DigitalSkola
Uncover The World of Digital Skills with Us

LEARNING PROGRESS REVIEW

"WEEK 8"

BY : DREAM (KELOMPOK 1)

Data Realm Engineers And Maestros

ANGGOTA KELOMPOK

01 AFROH FAUZIAH

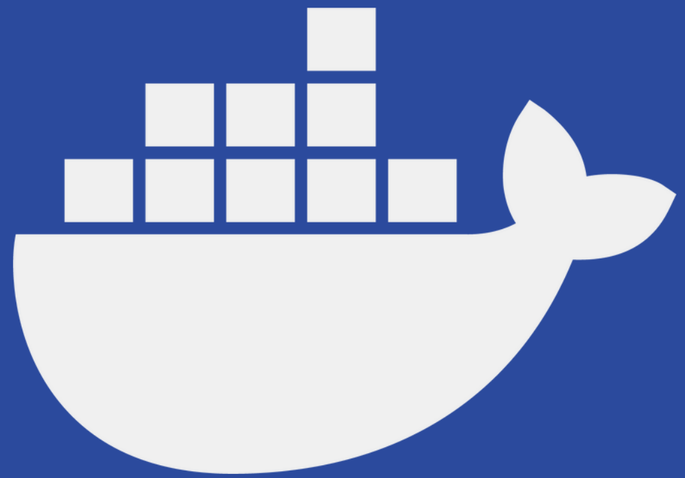
02 ALTHAF NAWADIR TAQIYYAH

03 ANDI ROSILALA

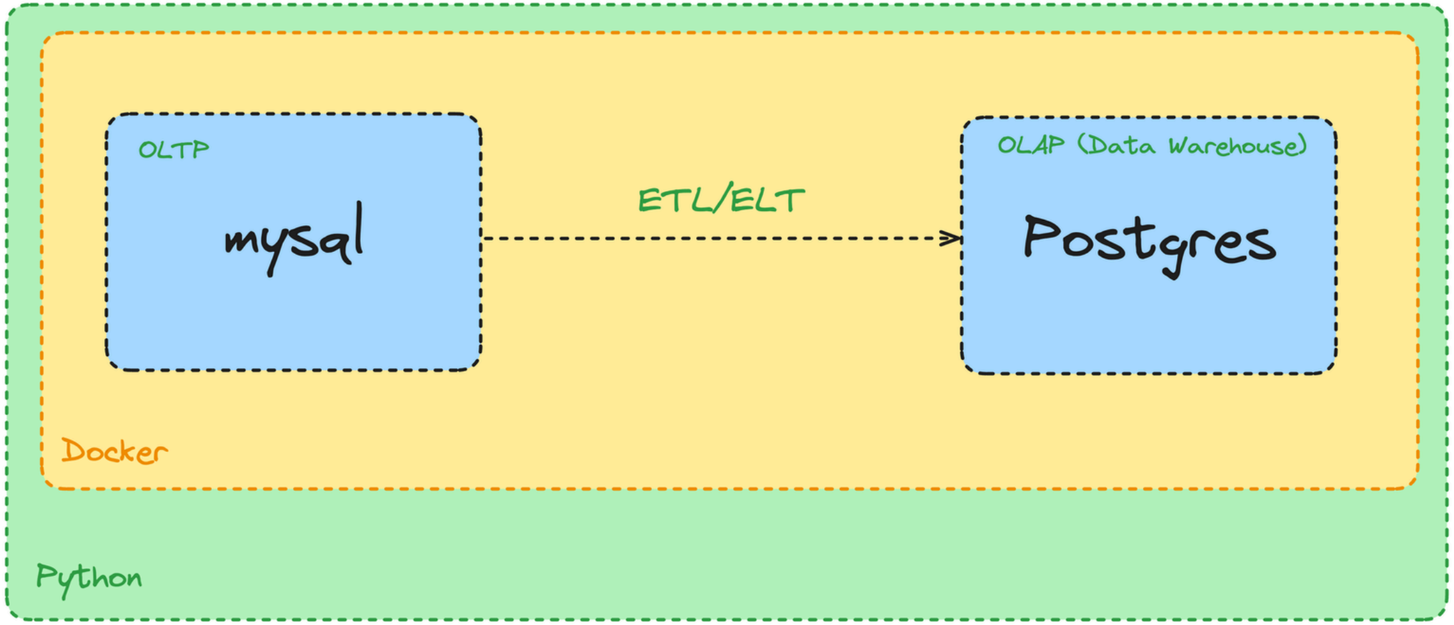
04 ANDREW BINTANG PRATAMA

05 ANDREW FORTINO MAHARDIKA SUADNYA

PROJECT 4

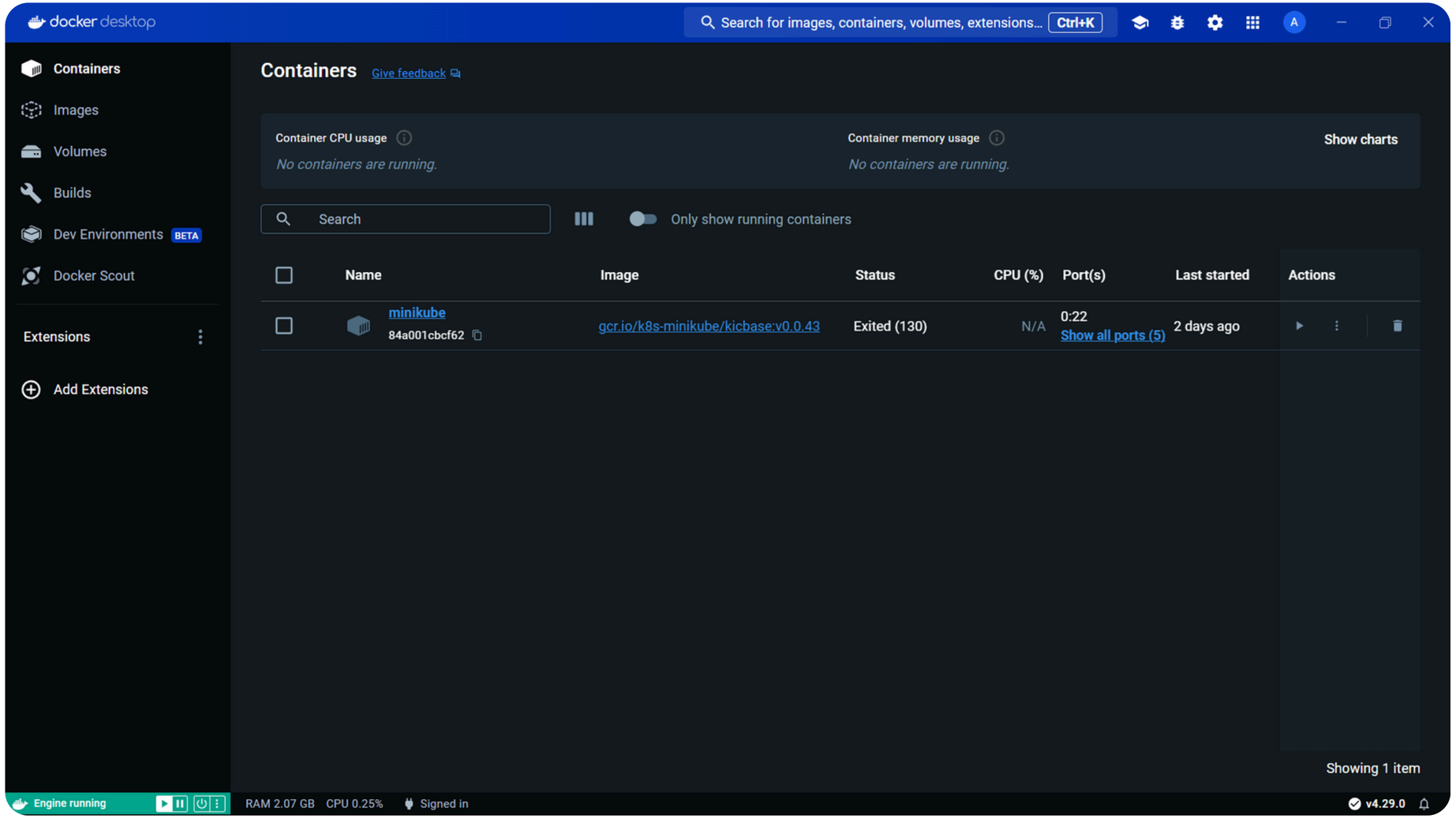


PROJECT 4
ETL WITH DOCKER



Project 4 melibatkan implementasi proses ETL dengan menggunakan Docker. Proses ini mencakup pemindahan data dari MySQL ke PostgreSQL (Data Warehouse).

STEP 1: Menjalankan Docker Menggunakan Docker Desktop



STEP 2: Menyiapkan Data Dummy Dalam Bentuk CSV

Menggunakan data channel youtube (global_yotube_stat.csv) yang nantinya akan diload ke dalam database mysql.

DigitalSkola
Uncover The World of Digital Skills with Us

STEP 3: Siapkan file docker-compose.yaml

```yaml
version: "3.7"
services:
  db-mysql:
    image: mysql:8
    container_name: db-mysql
    environment:
      MYSQL_ROOT_PASSWORD: mysql
      MYSQL_USER: mysql
      MYSQL_PASSWORD: mysql
      MYSQL_DATABASE: operational
    volumes:
      # - ./init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./data:/docker-entrypoint-initdb.d/init_data
    ports:
      - "3305:3306" # local:docker

  db-psql:
    image: postgres:11
    container_name: db-postgres
    environment:
      POSTGRES_USER: postgres1
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
```

File docker-compose.yaml ini mendefinisikan dua layanan database yaitu db-mysql dan db-psql, yang masing-masing merupakan kontainer untuk MySQL dan PostgreSQL.

STEP 4: Jalankan kontainer

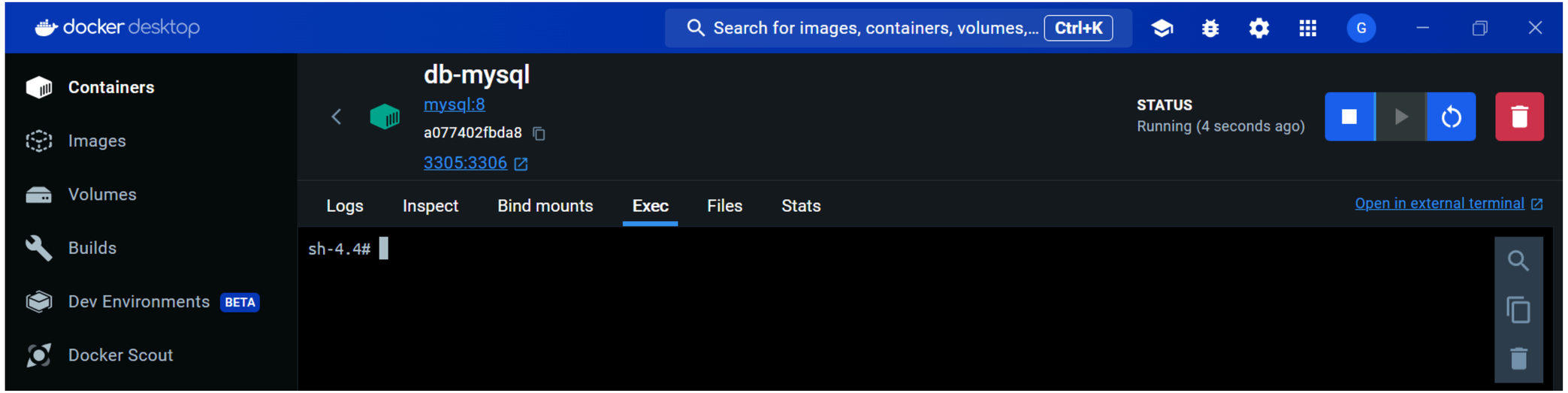
Jalankan container MySQL dan PostgreSQL secara deklaratif sesuai dengan konfigurasi pada file yaml dengan command `docker-compose up`.

```
\docker_etl\examples\etl>docker-compose up
```

```
[+] Running 2/0
 ✓ Container db-postgres  Created        0.0s
 ✓ Container db-mysql     Created        0.0s
```

STEP 5: Masuk ke Terminal Kontainer db-mysql

1 MENGGUNAKAN DOCKER DESKTOP

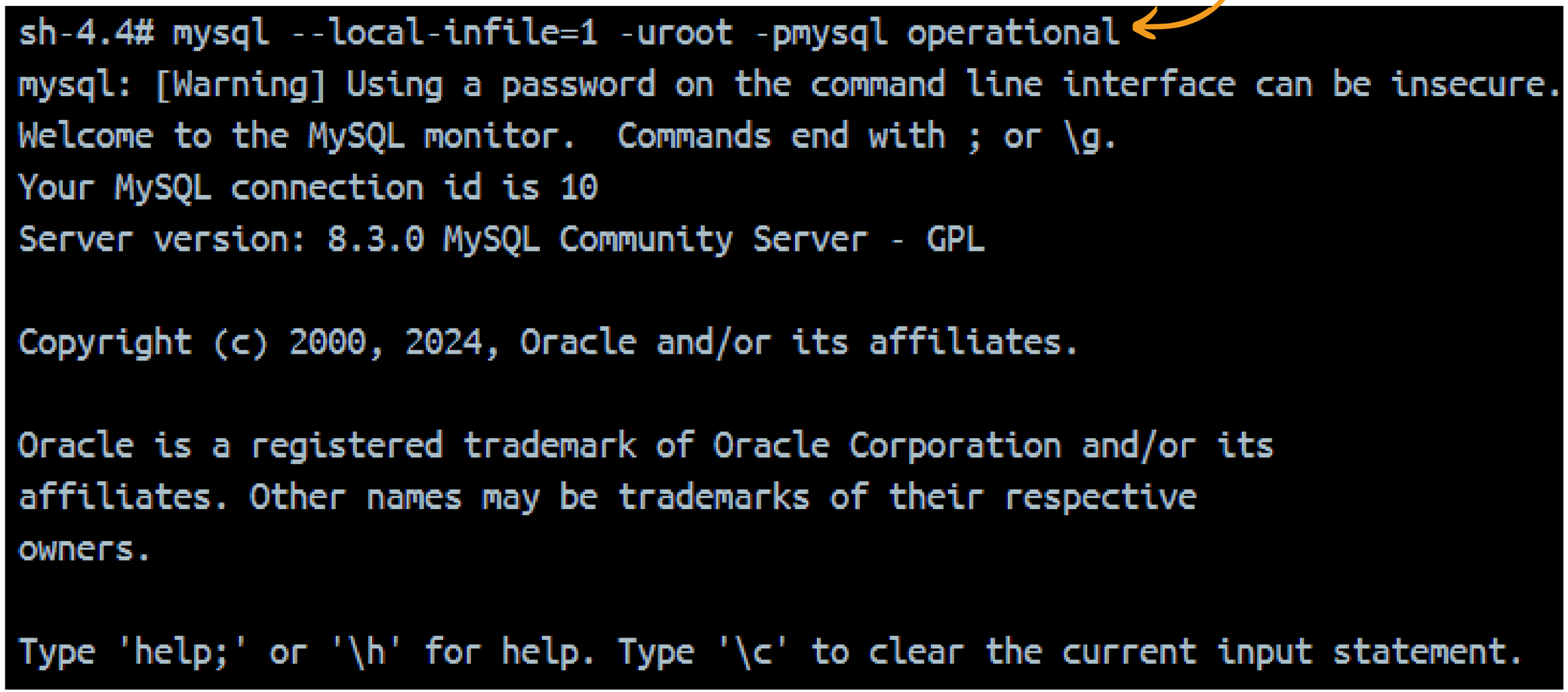


2 MENGGUNAKAN TERMINAL - Command: `docker exec -it <container_id> bash`

```
\docker_etl\examples\etl>docker exec -it fb9a5238ecf4 bash
```

STEP 6: Masuk ke database mysql

~ Gunakan command: `mysql --local-infile=1 -uroot -pmysql operational` ~



init.sql

examples > etl > init.sql

```sql
CREATE DATABASE IF NOT EXISTS operational;

SET GLOBAL local_infile=1;

USE operational;

CREATE TABLE IF NOT EXISTS youtube (
`rank` INT(5),
youtuber VARCHAR(50),
subscribers BIGINT,
video_views VARCHAR(50),
category VARCHAR(50),
title VARCHAR(50),
uploads INT(10),
country VARCHAR(50),
abbreviation VARCHAR(50),
channel_type VARCHAR(50),
video_views_rank INT(10),
country_rank INT(11),
channel_type_rank VARCHAR(50),
video_views_for_the_last_30_days INT(10),
lowest_monthly_earnings INT(10),
highest_monthly_earnings INT(10),
lowest_yearly_earnings INT(10),
highest_yearly_earnings INT(10),
subscribers_for_last_30_days INT(10),
created_year INT(10),
created_month VARCHAR(50),
created_date INT(5),
gross_tertiary_education_enrollment_percent FLOAT,
population BIGINT,
unemployment_rate FLOAT,
urban_population BIGINT,
latitude FLOAT,
longitude FLOAT
);

LOAD DATA LOCAL INFILE '/docker-entrypoint-initdb.d/init_data/global_youtube_stat.csv'
INTO TABLE youtube
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

STEP 7: Membuat File init.sql yang berisikan syntax ddl dan syntax untuk meload data global_yotube_stat.csv

DigitalSkola
Uncover The World of Digital Skills with Us

STEP 8: Menjalankan Syntax" dari init.sql

```
mysql> CREATE DATABASE IF NOT EXISTS operational;
Query OK, 1 row affected, 1 warning (0.02 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| operational        |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Pertama membuat database "operational" kemudian menampilkannya dengan perintah berikut:

```
mysql> use operational;
Database changed
mysql> show tables;
Empty set (0.01 sec)
```

Selanjutnya menjalankan perintah 'use operational' yaitu untuk menggunakan database 'operational' yang dibuat sebelumnya.

```
mysql> CREATE TABLE IF NOT EXISTS youtube (
    -> `rank` INT(5),
    -> youtuber VARCHAR(50),
    -> subscribers BIGINT,
    -> video_views VARCHAR(50),
    -> category VARCHAR(50),
    -> title VARCHAR(50),
    -> uploads INT(10),
    -> country VARCHAR(50),
    -> abbreviation VARCHAR(50),
    -> channel_type VARCHAR(50),
    -> video_views_rank INT(10),
    -> country_rank INT(11),
    -> channel_type_rank VARCHAR(50),
    -> video_views_for_the_last_30_days INT(10),
    -> lowest_monthly_earnings INT(10),
    -> highest_monthly_earnings INT(10),
    -> lowest_yearly_earnings INT(10),
    -> highest_yearly_earnings INT(10),
    -> subscribers_for_last_30_days INT(10),
    -> created_year INT(10),
    -> created_month VARCHAR(50),
    -> created_date INT(5),
    -> gross_tertiary_education_enrollment_percent FLOAT,
    -> population BIGINT,
    -> unemployment_rate FLOAT,
    -> urban_population BIGINT,
    -> latitude FLOAT,
    -> longitude FLOAT
    -> );
Query OK, 0 rows affected, 12 warnings (0.13 sec)
```

Next yaitu membuat tabel “youtube” dengan nama kolom dan tipe data sebagai berikut:
(berdasarkan data dari file csv)

```
mysql> show tables;
+----------------------+
| Tables_in_operational |
+----------------------+
| youtube              |
+----------------------+
1 row in set (0.01 sec)
```

Menjalankan perintah ‘show tables’ untuk menampilkan tabel “youtube” yang telah dibuat sebelumnya.

DigitalSkola
Uncover The World of Digital Skills with Us

Memuat data dari file CSV ke tabel 'youtube'

```
mysql> LOAD DATA LOCAL INFILE '/docker-entrypoint-initdb.d/init_data/global_youtube_stat.csv'
    -> INTO TABLE youtube
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
ERROR 3948 (42000): Loading local data is disabled; this must be enabled on both the client and server sides
```

Disini terdapat eror: "*Loading local data is disabled; this must be enabled on both the client and server sides*".

Untuk debugnya yaitu mengaktifkan penggunaan local_infile untuk mengizinkan penggunaan perintah LOAD DATA LOCAL INFILE dengan perintah: *'SET GLOBAL local_infile=1;'*

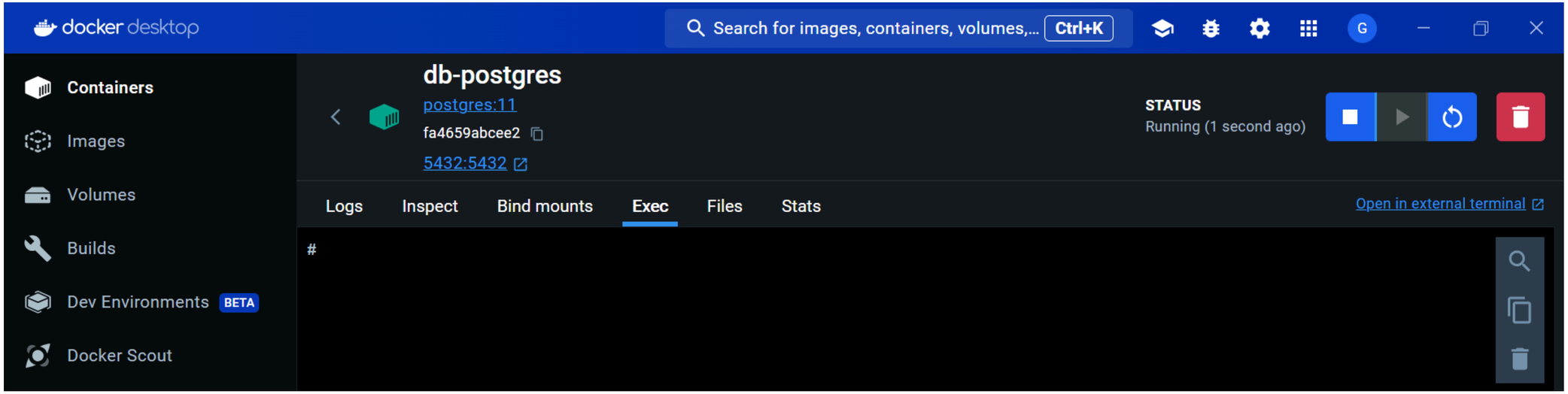
Menjalankan kembali perintah untuk memuat data dari file CSV ke tabel 'youtube'

```
mysql> SET GLOBAL local_infile=1;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD DATA LOCAL INFILE '/docker-entrypoint-initdb.d/init_data/global_youtube_stat.csv'
    -> INTO TABLE youtube
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 995 rows affected, 1265 warnings (0.16 sec)
Records: 995  Deleted: 0  Skipped: 0  Warnings: 1265
```

STEP 9: Masuk ke Terminal Kontainer db-postgresql

1 MENGGUNAKAN DOCKER DESKTOP



2 MENGGUNAKAN TERMINAL - Command: `docker exec -it <container_id> bash`

```
C:\Users\LENOVO\Stupen\SIB-Project4\docker_etl>docker exec -it fa4659abcee2 bash
root@fa4659abcee2:/#
```

STEP 10: Masuk ke database postgresql dan buat database data_warehouse

Gunakan command: `psql -Upostgres` untuk masuk ke db postgres.

```
# psql -Upostgres
psql (11.16 (Debian 11.16-1.pgdg90+1))
Type "help" for help.

postgres=#
```

```
postgres=# create database data_warehouse;
CREATE DATABASE
```

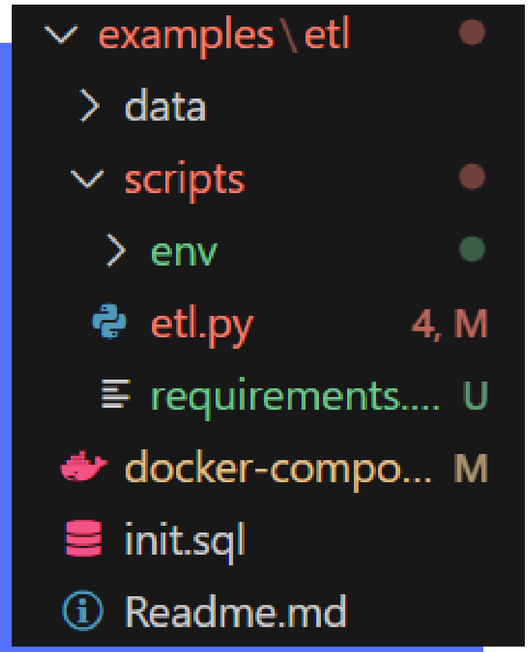
Buat database baru dengan nama data_warehouse.

```
postgres-# \l
                                  List of databases
     Name      |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
---------------+----------+----------+------------+------------+-----------------------
 data_warehouse | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres       | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0      | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
                |          |          |            |            | postgres=CTc/postgres
 template1      | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
                |          |          |            |            | postgres=CTc/postgres
(4 rows)
```

Database data_warehouse berhasil dibuat

STEP 11: Buat script untuk melakukan proses ETL

~ struktur folder ~



~ requirements ~

requirements.txt

examples > etl > scripts > requirements.txt

```
pandas
mysql-connector-python
Flask-SQLAlchemy
psycopg2-binary
```

~ isi etl.py ~

```python
import mysql.connector as connection
import pandas as pd
import time
from sqlalchemy import create_engine
import psycopg2


try:
    # ========== Read data from MySQL ================
    mydb = connection.connect(
            host="localhost",
            database='operational',
            user="root",
            passwd="mysql",
            use_pure=True,
            port=3305)

    query = "Select * from youtube;"
    result_dataFrame = pd.read_sql(query, mydb)
    print(result_dataFrame)

    # ========== Write data to Postgresql ================
    # INPUT YOUR OWN CONNECTION STRING HERE
    conn_string = 'postgresql://postgres1:postgres@localhost:5432/data_warehouse'

    # perform to_sql test and print result
    db = create_engine(conn_string, pool_size=10, max_overflow=20)
    conn = db.connect()

    start_time = time.time()
    result_dataFrame.to_sql('youtube_etl',
                            con=conn,
                            if_exists='replace',
                            index=False)
    print("to_sql duration: {} seconds".format(time.time() - start_time))

    mydb.close()  # close the connection
except Exception as e:
    mydb.close()
    print(str(e))
```

STEP 12: Jalankan script ETL

```
(env) C:\Users\afroh\docker_etl\examples\etl\scripts>python C:\Users\afroh\docker_etl\examples\etl\scripts\etl.py
C:\Users\afroh\docker_etl\examples\etl\scripts\etl.py:9: UserWarning: pandas only supports SQLAlchemy connectable (engine/co
nnection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQ
LAlchemy.
  result_dataFrame = pd.read_sql(query, mydb)
     rank                  youtuber  subscribers  video_views  ... unemployment_rate urban_population  latitude longitude
0       1                   T-Series    245000000     2.28E+11  ...             5.36        471031528   20.5937  78.96290
1       2              YouTube Movies    170000000            0  ...            14.70        270663028   37.0902 -95.71290
2       3                   MrBeast    166000000  28368841870  ...            14.70        270663028   37.0902 -95.71290
3       4  Cocomelon - Nursery Rhymes    162000000     1.64E+11  ...            14.70        270663028   37.0902 -95.71290
4       5                 SET India    159000000     1.48E+11  ...             5.36        471031528   20.5937  78.96290
..    ...                       ...          ...          ...  ...              ...              ...       ...       ...
990   991               Natan por A      12300000   9029609749  ...            12.08        183241641  -14.2350 -51.92530
991   992    Free Fire India Official    12300000   1674409945  ...             5.36        471031528   20.5937  78.96290
992   993                     Panda      12300000   2214684303  ...             3.85         55908316   55.3781  -3.43597
993   994                RobTopGames     12300000    374123483  ...             6.48          9021165   60.1282  18.64350
994   995                Make Joke Of    12300000   2129773714  ...             5.36        471031528   20.5937  78.96290

[995 rows x 28 columns]
```

Jalankan script etl.py menggunakan command `python (path to) etl.py`

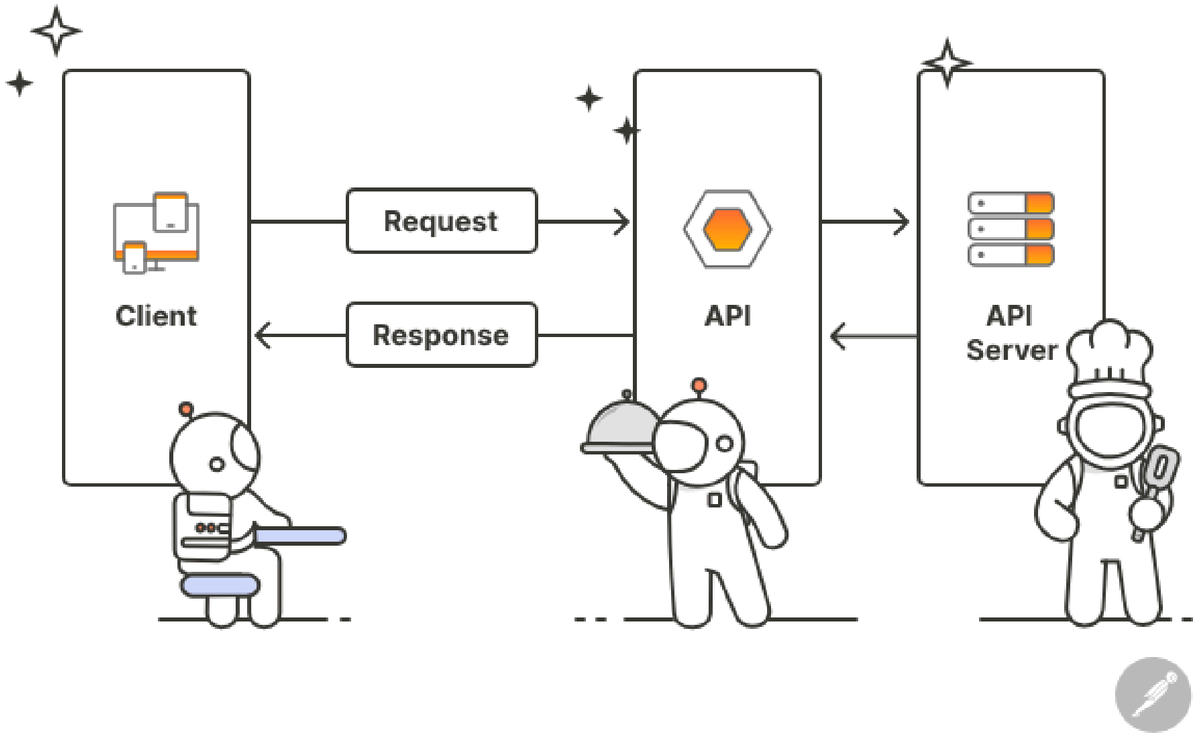
Connect ke database data_warehouse pada postgresql menggunakan command: `\c data_warehouse`.

Kemudian cek apakah tabel youtube_etl sudah ada di dalam database tersebut menggunakan command `\d`.

```
postgres=# \c data_warehouse
You are now connected to database "data_warehouse" as user "postgres".
data_warehouse=# \d
              List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | youtube_etl | table | postgres
(1 row)
```

API I
&
API II

API INTRODUCTION



API - Application Programming Interface adalah jembatan yang menghubungkan berbagai aplikasi dan memungkinkan aplikasi-aplikasi tersebut berinteraksi untuk berbagi data dan berfungsi satu sama lain.

DigitalSkola

PROTOKOL API

SOAP
(Simple Object Access protocol)

- SOAP adalah protokol komunikasi yang dirancang untuk komunikasi melalui jaringan.
- SOAP menggunakan format XML sebagai object data
- SOAP menggunakan HTTP request & response

REST
(Representational State Transfer)

- REST adalah arsitektur gaya yang biasanya berjalan di atas HTTP.
- REST menggunakan format JSON sebagai object data.
- REST menggunakan HTTP request & response (GET, POST, PUT, DELETE)

METODE API

GET
(Getting a resource from the server)

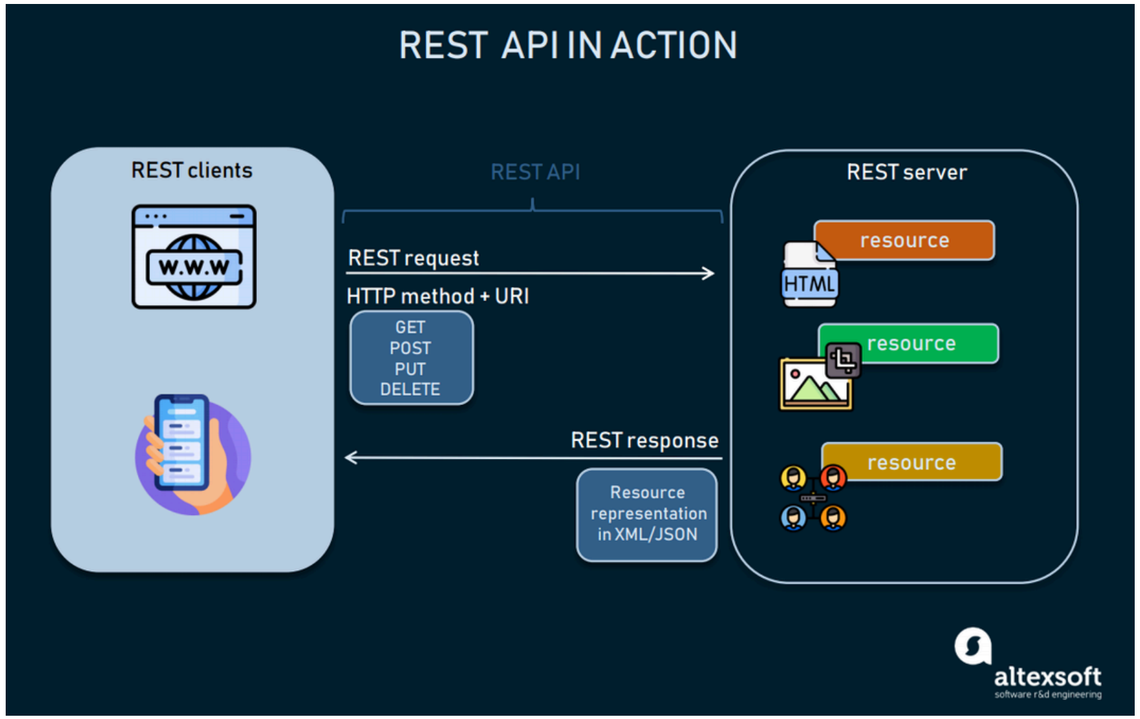
POST
(Creating or appending a resource at the server)

PUT
(Create or over-write a resource)

DELETE
(Delete a resource)

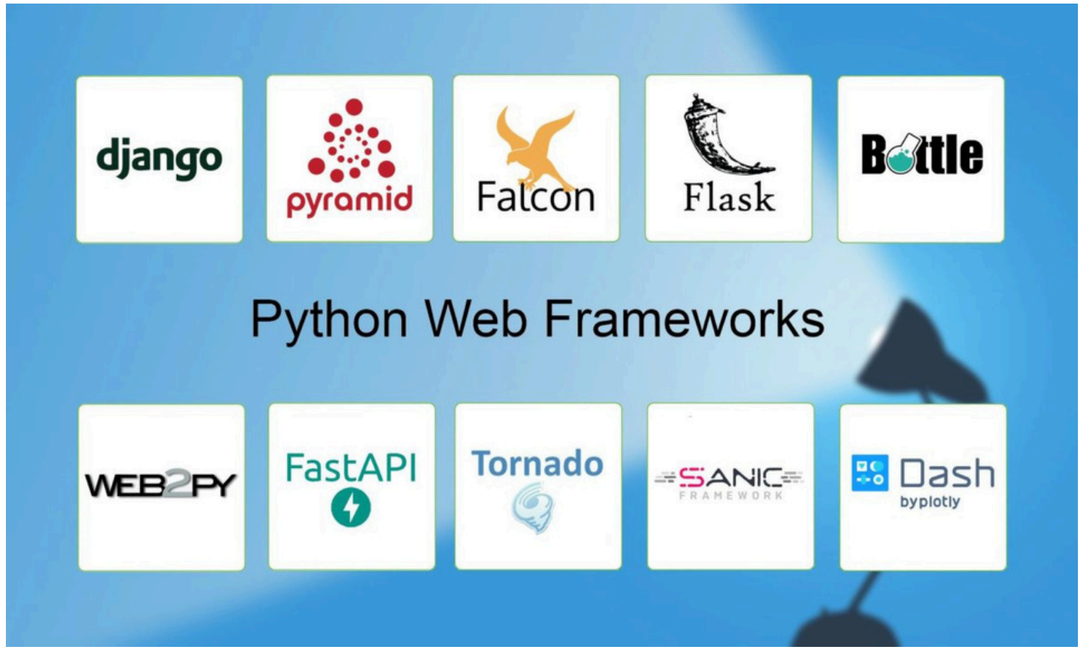
REST API merupakan jenis API (Application Programming Interface) yang menggunakan permintaan HTTP untuk mengakses dan menggunakan data. Data tersebut dapat digunakan untuk tipe data GET, POST, PUT, dan DELETE, yang merupakan empat metode HTTP utama yang digunakan dalam REST API.

REST API



1. GET: Metode ini digunakan untuk mengambil data dari server.
2. POST: Metode ini digunakan untuk mengirim data baru ke server.
3. PUT: Metode ini digunakan untuk memperbarui data yang sudah ada di server.
4. DELETE: Metode ini digunakan untuk menghapus data dari server.

PYTHON WEB FRAMEWORK(FLASK)



Web framework adalah sebuah kerangka kerja yang dirancang untuk membantu pengembangan aplikasi web (termasuk API di dalamnya). Framework ini memudahkan pengembangan aplikasi web dengan menyediakan berbagai fitur dan fungsionalitas yang dapat digunakan oleh pengembang. Dengan menggunakan framework, pengembang tidak perlu mengembangkan aplikasi web dari nol, sehingga dapat menghemat waktu dan tenaga.

IMPLEMENTASI FLASK

Instalasi Flask

```
C:\Users\ASUS>pip install flask
Requirement already satisfied: flask in c:\python310\lib\site-packages (3.0.2)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\python310\lib\site-packages (from flask) (3.0.1)
Requirement already satisfied: Jinja2>=3.1.2 in c:\python310\lib\site-packages (from flask) (3.1.3)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\python310\lib\site-packages (from flask) (2.1.2)
Requirement already satisfied: click>=8.1.3 in c:\python310\lib\site-packages (from flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in c:\python310\lib\site-packages (from flask) (1.7.0)
Requirement already satisfied: colorama in c:\python310\lib\site-packages (from click>=8.1.3->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\python310\lib\site-packages (from Jinja2>=3.1.2->flask) (2.1.5)
```

Import flask

```
from flask import Flask
```

Inisialisasi flask

```
app = Flask(__name__)
```

IMPLEMENTASI FLASK

Definisikan rute

```
@app.route('/')
def index():
    return 'Halo, dunia!'
```

Jalankan aplikasi

```
if __name__ == '__main__':
    app.run(debug=True)
```

CONNECTION TO DATABASE USING PSYCOPG2

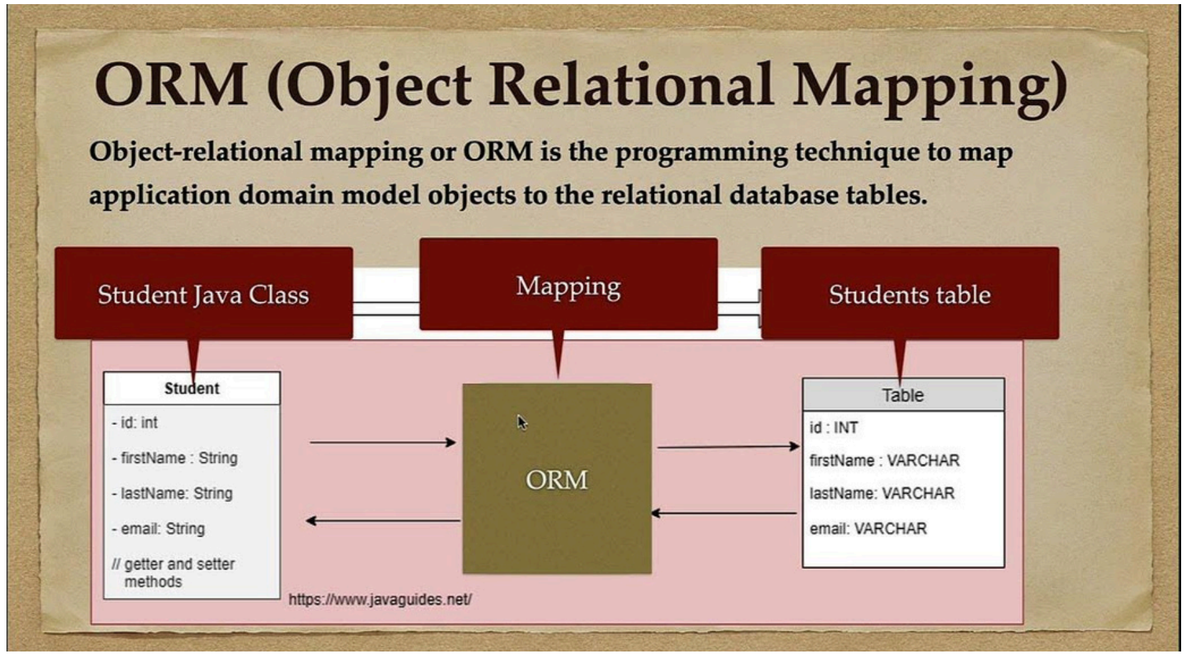
```
import psycopg2

try:
connection = psycopg2.connect( host="http://localhost", user="postgres", password="postgres",
database="postgres", port=5432
)
cursor = connection.cursor(cursor_factory=RealDictCursor) except Exception as e:
raise e
```

Tujuan dari kode tersebut adalah untuk melakukan koneksi ke database PostgreSQL dari aplikasi Python menggunakan modul psycopg2

DigitalSkola
Uncover The World of Digital Skills with Us

OBJECT-RELATIONAL MAPPING(ORM)



ORMapping, atau Object-Relational Mapping, adalah teknik pemrograman yang menghubungkan objek model domain aplikasi dengan tabel database relasional. Dalam istilah yang lebih sederhana, ini adalah cara untuk menjembatani kesenjangan antara dunia pemrograman berorientasi objek dan dunia database relasional.

IMPLEMENTASI ORM

```python
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

# Buat kelas basis untuk ORM
Base = declarative_base()

# Definisikan kelas yang mewakili tabel dalam database
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    username = Column(String)
    email = Column(String)
```

ORM dalam Python menggunakan SQLAlchemy

```python
# Buat engine untuk koneksi ke database
engine = create_engine('sqlite:///example.db')

# Buat tabel di dalam database (jika belum ada)
Base.metadata.create_all(engine)

# Buat session untuk berinteraksi dengan database
Session = sessionmaker(bind=engine)
session = Session()

# Tambahkan data ke dalam tabel menggunakan objek
new_user = User(username='john', email='john@example.com')
session.add(new_user)
session.commit()

# Query data dari tabel dan tampilkan
users = session.query(User).all()
for user in users:
    print(user.username, user.email)

# Tutup session
session.close()
```

ORM: STORING DATA USING ORM

Buat instance kelas Python dan tambahkan ke sesi. Setelah ditambahkan ke sesi, perubahan pada instans ini akan dilacak, dan terapkan ke database menggunakan metode ini .commit()

```
# Create instances of the Product class
product1 = Product(name='Laptop', price=1000)
product2 = Product(name='Mouse', price=20)

# Add instances to the session
session.add(product1)
session.add(product2)

# Commit changes to the database
session.commit()
```

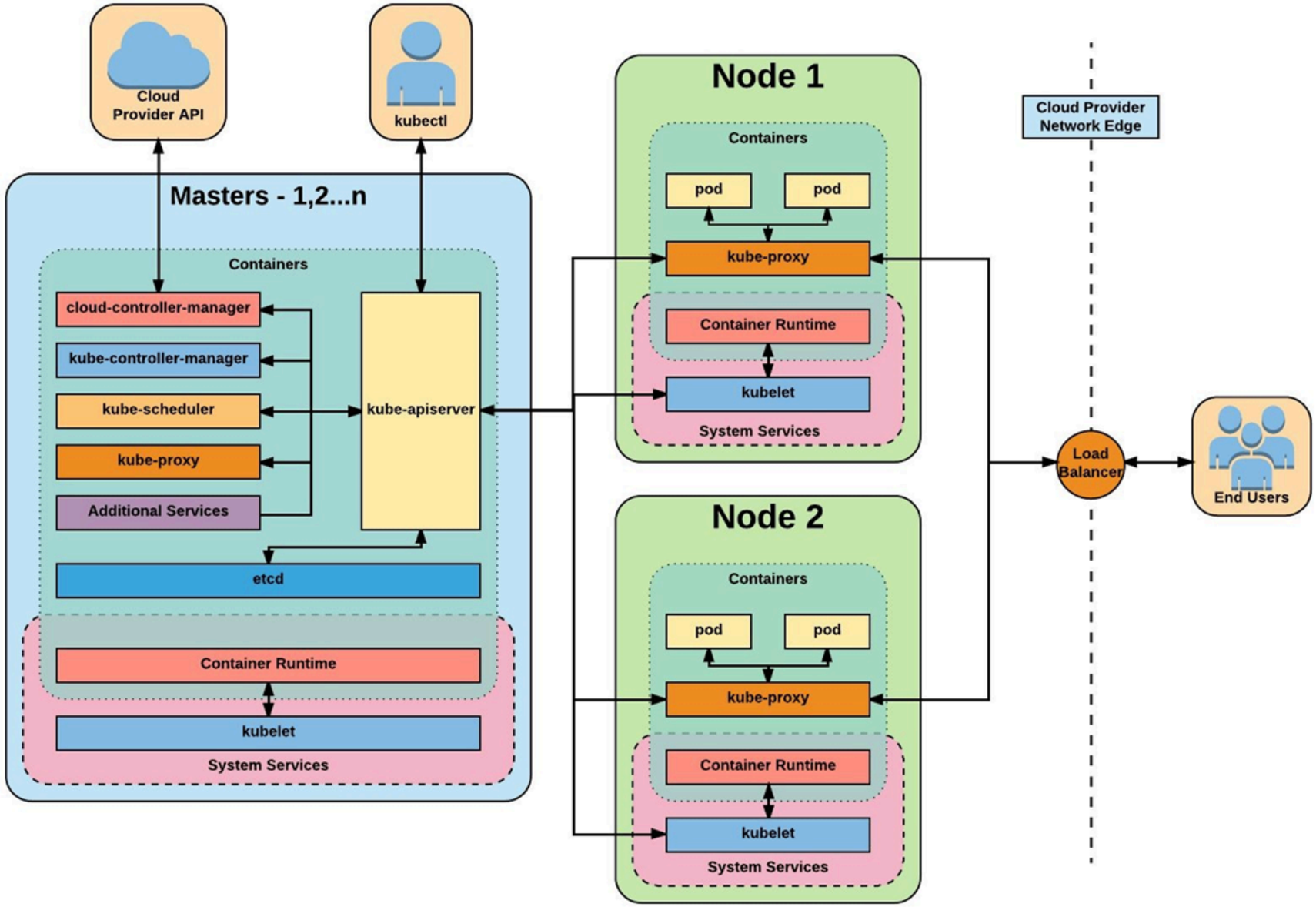
KUBERNETES II

KUBERNETES - ARCHITECTURE OVERVIEW

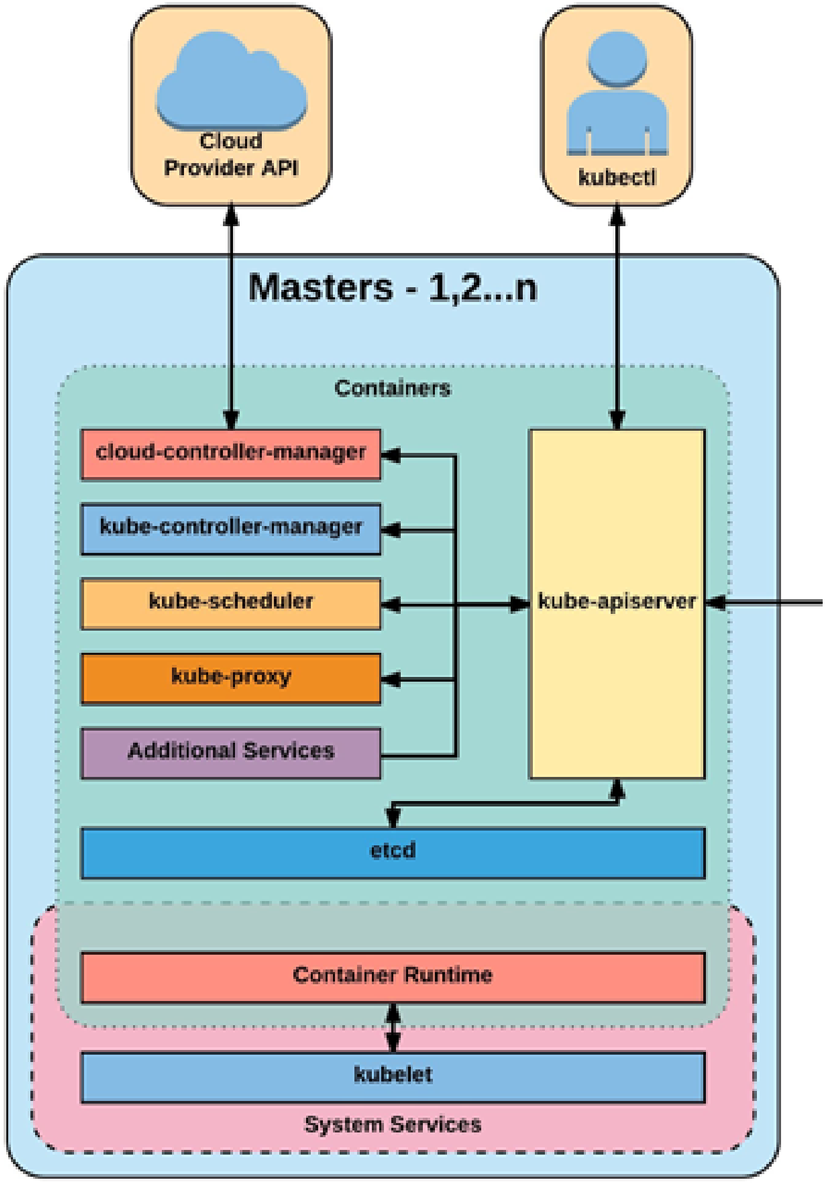
Master - Bertindak sebagai bidang kontrol utama untuk Kubernetes. Master bertanggung jawab minimal untuk menjalankan Server API, penjadwal, dan pengontrol cluster. Master biasanya juga mengelola penyimpanan status klaster, komponen khusus penyedia cloud, dan layanan penting klaster lainnya.

Node - Merupakan 'pekerja' dari sebuah klaster Kubernetes. Node menjalankan agen minimal yang mengelola node itu sendiri, dan ditugaskan untuk menjalankan beban kerja seperti yang ditentukan oleh master.

ARCHITECTURE OVERVIEW



MASTER COMPONENTS



●Kube-apiserver

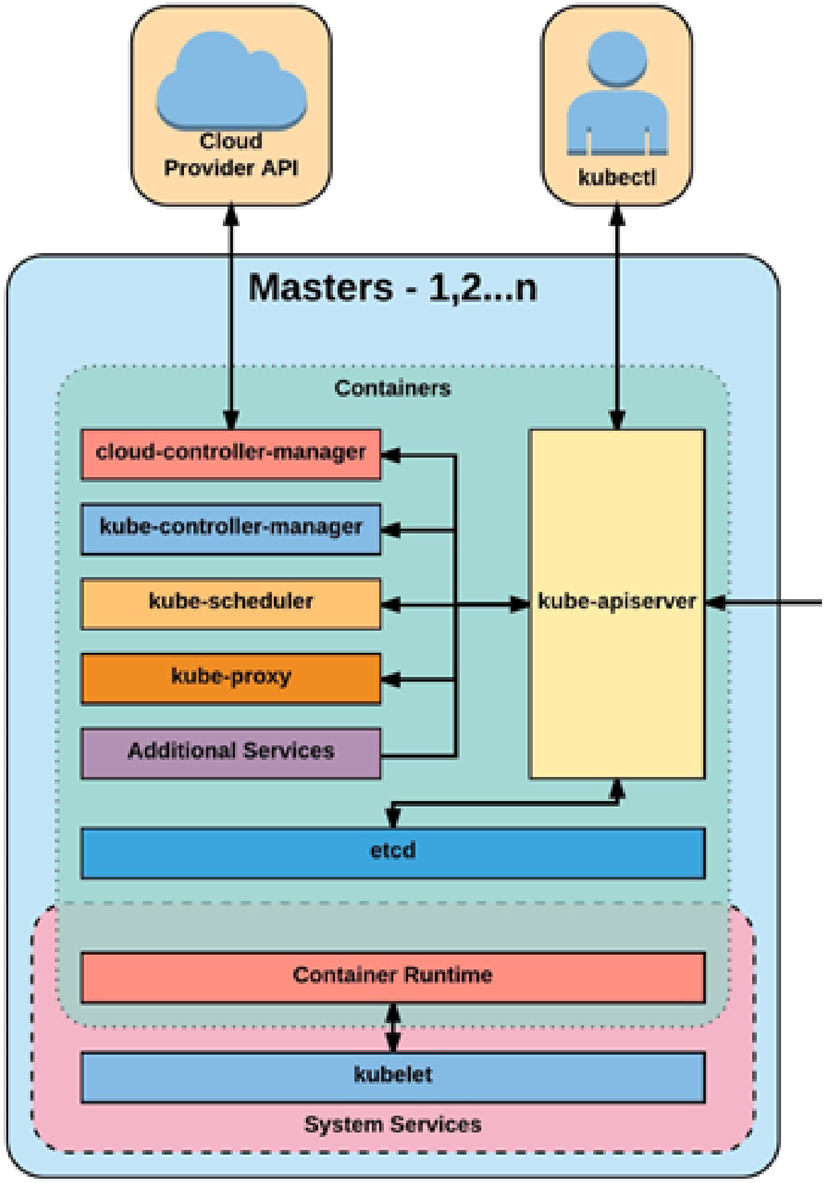
The apiserver provides a forward facing REST interface into the kubernetes control plane and datastore. All clients, including nodes, users and other applications interact with kubernetes strictly through the API Server.

It is the true core of Kubernetes acting as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

●Etcd

Etcd acts as the cluster datastore; providing a strong, consistent and highly available key-value store used for persisting cluster state.

MASTER COMPONENTS



●Kube-controller-manager

The controller-manager is the primary daemon that manages all core component control loops. It monitors the cluster state via the apiserver and steers the cluster towards the desired state.

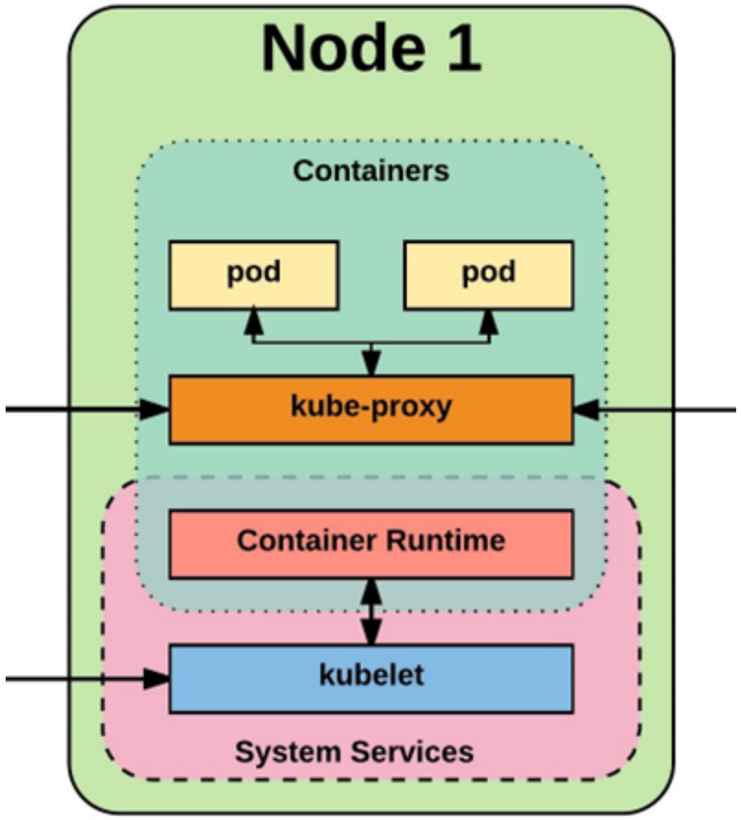
●Cloud-controller-manager

The cloud-controller-manager is a daemon that provides cloud-provider specific knowledge and integration capability into the core control loop of Kubernetes. The controllers include Node, Route, Service, and add an additional controller to handle PersistentVolumeLabels .

●Kube-scheduller

Kube-scheduler is a verbose policy-rich engine that evaluates workload requirements and attempts to place it on a matching resource. These requirements can include such things as general hardware reqs, affinity, anti-affinity, and other custom resource requirements.

NODE



COMPONENTS

●Kubelet

Acts as the node agent responsible for managing pod lifecycle on its host. Kubelet understands YAML container manifests that it can read from several sources:

- File path
- HTTP Endpoint
- Etcd watch acting on any changes
- HTTP Server mode accepting container manifests over a simple API.

●Container runtime

With respect to Kubernetes, A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.

- Containerd (docker)
- Cri-o
- Rkt
- Kata (formerly clear and hyper)
- Virtlet (VM CRI compatible runtime)

●Kube proxy

Manages the network rules on each node and performs connection forwarding or load balancing for Kubernetes cluster services.

Available Proxy Modes:

- Userspace
- iptables
- ipvs (alpha in 1.8)

NODE

~ Melihat Detail Node ~

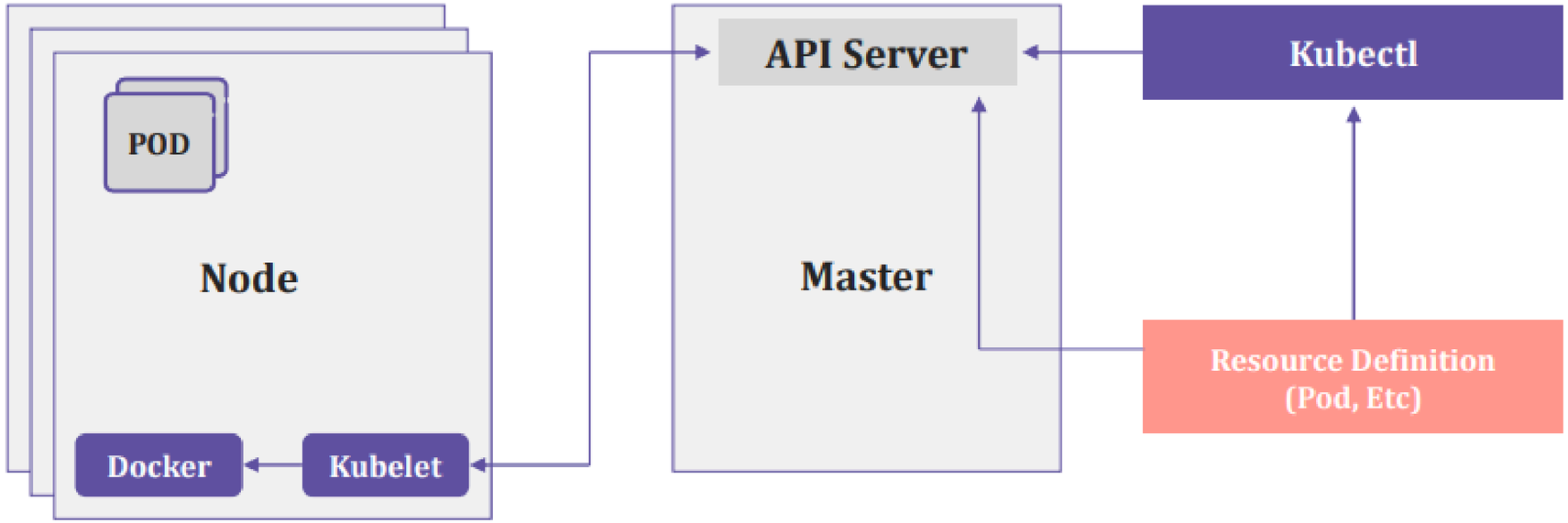
```
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason                       Message
  ----             ------  -----------------                 ------------------                ------                       -------
  MemoryPressure   False   Sat, 20 Apr 2024 13:45:32 +0700   Fri, 19 Apr 2024 20:49:00 +0700   KubeletHasSufficientMemory   kubelet
has sufficient memory available
  DiskPressure     False   Sat, 20 Apr 2024 13:45:32 +0700   Fri, 19 Apr 2024 20:49:00 +0700   KubeletHasNoDiskPressure     kubelet
has no disk pressure
  PIDPressure      False   Sat, 20 Apr 2024 13:45:32 +0700   Fri, 19 Apr 2024 20:49:00 +0700   KubeletHasSufficientPID      kubelet
has sufficient PID available
  Ready            True    Sat, 20 Apr 2024 13:45:32 +0700   Fri, 19 Apr 2024 20:49:07 +0700   KubeletReady                 kubelet
is posting ready status
Addresses:
  InternalIP:  192.168.65.3
  Hostname:    docker-desktop
Capacity:
  cpu:                4
  ephemeral-storage:  1055762868Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             3948188Ki
  pods:               110
Allocatable:
  cpu:                4
  ephemeral-storage:  972991057538
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             3845788Ki
  pods:               110
System Info:
  Machine ID:                 9c4dddf0-f42a-45e0-8c85-c8549382c61a
```

~ Melihat Semua Node ~
“kubectl get node”

```
PS D:\SIB6\belajar-kubernetes> kubectl get node
NAME             STATUS   ROLES           AGE   VERSION
docker-desktop   Ready    control-plane   16h   v1.29.1
PS D:\SIB6\belajar-kubernetes> kubectl describe node docker-desktop
Name:               docker-desktop
Roles:              control-plane
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=docker-desktop
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=
                    node.kubernetes.io/exclude-from-external-load-balancers=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: unix:///var/run/cri-dockerd.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Fri, 19 Apr 2024 20:49:07 +0700
Taints:             <none>
Unschedulable:      false
Lease:
  HolderIdentity:  docker-desktop
  AcquireTime:     <unset>
  RenewTime:       Sat, 20 Apr 2024 13:45:37 +0700
Conditions:
```

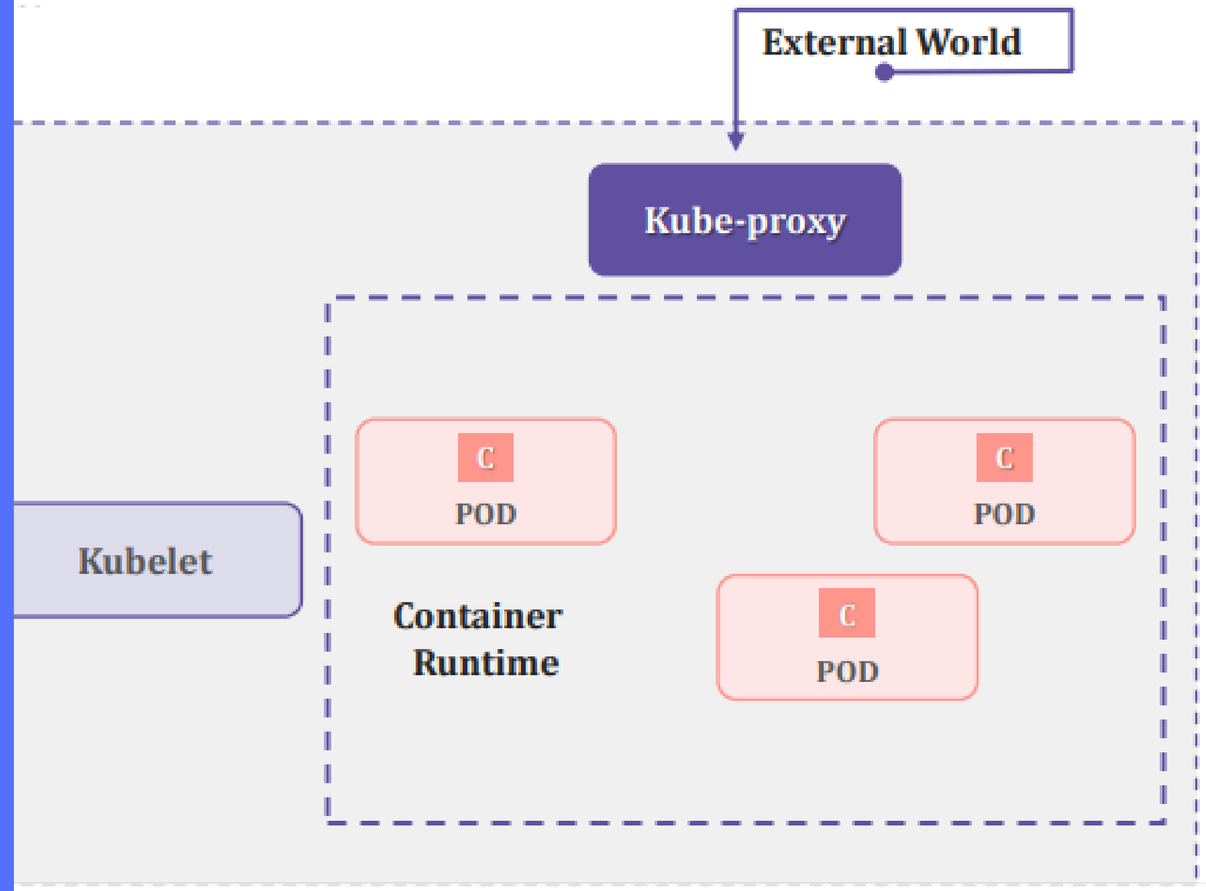
“kubectl describe node namanode”

POD



a Pod is the smallest deployable unit that represents a single instance of a running process. It encapsulates one or more containers, storage resources, and networking configurations that are tightly coupled and share the same lifecycle. Pods are the basic building blocks of Kubernetes applications.

POD

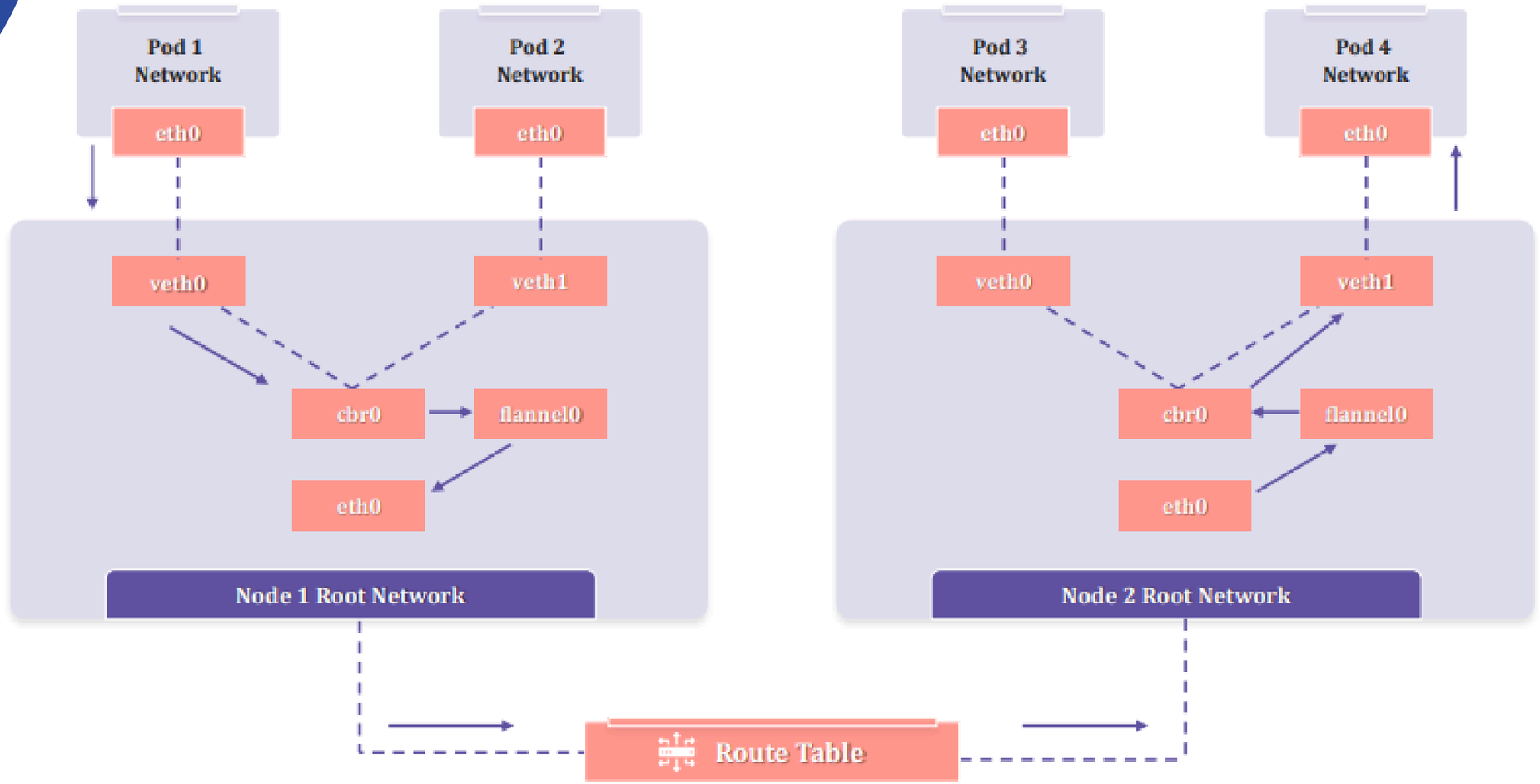


It is a physical server otherwise you will say a VM that runs the applications victimization Pods (a pod programing unit) that is controlled by the master node.

On a physical server (worker/slave node), pods area unit scheduled.

For accessing the applications from the external world, we have a tendency to connect with nodes.

POD



Ingress Networking in Kubernetes. The slide explains the Ingress Networking in Kubernetes and its working

POD

~ MEMBUAT POD ~

“kubectl create -f filepod.yaml”

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl create -f nginx.yaml
pod/nginx created
```

~ MELIHAT DETAIL POD ~

kubectl get pod
kubectl get pod -o wide
kubectl describe pod namapod

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             docker-desktop/192.168.65.3
Start Time:       Sat, 20 Apr 2024 13:52:14 +0700
Labels:           <none>
Annotations:      <none>
Status:           Pending
```

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod -o wide
NAME    READY   STATUS    RESTARTS   AGE     IP          NODE             NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          2m43s   10.1.0.10   docker-desktop   <none>           <none>
```

~ MELIHAT SEMUA POD ~

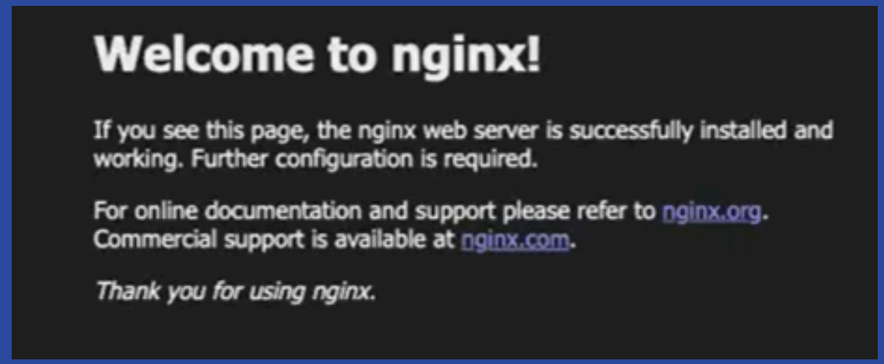
“kubectl get pod”

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          24s
```

~ MENGAKSES POD ~

kubectl port-forward namapod portAkses:portPod
kubectl port-forward namapod 8888:8080

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl port-forward nginx 8888:8080
Forwarding from 127.0.0.1:8888 -> 8080
Forwarding from [::1]:8888 -> 8080
Handling connection for 8888
Handling connection for 8888
```



LABEL

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pods --show-labels
NAME    READY   STATUS    RESTARTS   AGE     LABELS
nginx   1/1     Running   0          9m34s   <none>
PS D:\SIB6\belajar-kubernetes\examples> kubectl create -f nginx-with-label.yaml
pod/nginx-with-label created
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod
NAME               READY   STATUS    RESTARTS   AGE
nginx              1/1     Running   0          11m
nginx-with-label   1/1     Running   0          19s
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pods --show-labels
NAME               READY   STATUS    RESTARTS   AGE   LABELS
nginx              1/1     Running   0          11m   <none>
nginx-with-label   1/1     Running   0          43s   environment=production,team=finance,version=1.4.5
```

~ Menambahkan Label di Pod ~

- **kubectl create -f namafile.yaml**
- **kubectl get pods --show-labels**

Fungsi Label :

- Memberi tanda pada Pod
- Mengorganisir Pod
- Memberi informasi tambahan pada Pod
- Bisa digunakan pada semua resource di Kubernetes, seperti Replication Controller, Replica Set, Service, dan lain-lain.

LABEL

~ Menambah / Mengubah Label di Pod ~

- kubectl label pod namapod key=value
- kubectl label pod namapod key=value --overwrite

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl label pod nginx-with-label environment=dev --overwrite
pod/nginx-with-label labeled
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pods --show-labels
NAME                READY   STATUS    RESTARTS   AGE     LABELS
nginx               1/1     Running   0          13m     <none>
nginx-with-label    1/1     Running   0          3m18s   environment=dev,team=finance,version=1.4.5
PS D:\SIB6\belajar-kubernetes\examples> kubectl label pod nginx environment=dev
pod/nginx labeled
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pods --show-labels
NAME                READY   STATUS    RESTARTS   AGE     LABELS
nginx               1/1     Running   0          15m     environment=dev
nginx-with-label    1/1     Running   0          4m28s   environment=dev,team=finance,version=1.4.5
PS D:\SIB6\belajar-kubernetes\examples> kubectl label pod nginx environment=prod --overwrite
pod/nginx labeled
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pods -l environment=dev
NAME                READY   STATUS    RESTARTS   AGE
nginx-with-label    1/1     Running   0          7m41s
```

~ Mencari Pod dengan Label / Beberapa Label ~

- kubectl get pods -l key
- kubectl get pods -l key=value
- kubectl get pods -l '!key'
- kubectl get pods -l key!=value
- kubectl get pods -l 'key in (value1,value2)'
- kubectl get pods -l 'key notin (value1,value2)'

- kubectl get pods key,key2=value
- kubectl get pods key=value,key2=value

NAMESPACE

- Namespace digunakan ketika resources di Kubernetes sudah terlalu banyak, butuh memisahkan resources untuk multi-tenant, team atau environment, dan juga karena nama resources bisa sama jika berapa di namespace yang berbeda.

- Pod dengan nama yang sama boleh berjalan asalkan di Namespace yang berbeda. Namespace bukan mengisolasi resource, juga walaupun berbeda namespace, pod akan tetap bisa saling berkomunikasi dengan pod lain di namespace yang berbeda.

DigitalSkola

NAMESPACE

~ MELIHAT POD DI NAMESPACE ~

- kubectl get pod --namespace namanamespace
- kubectl get pod -n namanamespace

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod --namespace default
NAME              READY   STATUS    RESTARTS   AGE
nginx             1/1     Running   0          22m
nginx-with-label  1/1     Running   0          11m
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod --namespace kube-system
NAME                                     READY   STATUS    RESTARTS        AGE
coredns-76f75df574-xnzrv                 1/1     Running   1 (5h10m ago)   17h
coredns-76f75df574-zcgk2                 1/1     Running   1 (5h10m ago)   17h
etcd-docker-desktop                      1/1     Running   1 (5h10m ago)   17h
kube-apiserver-docker-desktop            1/1     Running   1 (5h10m ago)   17h
kube-controller-manager-docker-desktop   1/1     Running   2 (5h10m ago)   17h
kube-proxy-9df6g                         1/1     Running   1 (5h10m ago)   17h
kube-scheduler-docker-desktop            1/1     Running   1 (5h10m ago)   17h
storage-provisioner                      1/1     Running   2 (5h10m ago)   17h
vpnkit-controller                        1/1     Running   1 (5h10m ago)   17h
```

~ MEMBUAT NAMESPACE ~

- kubectl create -f namafile.yaml
- kubectl create -f namafile.yaml --namespace namanamespace

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl create -f finance-namespace.yaml
namespace/finance created
PS D:\SIB6\belajar-kubernetes\examples> kubectl get ns
NAME              STATUS   AGE
default           Active   17h
finance           Active   15s
kube-node-lease   Active   17h
kube-public       Active   17h
kube-system       Active   17h
PS D:\SIB6\belajar-kubernetes\examples> kubectl create -f nginx-with-label.yaml -n finance
pod/nginx-with-label created
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod --namespace finance
NAME              READY   STATUS    RESTARTS   AGE
nginx-with-label  1/1     Running   0          22s
```

~ MENGHAPUS NAMESPACE & POD MENGGUNAKAN LABEL ~

- kubectl delete namespace namanamespace
- kubectl delete pod namapod

```
PS D:\SIB6\belajar-kubernetes\examples> kubectl delete pod nginx-with-label -n finance
pod "nginx-with-label" deleted
PS D:\SIB6\belajar-kubernetes\examples> kubectl get pod --namespace finance
No resources found in finance namespace.
PS D:\SIB6\belajar-kubernetes\examples> kubectl delete namespace finance
namespace "finance" deleted
PS D:\SIB6\belajar-kubernetes\examples> kubectl get ns
NAME              STATUS   AGE
default           Active   17h
kube-node-lease   Active   17h
kube-public       Active   17h
kube-system       Active   17h
```

- kubectl delete pod -l key=value
- kubectl delete pod --all --namespace namanamespace

TERIMA KASIH

DigitalSkola

Uncover The World of Digital Skills with Us