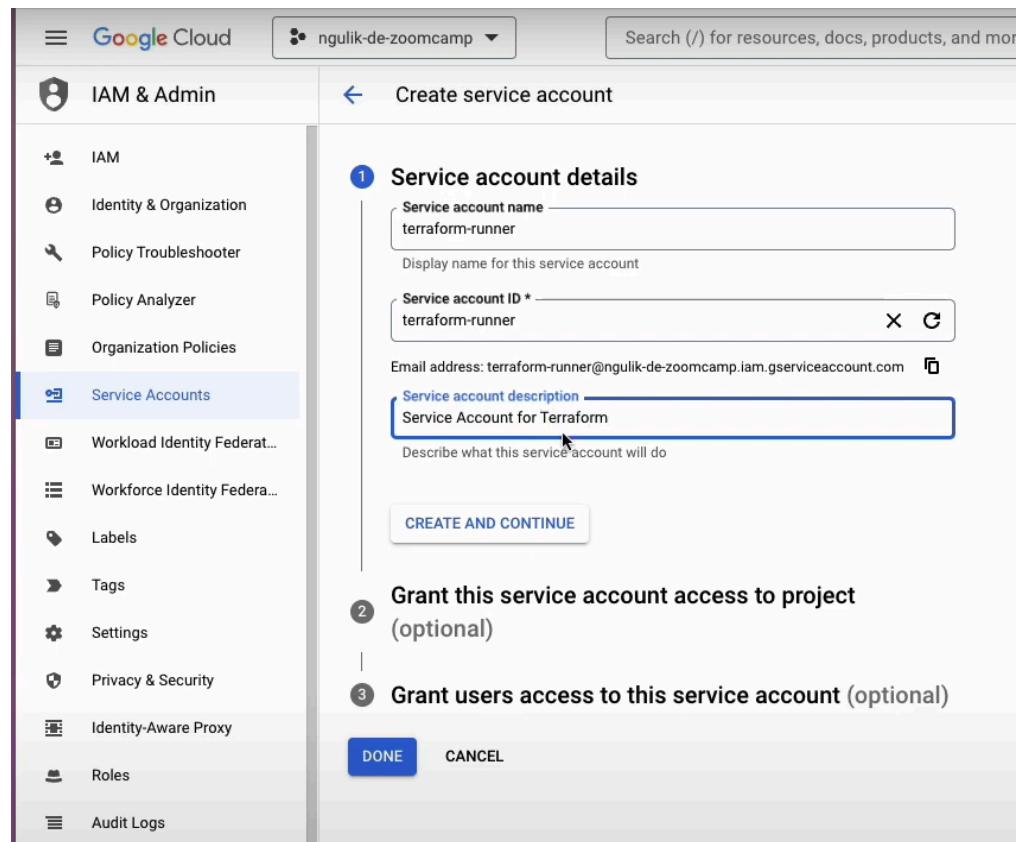
Analogi:

Provider adalah seperti tim subkontraktor khusus yang mengerti cara bekerja dengan material dan peraturan dari vendor tertentu. Dalam kasus ini, kita menggunakan tim yang berpengalaman dengan "material Google Cloud". Menentukan versi provider seperti memastikan tim mengikuti standar kerja yang spesifik yang kita kenal, sehingga tidak ada kejutan.

▼ 3. Menyiapkan Credentials (Service Account)

Langkah-langkah:

1. Buka Google Cloud Console: <https://console.cloud.google.com/>
2. Buat Project baru atau pilih yang sudah ada
3. Aktifkan billing jika belum
4. Navigasi ke IAM & Admin > Service Accounts



Service account details

Grant this service account access to project (optional)

Grant this service account access to ngulik-de-zoomcamp so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role — Storage Admin ▼ IAM condition (optional) ? [+ ADD IAM CONDITION](#) Delete

Grants full control of buckets and objects.

Role — BigQuery Admin ▼ IAM condition (optional) ? [+ ADD IAM CONDITION](#) Delete

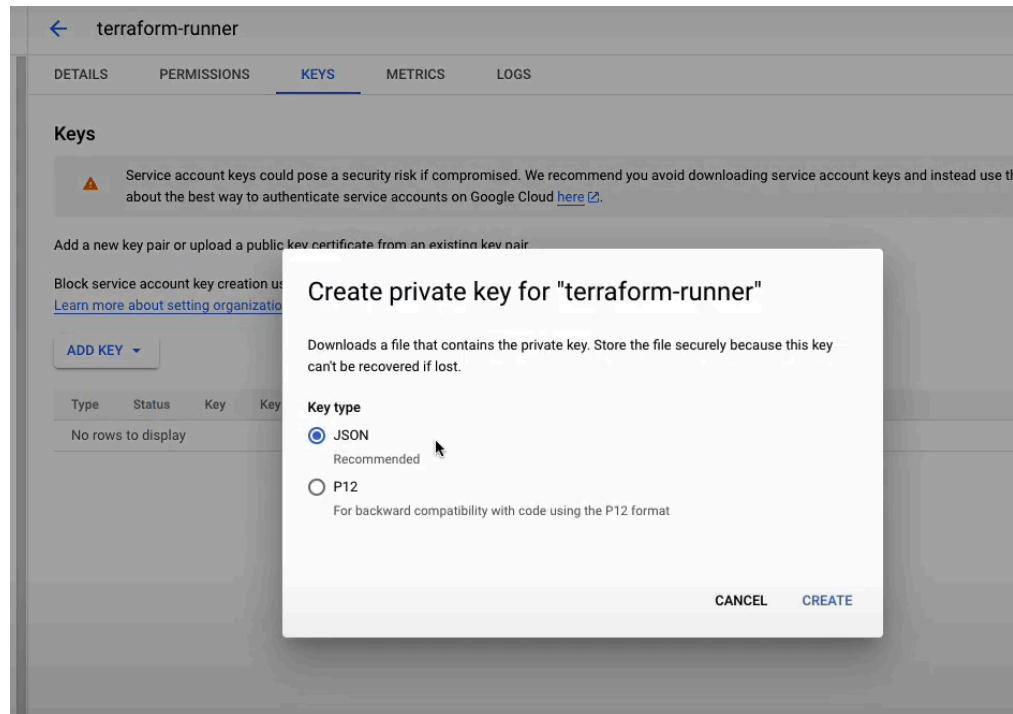
Administer all BigQuery resources and data

[+ ADD ANOTHER ROLE](#)

CONTINUE (optional)

3 Grant users access to this service account (optional)

DONE CANCEL



5. Buat Service Account baru dengan nama yang mudah diingat (misal: "terraform-account")
6. Berikan peran yang sesuai (minimal: Storage Admin, BigQuery Admin)
7. Buat kunci (key) untuk service account (format JSON)
8. Unduh file JSON dan simpan di lokasi aman
9. Dalam file Terraform, gunakan path ke credential:

```
provider "google" {  
  credentials = "PATH/TO/CREDENTIALS.json"  
  project    = "YOUR-PROJECT-ID"  
  region     = "REGION"  
}
```

Analogi:

Service Account adalah seperti surat izin konstruksi dan kunci akses ke lokasi pembangunan. File credentials adalah seperti ID badge khusus yang memungkinkan tim Anda masuk ke lokasi dan membangun di properti Google Cloud. Beda peran adalah seperti beda tingkat otorisasi - ada yang hanya bisa membangun gudang, ada yang bisa membangun seluruh kompleks.

▼ 4. Menjalankan Terraform Init

Perintah:

```
terraform init
```

Penjelasan:

`terraform init` mempersiapkan direktori kerja Anda dengan:

- Mengunduh provider yang ditentukan
- Menyiapkan file backend (mengelola state)
- Menginstal modul yang diperlukan

Analogi:

Ini seperti tahap persiapan lokasi konstruksi. Tim membawa semua alat dan material yang dibutuhkan ke lokasi, menyiapkan gudang penyimpanan material (state file), dan memastikan bahwa blueprint sejalan dengan peraturan lokal. `init` hanya dilakukan sekali di awal atau saat ada perubahan pada provider/backend.

▼ 5. Terraform Plan

Perintah:

```
terraform plan
```

Penjelasan:

`plan` membuat simulasi perubahan yang akan dilakukan, menunjukkan:

- Resources yang akan dibuat (+)
- Resources yang akan diubah (~)
- Resources yang akan dihapus (-)

Ini tidak melakukan perubahan nyata.

Analogi:

Ini seperti pemeriksaan blueprint dan simulasi virtual pembangunan. Kontraktor memvisualisasikan bagaimana bangunan akan terlihat dan memastikan tidak ada konflik atau masalah. Mereka menunjukkan, "Kita akan membangun bucket penyimpanan di sini dengan fitur-fitur ini," tetapi belum ada semen yang dituangkan.

```

PS D:\DE-Zoomcamp-Practice\DE-Zoomcamp-DwiANS-
2025\Module_1_Containerization&InfrastructureAsCode\02_Terraform> terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the
following symbols:
+ create

Terraform will perform the following actions:

# google_storage_bucket.data-lake-bucket will be created
+ resource "google_storage_bucket" "data-lake-bucket" {
    + effective_labels          = {
        + "goog-terraform-provisioned" = "true"
    }
    + force_destroy              = true
    + id                         = (known after apply)
    + location                   = "ASIA-SOUTHEAST2"
    + name                       = "dwians-de-zoomcamp-data-lake"
    + project                    = (known after apply)
    + project_number              = (known after apply)
    + public_access_prevention   = (known after apply)
    + rpo                        = (known after apply)
    + self_link                  = (known after apply)
    + storage_class               = "STANDARD"
    + terraform_labels            = {
        + "goog-terraform-provisioned" = "true"
    }
    + uniform_bucket_level_access = true
    + url                        = (known after apply)

    + lifecycle_rule {
        + action {
            + type      = "Delete"
            # (1 unchanged attribute hidden)
        }
        + condition {
            + age           = 30
            + matches_prefix = []
            + matches_storage_class = []
            + matches_suffix = []
            + with_state     = (known after apply)
            # (3 unchanged attributes hidden)
        }
    }
}

+ soft_delete_policy (known after apply)

+ versioning {
    + enabled = true
}

+ website (known after apply)
}

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

```

Note: You didn't use the `-out` option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "`terraform apply`" now.

▼ 6. Terraform Apply

Perintah:

```
terraform apply
```

atau untuk langsung mengeksekusi tanpa konfirmasi:

```
terraform apply -auto-approve
```

Penjelasan:

`apply` mengimplementasikan perubahan yang diusulkan dalam `plan`:

- Membuat resource baru
- Memperbarui resource yang sudah ada
- Menghapus resource yang tidak diperlukan lagi
- Mencatat state terbaru

Analogi:

Ini adalah fase pembangunan aktual. Tim konstruksi mulai membangun sesuai blueprint - menggali fondasi, menuangkan semen, membangun struktur. Dalam kasus kita, Google Cloud Storage Bucket sedang dibangun dengan semua spesifikasi yang kita tentukan (lokasi di Asia Tenggara, aturan lifecycle 30 hari, versioning, dll).

▼ 7. Terraform Destroy

Perintah:

```
terraform destroy
```

Penjelasan:

`destroy` menghapus semua resource yang dikelola oleh konfigurasi Terraform saat ini.

Analogi:

Ini seperti membongkar bangunan dan membersihkan lokasi konstruksi. Semua struktur yang telah dibangun (bucket penyimpanan) dihapus, dan lokasi dikembalikan ke kondisi semula. Berguna saat proyek sudah tidak diperlukan lagi, atau saat bereksperimen dan ingin "reset" infrastruktur.

▼ 8. Mengelola State

Penjelasan:

Terraform menyimpan informasi tentang infrastruktur di file state (`terraform.tfstate`). File ini crucial untuk operasi Terraform.

Analogi:

State file seperti catatan dan dokumentasi konstruksi resmi. Ini mencatat secara detail apa yang telah dibangun, dimensinya, material yang digunakan, dll. Jika ini hilang, tim tidak akan tahu apa yang sudah ada dan mungkin mencoba membangun sesuatu di tempat yang sudah ada bangunan.

▼ 9. Praktik Terbaik dan Tips

1. **Gunakan Version Control:** Simpan kode Terraform di Git
 - Analogi: Menyimpan revisi blueprint di tempat yang aman
2. **Modul Terraform:** Gunakan untuk kode yang dapat digunakan kembali
 - Analogi: Menggunakan desain modular yang dapat direplikasi di proyek lain
3. **Remote Backend:** Untuk kolaborasi tim
 - Analogi: Menyimpan blueprint dan dokumentasi di kantor pusat agar semua tim dapat mengakses
4. **Variabel:** Gunakan untuk nilai yang berubah-ubah
 - Analogi: Menggunakan parameter yang dapat disesuaikan pada blueprint

▼ 10. Menjalankan Contoh Kode Anda

Untuk konfigurasi Terraform yang Anda tunjukkan:

```

# Google Cloud Storage Bucket
resource "google_storage_bucket" "data-lake-bucket" {
    name      = "dwians-de-zoomcamp-data-lake"
    location   = "asia-southeast2"
    storage_class = "STANDARD"
    uniform_bucket_level_access = true

    versioning {
        enabled    = true
    }

    lifecycle_rule {
        action {
            type = "Delete"
        }
        condition {
            age = 30 // days
        }
    }

    force_destroy = true
}

```

Konfigurasi bucket Anda mirip dengan membangun gudang penyimpanan digital:

- `name` : Alamat resmi gudang
- `location` : Lokasi fisik (Jakarta/Asia Tenggara)
- `storage_class` : Kualitas konstruksi gudang (STANDARD adalah konstruksi standar)
- `versioning` : Sistem pelacakan sejarah (menyimpan versi barang lama)
- `lifecycle_rule` : Aturan pembersihan otomatis (membuang barang setelah 30 hari)
- `force_destroy` : Izin untuk menghancurkan gudang meskipun masih ada barang di dalamnya

▼ Terraform Variables

▼ Apa itu Terraform Variables?

Terraform variables adalah parameter yang digunakan untuk menyimpan dan mengatur nilai-nilai dalam konfigurasi Terraform. Variabel ini memungkinkan Anda memisahkan nilai-nilai spesifik dari kode infrastruktur Anda, sehingga konfigurasi menjadi lebih fleksibel dan dapat digunakan kembali.

▼ Analogi: Resep Masakan yang Dapat Disesuaikan

Jika infrastruktur Terraform dianalogikan sebagai resep masakan, maka:

- **Kode Terraform (`main.tf`)** adalah instruksi dasar cara memasak
- **Variables** adalah daftar bahan yang dapat disesuaikan (garam, gula, jumlah porsi)
- **`terraform.tfvars`** adalah catatan spesifik tentang bahan yang akan digunakan untuk masakan hari ini

Tanpa variabel, Anda seperti membuat resep yang kaku: "tambahkan tepat 500g tepung". Dengan variabel, resep menjadi: "tambahkan tepung sesuai jumlah di daftar bahan", sehingga dapat disesuaikan untuk situasi berbeda.

▼ Kegunaan Terraform Variables

1. **Reusabilitas:** Gunakan konfigurasi yang sama di lingkungan berbeda (development, staging, production)
2. **Keamanan:** Pisahkan informasi sensitif (seperti kredensial) dari kode
3. **Kolaborasi Tim:** Memudahkan kerja tim dengan memisahkan nilai-nilai yang dapat berubah
4. **Fleksibilitas:** Ubah nilai tanpa mengubah kode inti infrastruktur
5. **Dokumentasi:** Variables berfungsi sebagai dokumentasi nilai-nilai yang dapat dikonfigurasi

▼ Implementasi Terraform Variables

Berikut implementasi untuk infrastruktur Anda menggunakan variables:

1. File: variables.tf

```
# General variables
variable "project" {
    description = "Google Cloud Project ID"
    type       = string
}

variable "region" {
    description = "Region for GCP resources"
    type       = string
    default    = "asia-southeast2"
}

variable "credentials_file" {
    description = "Path to the service account credentials JSON file"
    type       = string
}

# GCS variables
variable "bucket_name" {
    description = "The name of the Google Cloud Storage bucket"
    type       = string
}

variable "storage_class" {
    description = "Storage class type for your bucket"
    type       = string
    default    = "STANDARD"
}

variable "bucket_location" {
    description = "Location of the bucket"
    type       = string
    default    = "asia-southeast2"
}

variable "lifecycle_age" {
```

```

description = "Number of days before object deletion"
type      = number
default    = 30
}

# BigQuery variables
variable "bq_dataset_id" {
  description = "BigQuery Dataset ID"
  type      = string
}

variable "bq_dataset_location" {
  description = "BigQuery Dataset Location"
  type      = string
  default    = "asia-southeast2"
}

```

2. File: **terraform.tfvars**

```

# General configuration
project      = "de-zoomcamp-2025-454709"
region       = "asia-southeast2"
credentials_file = "D:\\\\DE-Zoomcamp-Practice\\\\DE-Zoomcamp
-DwiANS-2025\\\\Module_1_Containerization&InfrastructureAsCod
e\\\\02_Terraform\\\\Keys\\\\my-creds.json"

# GCS configuration
bucket_name   = "dwians-de-zoomcamp-data-lake"
storage_class = "STANDARD"
bucket_location = "asia-southeast2"
lifecycle_age = 30

# BigQuery configuration
bq_dataset_id     = "de_zoomcamp_dataset"
bq_dataset_location = "asia-southeast2"

```

3. File: **main.tf** (Dimodifikasi)

```

terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "6.17.0"
    }
  }
}

provider "google" {
  credentials = var.credentials_file
  project    = var.project
  region     = var.region
}

# Google Cloud Storage Bucket
resource "google_storage_bucket" "data-lake-bucket" {
  name        = var.bucket_name
  location    = var.bucket_location

  # Optional, but recommended settings:
  storage_class = var.storage_class
  uniform_bucket_level_access = true

  versioning {
    enabled  = true
  }

  lifecycle_rule {
    action {
      type = "Delete"
    }
    condition {
      age = var.lifecycle_age // days
    }
  }

  force_destroy = true
}

```

```

}

# Google BigQuery Dataset
resource "google_bigquery_dataset" "dataset" {
    dataset_id = var.bq_dataset_id
    location   = var.bq_dataset_location
}

```

▼ Cara Menggunakan Terraform Variables

1. **Definisi:** Definisikan variabel dalam `variables.tf`
2. **Nilai Default:** Bisa ditambahkan langsung dalam definisi variabel
3. **Nilai Spesifik:** Simpan di file `terraform.tfvars` (otomatis diload) atau `.auto.tfvars`
4. **Passing Values:** Saat runtime via CLI dengan `var 'name=value'` atau `var-file=custom.tfvars`
5. **Environment Variables:** Menggunakan `TF_VAR_name`

▼ Analogi Lengkap: Membangun Rumah yang Kustomisasi

Bayangkan Terraform sebagai jasa kontraktor pembangunan rumah:

1. **Tanpa Variables:** Anda memberikan blueprint dengan semua detail tetap: "Buat rumah dengan 3 kamar di Jakarta, cat biru, lantai marmer." Jika ingin rumah dengan 4 kamar, Anda harus membuat blueprint baru.
2. **Dengan Variables:** Anda memberikan blueprint yang fleksibel: "Buat rumah dengan {jumlah_kamar} di {lokasi}, cat {warna}, lantai {material}." Blueprint tetap sama, tetapi Anda bisa mengubah parameter untuk mendapatkan rumah yang berbeda.

Sekarang `main.tf` adalah instruksi umum ("cara membangun rumah"), `variables.tf` adalah daftar hal yang dapat disesuaikan ("parameter apa yang bisa diubah"), dan `terraform.tfvars` adalah spesifikasi akhir ("untuk proyek ini, kita ingin 3 kamar di Jakarta").

Pendekatan ini memungkinkan tim Anda membangun berbagai "rumah" (infrastruktur) dengan desain dasar yang sama tetapi dengan

karakteristik yang berbeda, tanpa harus menulis ulang blueprint untuk setiap proyek.

▼ Keuntungan Menggunakan Variables dalam Proyek Data Engineering

Dalam konteks Data Engineering, variabel Terraform memungkinkan Anda:

1. **Ruang Lingkup Berbeda:** Menggunakan konfigurasi yang sama untuk data lake development (kecil) dan production (besar)
2. **Scaling Mudah:** Menyesuaikan parameter seperti ukuran storage atau retention policy
3. **Multi-Region:** Menjalankan infrastruktur yang sama di berbagai region Google Cloud
4. **Tim Collaboration:** Memungkinkan insinyur yang berbeda menyesuaikan parameter tanpa mengubah kode utama

Dengan implementasi ini, Anda sekarang memiliki infrastruktur yang lebih fleksibel dan dapat digunakan kembali untuk berbagai kebutuhan data engineering Anda.