



**LOAD
BALANCING**

MODUL PEMBELAJARAN
TEKNOLOGI KOMPUTASI AWAN
LOAD BALANCING

Modul 5 Load Balancing

Dosen pengampu :

Henning Titi Ciptaningtyas, S.Kom, M.Kom.

Disusun oleh :

Abadila Barasmara Bias Dewandra

Muhammad Hanif Fatihurrizqi

Naufal Dhiya Ulhaq

Sharira Saniane

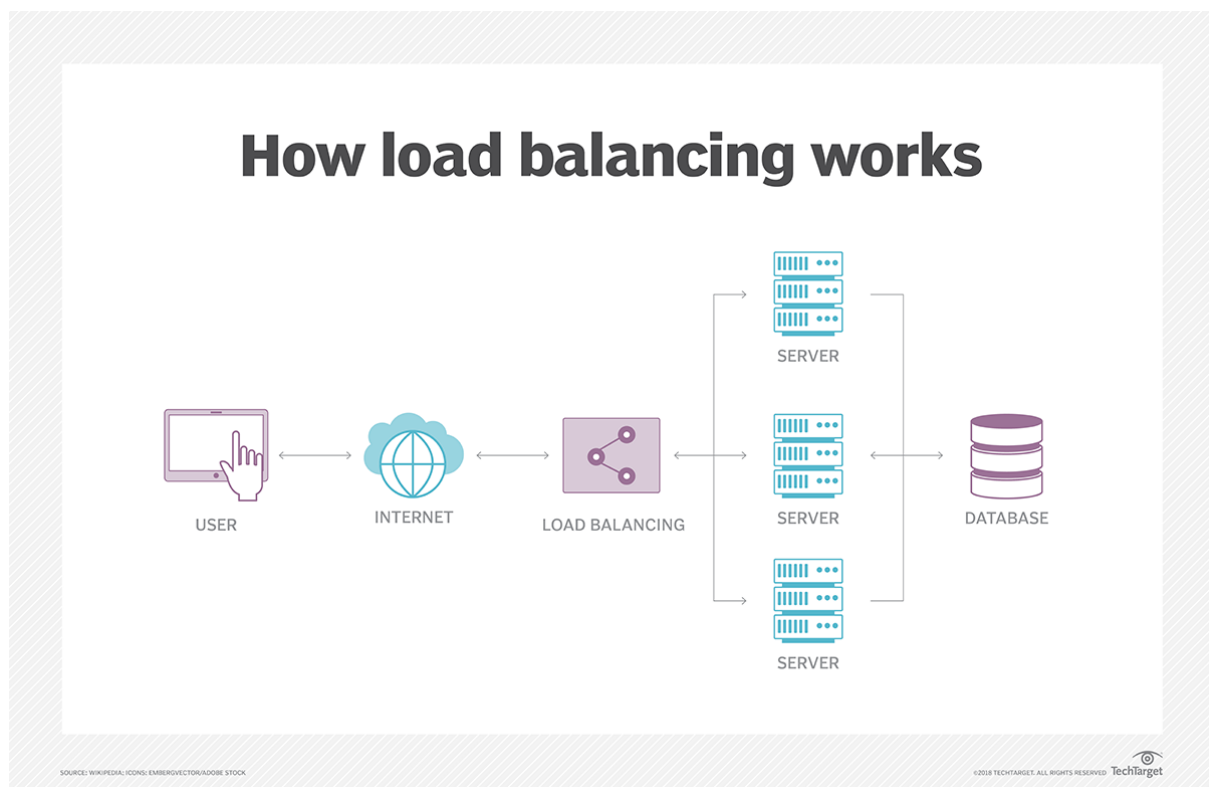
DAFTAR ISI

Load Balancing	3
Load Balancer Algorithm	4
NGINX	5
Setup Load Balancer	5
Menginstall Library	5
Menyiapkan Struktur Folder/File	6
Konfigurasi main.py	6
Konfigurasi dockerfile	9
Konfigurasi requirements.txt	9
Konfigurasi nginx	10
Konfigurasi docker-compose	10
app, app2, app3	12
nginx	12
mongodb	12
mongo-express	13
Done 🎉🎉🎉	13
Task Scheduling	15
Cloudsim	15
Komponen	15
Task Scheduling Algorithm	16
Instalasi	16

Load Balancing

Cloud Load balancing adalah sebuah proses mendistribusikan beban kerja (workload) kepada seluruh sumber daya di lingkungan komputasi awan tertentu dengan konsiderasi jaringan lalu lintas.

Cara kerjanya, load balancer mendapat workload dan me-routing ke sumber daya sehingga workload yang lain diurus terus-menerus tanpa ada waiting yang panjang. Tidak hanya dalam beberapa server, load balancing bisa melakukan distribusi secara geographical.



Contoh Penyedia Layanan Cloud Load Balancing:

- Amazon Web Services (AWS) menggunakan Elastic Load Balancing (ELB).
- Google Cloud Platform (GCP) menggunakan Google Cloud Load Balancing.
- Microsoft Azure menggunakan Azure Load Balancer.

Load Balancer Algorithm

Dibagi menjadi 2

1. Static
 - a. Round robin
Pembagian menggunakan DNS dalam bentuk rotasi.
 - b. Weighted round robin

Pembagian berdasarkan beban yang ditentukan. Jika bisa handle traffic besar, maka weight nya makin besar.

- c. IP hash
Menggunakan fungsi matematika untuk mengubah IP Address ke hash. Berdasarkan hash tersebut, koneksi dihubungkan pada server tersebut.
2. Dynamic
 - a. Least connection
Melihat server yang memiliki koneksi yang sedikit (Asumsi seluruh kekuatan proses sama).
 - b. Weighted response time
Melihat rata-rata waktu respons tiap server, tetapi juga melihat jumlah koneksi pada server.
 - c. Resource-based
Melihat sumber daya (eg. CPU) pada server. Memerlukan 'agent' yang dapat memonitor sumber daya server.
 - d. Dan masih banyak lagi!

NGINX

Merupakan program open source yang digunakan untuk web serving, reverse proxy, caching, load balancing, streaming, dll.

Setup Load Balancer

Menginstall Library

Untuk host, akan menggunakan Docker dan Docker Compose dengan FastAPI dan MongoDB sebagai Database.

Jadi, mari menginstall ketiga library tersebut dengan:

1. Docker dan Docker Compose
Sudah dijelaskan di modul sebelumnya ya 😊
2. FastAPI

```
pip install fastapi
```

3. Kemudian menginstall uvicorn dengan

```
pip install uvicorn
```

4. MongoDB
Untuk library ini, hanya akan menginstall pydantic untuk keperluan input database dan pymongo untuk mengkoneksikan FastAPI dengan database.

```
pip install pymango pydantic
```

Menyiapkan Struktur Folder/File

Silakan mengikuti struktur berikut

```
.
├── Awan/
│   ├── app/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirement.txt
│   ├── app2/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirement.txt
│   ├── app3/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirement.txt
│   ├── docker-compose.yml
│   └── locustfile.py
```

Konfigurasi main.py

Mengimport library

```
from fastapi import FastAPI, Body, Request # FastAPI utama
from fastapi.encoders import jsonable_encoder # JSON Handler
import pymongo # Menghubungkan ke Database MongoDB
from pydantic import BaseModel #
from bson.objectid import ObjectId # Identifikasi ID pada mongoDB
import uuid # Default ID Generator
import socket # Identifier tiap docker
import time # Simulasi waktu request
```

selanjutnya, mari kita menghubungkan python dengan database mongoDB.

```
MONGO_DETAILS = "mongodb://admin:admin@mongodb:27017/" # nama aplikasi
docker mongodb
client = pymongo.MongoClient(MONGO_DETAILS)
db = client['tes'] # nama databesnya 'tes'
```

```
collection = db['tes'] # nama collectionnya 'tes'
```

untuk nama database dan collection bisa disesuaikan.

Selanjutnya, mendefinisikan jenis data di dalam collection:

```
class Item(BaseModel):  
    name: str  
    age: int  
    rank: str
```

collection mongoDB akan menggunakan struktur di atas.

Kemudian, mari mendefinisikan fungsi-fungsi untuk me-return data:

```
def myData(data):  
    return {  
        "id": str(data["_id"]),  
        "name": data["name"],  
        "age": data["age"],  
        "rank": data["rank"],  
    }  
  
def myFullData(datas):  
    return [myData(data) for data in datas]  
  
def ResponseModel(data, message = "Success"):  
    return {  
        "data": [data],  
        "code": 200,  
        "message": message,  
    }  
  
def ErrorResponseModel(error, code, message):  
    return {  
        "error": error,  
        "code": code,  
        "message": message  
    }
```

Penjelasannya:

- myData untuk me-return data dengan format menyesuaikan struktur data atau lainnya.

- myFullData untuk mengembalikan seluruh isi data pada collection.
- ResponseModel sebagai handler data jika berhasil dilakukan.
- ErrorResponseModel sebagai handler data jika gagal dilakukan.

Setelah seluruh handler telah dibuat, mari kita ke main course, yaitu pembuatan REST API dengan FastAPI.

Deklarasikan aplikasi

```
app = FastAPI()
```

Kita akan menggunakan 6 endpoint untuk mensimulasikan aplikasi REST API sederhana.

- GET / sebagai testing.
- GET /fast sebagai testing dengan response cepat.
- GET /slow sebagai testing dengan response lambat.
- GET /all mendapatkan semua data pada collection.
- POST /create memasukkan data baru pada collection.
- GET /get/{id} mendapatkan data sesuai id.

berikut implementasinya:

```
@app.get('/')
async def home():
    return {
        "message": "This is server A",
        "hostname": socket.gethostname()}

@app.get('/fast')
async def hello():
    time.sleep(0.5)
    return {
        "message": "Hello world from server A",
        "opt": "fast"}

@app.get('/slow')
async def hello():
    time.sleep(1)
    return {
        "message": "Hello world from server A",
        "opt": "slow"}

@app.get("/all")
async def get_all_data():
    # print(client.tes.tes.find())
```



```
# print(myFullData(client.tes.tes.find()))
return ResponseModel(myFullData(collection.find()), "All Good")

# insert new data
@app.post("/create", response_model=Item)
async def create_data(request: Request, lister: Item = Body(...,
embed=True)):
    lister = jsonable_encoder(lister)
    new_data = client.tes.tes.insert_one(lister)
    # return response with data inserted
    return lister

# get data by id
@app.get("/get/{id}")
async def get_data(id: str):
    Objinstance = ObjectId(id)
    data = collection.find_one({"_id": Objinstance})
    if data:
        return ResponseModel(myData(data), "A")
    else:
        return ErrorResponseModel("An error occurred.", 404, "Data
doesn't exist.")
```

Konfigurasi dockerfile

Adapun isi docker diisi berikut:

```
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Docker akan membuat folder app dan akan menginstall seluruh requirement di dalam file requirement.txt

Saat docker diluncurkan, DOcker python akan menjalankan uvicorn pada port 8000

uvicorn akan menjalankan file main dengan aplikasi app (akan dijelaskan pada pembuatan FastAPI).

Konfigurasi requirements.txt

silakan memasukkan library-library yang akan digunakan:

```
fastapi==0.78.0
uvicorn==0.18.2
pymongo
pydantic
uuid
```

Konfigurasi nginx

Pada nginx, kita akan mengkonfigurasi Dockerfile dan file konfigurasi.

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d/default.conf
```

dengan file konfigurasi nginx.conf sebagai berikut:

```
upstream app {
    server app:8000;
    server app2:8000;
    server app3:8000;
}

server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://app;
    }
}
```

mengingat seluruh app akan dijalankan di docker port 8000 dan nginx akan dijalankan pada host port 80

proxy_pass http://app nama app menyesuaikan nama uvicorn yang dijalankan (main:app)

Dari upstream yang kita masukkan, load balancer yang akan digunakan adalah Round-Robin.

Konfigurasi docker-compose

Pada docker compose, ada beberapa docker image yang akan digunakan, berupa:

- nginx

- mongo dan mongo-express

Buatlah docker-compose dengan isi seperti di bawah ini:

```
version: '3'

services:
  app:
    build: ./app
    ports:
      - "8001:8000"
    depends_on:
      - mongodb
  app2:
    build: ./app2
    ports:
      - "8002:8000"
    depends_on:
      - mongodb
  app3:
    build: ./app3
    ports:
      - "8003:8000"
    depends_on:
      - mongodb
  nginx:
    build: ./nginx
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - app
      - app2
      - app3
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=admin
    volumes:
      - mongodb_data:/data/db
  mongo-express:
    image: mongo-express
    ports:
```

```
- "8082:8081"
environment:
  - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
  - ME_CONFIG_MONGODB_ADMINPASSWORD=admin
  - ME_CONFIG_MONGODB_SERVER=mongodb
  - ME_CONFIG_MONGODB_ENABLE_ADMIN=true
  - ME_CONFIG_BASICAUTH_USERNAME=admin
  - ME_CONFIG_BASICAUTH_PASSWORD=admin
depends_on:
  - mongodb
volumes:
  mongodb_data:
    driver: local
```

app, app2, app3

pada service app, app2, app3, pastikan arah file sudah menuju folder yang memiliki Dockerfile yang telah dibuat sebelumnya.

adapun host ports yang akan digunakan tiap service adalah 8001 8002 8003 dan docker port seluruhnya diarahkan ke 8000

Why? karena kita menjalankan uvicorn tiap app di port 8000 namun tiap docker harus dijalankan di port host yang berbeda.

Karena aplikasi akan menghubungkan database, dan database perlu di load terlebih dahulu, maka tambahkan mongodb

nginx

build diarahkan ke folder berisi Dockerfile nginx.

port yang akan digunakan minimal 80:80.

nginx akan bergantung penggunaannya dengan app yang ada, sehingga ditambahkan depends_on ke tiap app.

mongodb

Karena kita tidak menkonfigurasi Dockerfile, kita akan load image dari repository Docker bernama mongo.

Port mongo dijalankan di 27017:27017 (default).

WAJIB memasukkan MONGO_INITDB_ROOT_USERNAME dan MONGO_INITDB_ROOT_PASSWORD untuk setup awal database.

Agar isi database tidak berubah setiap docker-composer di up, deklarasikan volumes dan mengisi sesuai isi di atas.

di luar struktur services, jangan lupa deklarasikan:

```
volumes:  
  mongodb_data:  
    driver: local
```

mongo-express

Media penampilan database mongodb pada webUI.

Deklarasikan image mongo-express karena tidak ada konfigurasi tambahan.

Defaut port mongo-express adalah 8081 Jalankan pada ports 8082:8081.

Adapun enviroentment yang perlu disetup:

- `ME_CONFIG_MONGODB_ADMINUSERNAME=admin` username ke mongodb
- `ME_CONFIG_MONGODB_ADMINPASSWORD=admin` password ke mongodb
- `ME_CONFIG_MONGODB_SERVER=mongodb` nama server mongodb yang digunakan (sesuai nama services)
- `ME_CONFIG_MONGODB_ENABLE_ADMIN=true` menghidupkan admin
- `ME_CONFIG_BASICAUTH_USERNAME=admin` username untuk mengakses webUI
- `ME_CONFIG_BASICAUTH_PASSWORD=admin` password untuk mengakses webUI

karena mongo-express harus menunggu mongodb selesai load, maka harus menambahkan mongodb pada `depends_on`.

Done 🎉🎉🎉

Apabila sudah sesuai, jalankan docker compose dengan

```
docker-compose up --build
```

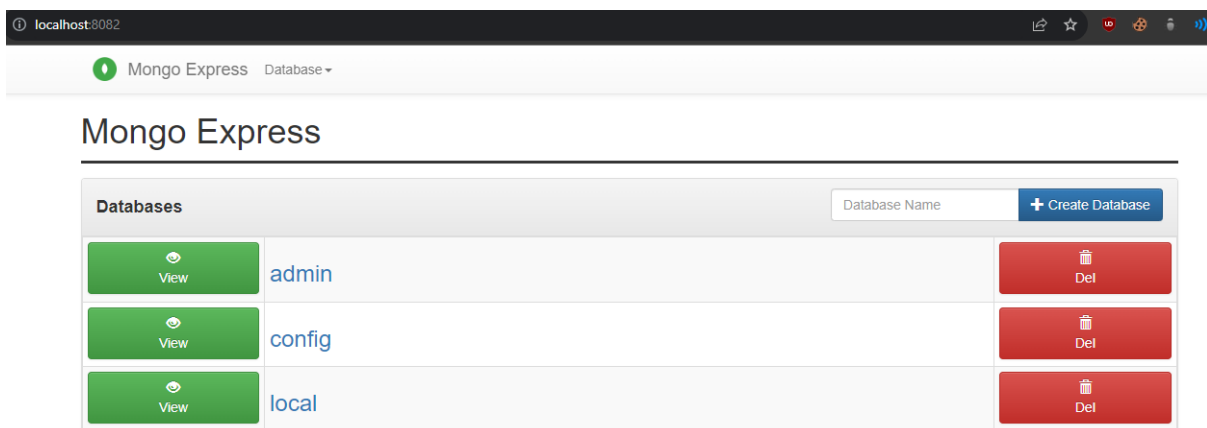
Kemudian buka localhost dan coba refresh beberapa kali

```
localhost  
{"message":"This is server A","hostname":"01b5514c7374"}
```

```
localhost  
{"message":"This is server B","hostname":"77cb19c61e36"}
```

```
localhost  
{"message":"This is server C","hostname":"1f4f3764bfd9"}
```

untuk setup mongodb, maka buka localhost:8082, kemudian masukkan username dan password yang sudah ditentukan sebelumnya.



Langkah selanjutnya:

masukkan nama database baru > Tekan create database > Buka database > masukkan nama koleksi baru > tekan Create collection

Pada halaman koleksi, mari buat dokumen baru dengan struktur seperti berikut:

```
{  
  "_id": ObjectId(),  
  "name": "Lorem ipsum",  
  "age": 20,  
  "rank": "admin"  
}
```

Jangan lupa mengganti nama database dan collection di main.py, kemudian docker-compose up ulang

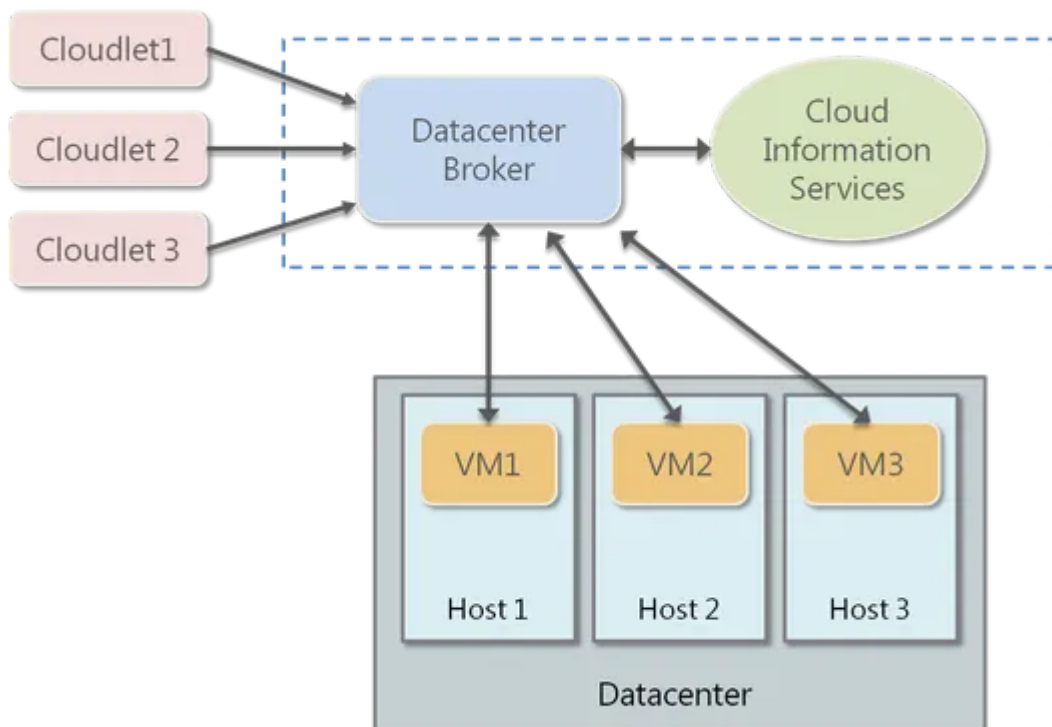
```
localhost/all
{"data": [{"id": "656203fc96bebbe58c77e6ea", "name": "Lorem ipsum", "age": 20, "rank": "admin"}], "code": 200, "message": "All Good"}
```

Task Scheduling

Merupakan proses mengelola eksekusi tugas di cloud. "Tugas" disini merupakan komputasi seperti pemrosesan data, analisis, komputasi paralel, dan lain-lain.

Cloudsim

CloudSim adalah kerangka kerja sumber terbuka, yang digunakan untuk mensimulasikan infrastruktur dan layanan komputasi awan. Ini dikembangkan oleh organisasi CLOUDS Lab dan ditulis sepenuhnya dalam bahasa Java. Ini digunakan untuk memodelkan dan mensimulasikan lingkungan komputasi awan sebagai sarana untuk mengevaluasi hipotesis sebelum pengembangan perangkat lunak untuk mereproduksi pengujian dan hasil.



Komponen

1. Datacenter
digunakan untuk memodelkan peralatan perangkat keras dasar dari setiap lingkungan cloud, yaitu Pusat Data. Kelas ini menyediakan metode untuk menentukan persyaratan fungsional Pusat Data serta metode untuk mengatur kebijakan alokasi VM, dll.
2. Broker

adalah entitas yang bertindak atas nama pengguna/pelanggan. Ini bertanggung jawab atas fungsi VM, termasuk pembuatan, pengelolaan, penghancuran, dan pengiriman cloudlet ke VM.

3. Cloudlet

kelas cloudlet mewakili tugas apa pun yang dijalankan pada VM, seperti tugas pemrosesan, atau tugas akses memori, atau tugas pembaruan file, dll. Kelas ini menyimpan parameter yang mendefinisikan karakteristik tugas seperti panjang, ukuran, mi (juta instruksi) dan menyediakan metode yang serupa dengan kelas VM, serta menyediakan metode yang mendefinisikan waktu eksekusi, status, biaya, dan riwayat tugas.

4. VM

kelas ini merepresentasikan mesin virtual dengan menyediakan anggota data yang mendefinisikan bandwidth, RAM, mips (juta instruksi per detik), ukuran VM, dan juga menyediakan metode pengatur dan pengambil untuk parameter-parameter ini.

Task Scheduling Algorithm

Beberapa algoritma yang akan dibahas pada modul ini:

1. Round Robin

Task diberi secara 'melingkar'. Bisa menangani banyak pekerjaan, tetapi throughput bisa kecil dan waktu tunggu yang lama pada beberapa task.

2. Shortest Job First (SJF)

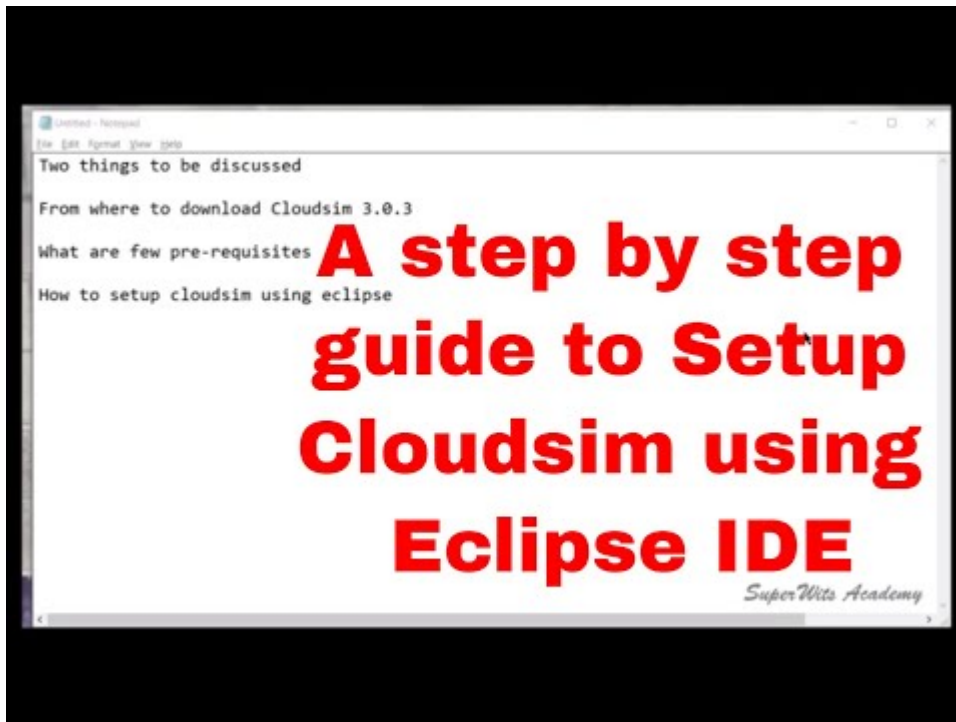
Melihat jumlah waktu proses terkecil. Waktu turnaround cepat, tetapi memerlukan estimasi waktu proses yang akurat agar menjadi efektif.

Instalasi

Beberapa komponen yang perlu diinstall:

1. [Eclipse IDE](#)
2. [cloudsim-3.0.3](#)
3. [common-math 3.6.1](#)

Video Tutorial menyiapkan tools:



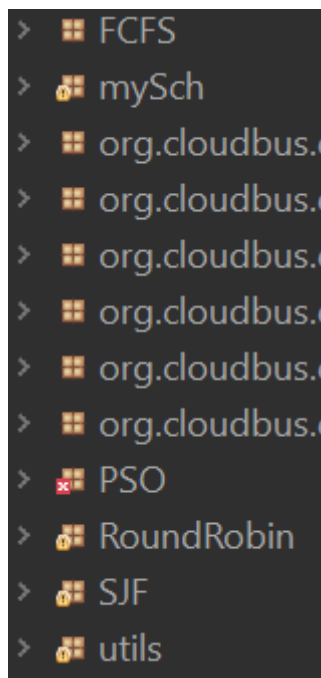
Atau tanya asisten Imao

Setelah selesai disiapkan, download repository berikut:

<https://github.com/michaelfahmy/cloudsim-task-scheduling/tree/master>

Masukkan isi folder src ke dalam folder example.

Lalu refresh



Jalankan RoundRobin/RoundRobinScheduler.java

Klik Kanan > Run As > Java Application

```
===== OUTPUT =====
Cloudlet ID   STATUS   Data center ID  VM ID   Time      Start Time   Finish Time
02           SUCCESS   04              04      936.48    00.1         936.58
01           SUCCESS   03              03      1897.32   00.1         1897.42
13           SUCCESS   05              05      1975.23   00.1         1975.33
04           SUCCESS   02              02      3960.94   00.1         3961.04
00           SUCCESS   06              06      4338.18   00.1         4338.28
03           SUCCESS   04              04      3497.03   936.58      4433.62
08           SUCCESS   03              03      2960.12   1897.42    4857.54
14           SUCCESS   05              05      3152.86   1975.33    5128.2
05           SUCCESS   02              02      1324.97   3961.04    5286.01
09           SUCCESS   03              03      1160.89   4857.54    6018.42
07           SUCCESS   06              06      2765.53   4338.28    7103.8
19           SUCCESS   05              05      2656.8    5128.2     7785
06           SUCCESS   02              02      2843.98   5286.01    8130
25           SUCCESS   03              03      2113.95   6018.42    8132.38
11           SUCCESS   04              04      4413.7    4433.62    8847.31
12           SUCCESS   06              06      2621.23   7103.8     9725.03
10           SUCCESS   02              02      1933.34   8130       10063.34
15           SUCCESS   02              02      646.76    10063.34   10710.1
26           SUCCESS   03              03      2781.52   8132.38    10913.9
17           SUCCESS   04              04      2484.29   8847.31    11331.6
28           SUCCESS   02              02      1369.2    10710.1    12079.3
16           SUCCESS   06              06      2360.29   9725.03    12085.32
18           SUCCESS   04              04      909.94    11331.6    12241.54
29           SUCCESS   02              02      1451.02   12079.3    13530.31
22           SUCCESS   04              04      2113.06   12241.54   14354.6
20           SUCCESS   06              06      2385.22   12085.32   14470.54
21           SUCCESS   06              06      1774.67   14470.54   16245.22
24           SUCCESS   06              06      181.38    16245.22   16426.6
23           SUCCESS   04              04      2255.32   14354.6    16609.92
27           SUCCESS   06              06      1064.74   16426.6    17491.34
Makespan using RR: 4629.656014755759
RoundRobin.RoundRobinScheduler finished!
```