

TRANSFER OF TECHNOLOGY

cybersolution technologies SDN BHD

## **DOCKER**

PENGENALAN KEPADA DOCKER

# 1. Apakah Itu Docker?

Docker ialah platform sumber terbuka yang digunakan untuk membangunkan, menghantar, dan menjalankan aplikasi dalam persekitaran yang dikenali sebagai container. Ia membolehkan pembangun untuk membungkus aplikasi bersama semua kebergantungannya supaya ia boleh berjalan secara konsisten di mana-mana sahaja – dari komputer pembangunan ke pelayan pengeluaran.

# 2. Konsep Asas Docker

|  |  |
| --- | --- |
| **Istilah** | **Maksud** |
| **Image** | Imej ialah templat baca-sahaja (read-only) yang mengandungi semua keperluan untuk menjalankan sesuatu aplikasi seperti kod, pustaka dan konfigurasi. |
| **Container** | Kontena ialah satu salinan aktif (instans) daripada imej yang sedang dijalankan. Ia menyediakan persekitaran yang terasing untuk aplikasi. |
| **Volume** | Volum ialah storan kekal yang digunakan oleh kontena untuk menyimpan data walaupun selepas kontena dimatikan atau dipadam. |
| **Dockerfile** | Dockerfile ialah fail skrip yang mengandungi arahan untuk membina imej Docker. Ia menentukan langkah-langkah pemasangan persekitaran aplikasi. |
| **Docker Hub** | Docker Hub ialah repositori awan untuk memuat naik, menyimpan dan berkongsi imej Docker. Ia seperti "GitHub" untuk imej. |
| **Docker Compose** | Docker Compose ialah alat untuk mentakrif dan mengurus aplikasi yang menggunakan lebih dari satu kontena, dengan fail konfigurasi docker-compose.yml. |

# 3. Kelebihan Docker

|  |  |  |
| --- | --- | --- |
| **Kelebihan Docker** | **Penjelasan** | **Contoh Penggunaan** |
| **Konsistensi Persekitaran** | Docker memastikan aplikasi berjalan sama di semua tempat (laptop, server, cloud) kerana persekitaran terasing. | Seorang pembangun membina aplikasi web menggunakan Docker, dan aplikasi itu berjalan sama di semua server. |
| **Pengurusan Pergantungan (Dependencies)** | Semua keperluan aplikasi (pustaka, versi perisian) dimasukkan dalam imej Docker. | Aplikasi Python yang perlukan versi tertentu numpy tidak terganggu oleh sistem utama. |
| **Mudah untuk Ujian dan Deploy (CI/CD)** | Docker memudahkan automasi ujian dan penghantaran (deployment) melalui pipeline CI/CD. | GitLab CI/CD digunakan untuk uji dan deploy aplikasi ke production menggunakan Docker container. |
| **Skalabiliti dan Microservices** | Menyokong seni bina microservices – setiap servis dalam kontena berasingan, mudah diurus dan diskalakan. | Satu sistem e-dagang mempunyai servis pembayaran, inventori, dan pengguna — setiap satu dalam kontena berasingan. |
| **Penggunaan Sumber Yang Efisien** | Lebih ringan dan pantas berbanding VM (mesin maya) kerana kongsi kernel OS. | Server yang kecil boleh jalankan banyak aplikasi serentak menggunakan Docker, tanpa perlukan VM besar. |
| **Portable dan Mudah Diedarkan** | Imej Docker boleh dihantar ke sesiapa atau mana-mana server/cloud tanpa ubah suai konfigurasi. | Pembangun hantar fail imej Docker kepada rakan sekerja, dan ia terus boleh dijalankan. |
| **Pemulihan & Isolasi Aplikasi** | Masalah dalam satu kontena tidak jejaskan sistem utama atau kontena lain. | Jika servis MySQL dalam satu kontena rosak, servis lain seperti backend tetap berfungsi. |

# 4. Perbandingan Docker vs Virtual Machine (VM)

|  |  |  |
| --- | --- | --- |
| **Aspek** | **Docker** | **Virtual Machine (VM)** |
| **Definisi** | Platform kontena ringan yang berkongsi kernel dengan host OS. | Mesin maya lengkap dengan sistem operasi tersendiri yang berjalan di atas hypervisor. |
| **Isolasi** | Terasing antara kontena, tetapi berkongsi kernel host. | Sangat terasing – setiap VM ada OS sendiri dan tidak berkongsi kernel. |
| **Kelajuan Boot** | Sangat pantas (dalam beberapa saat). | Perlahan – perlu masa untuk boot keseluruhan sistem operasi (beberapa minit). |
| **Penggunaan Sumber** | Ringan – hanya jalankan aplikasi dan dependensi. | Berat – setiap VM perlukan RAM, CPU dan storage untuk OS penuh. |
| **Saiz Fail (Image)** | Kecil (dalam ratusan MB). | Besar (beberapa GB). |
| **Deploy & Scalability** | Sangat mudah dan pantas untuk deploy dan skalakan. | Lebih kompleks dan perlahan untuk deploy dan klonkan. |
| **Contoh Penggunaan** | Jalankan aplikasi web, microservices, CI/CD pipeline dalam kontena. | Jalankan sistem operasi Ubuntu dalam Windows menggunakan VirtualBox atau VMware. |
| **Keselamatan (Security)** | Baik, tetapi kongsi kernel – perlu konfigurasi tambahan untuk keselamatan. | Sangat baik – isolasi penuh, serangan dalam VM sukar menjejaskan host. |
| **Backup & Recovery** | Imej mudah dieksport/import dan disimpan dalam Docker Hub. | Snapshot VM boleh diambil, tetapi saiz besar dan ambil masa lebih lama. |
| **Kesesuaian** | Sesuai untuk pembangunan moden, DevOps, microservices. | Sesuai untuk simulasi sistem operasi, legacy apps, atau virtualisasi penuh. |

# 5. Arahan Asas Docker

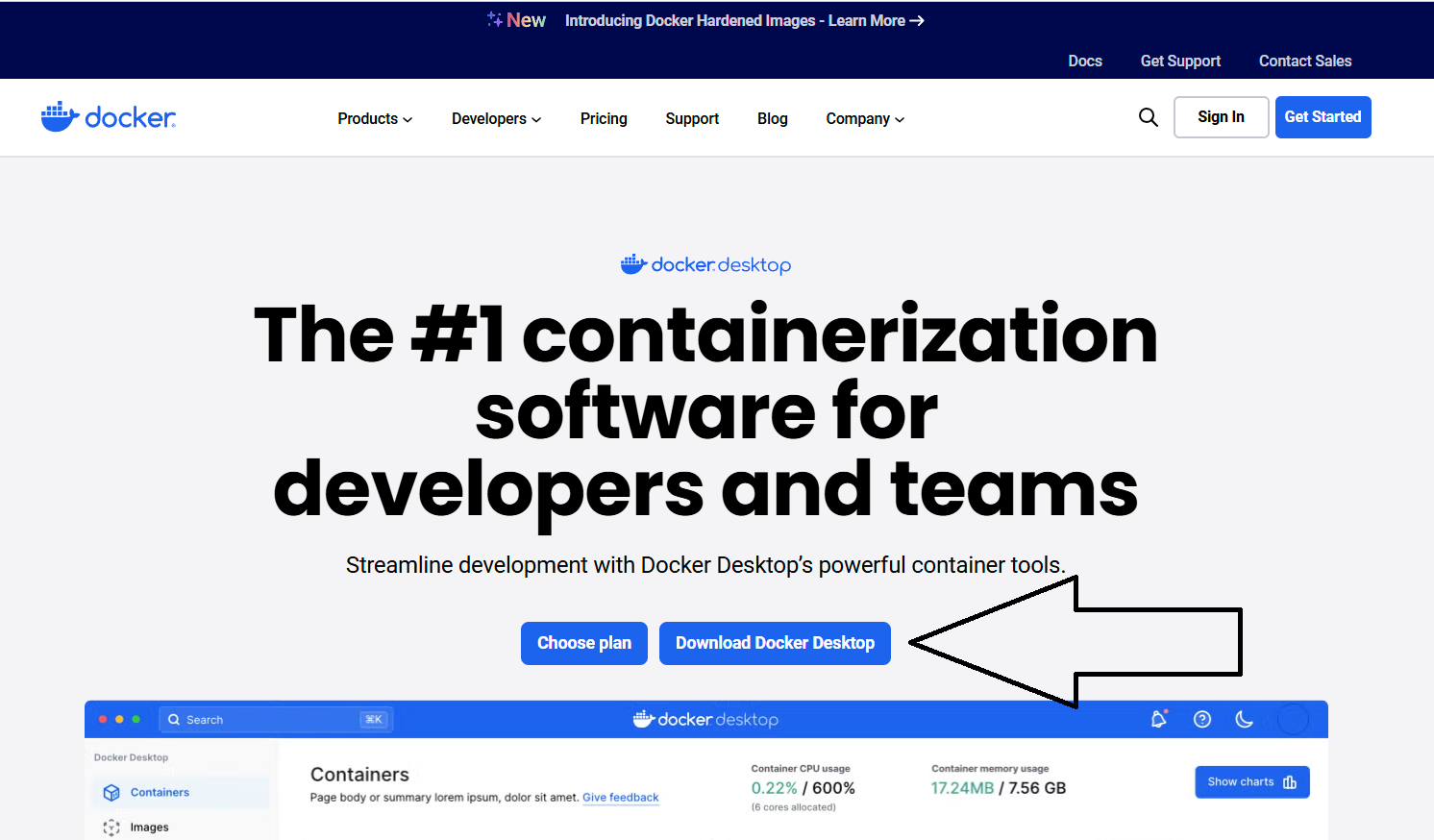
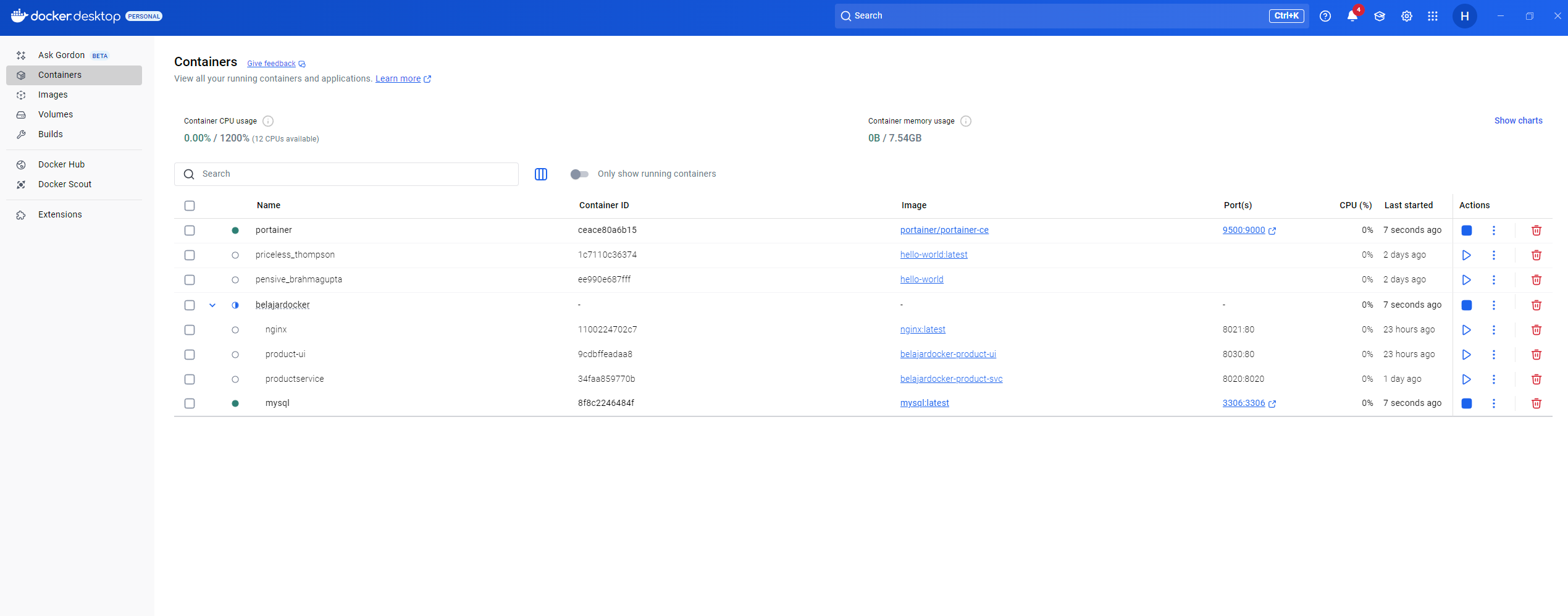
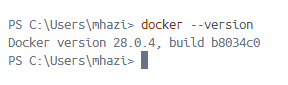
|  |  |  |
| --- | --- | --- |
| **Command** | **Tujuan** | **Contoh** |
| docker --version | Semak versi Docker yang dipasang | docker --version |
| docker pull | Muat turun imej dari Docker Hub | docker pull nginx |
| docker images | Senarai semua imej dalam sistem | docker images |
| docker run | Jalankan kontena baru daripada imej | docker run nginx |
| docker run -d | Jalankan kontena dalam latar belakang (detached mode) | docker run -d nginx |
| docker run -p host:container | Peta port antara host dan kontena | docker run -p 8080:80 nginx |
| docker ps | Lihat kontena yang sedang berjalan | docker ps |
| docker ps -a | Lihat semua kontena (termasuk yang telah dihentikan) | docker ps -a |
| docker stop | Hentikan kontena yang sedang berjalan | docker stop my\_container atau docker stop abc123 |
| docker rm | Padam kontena | docker rm my\_container atau docker rm abc123 |
| docker rmi | Padam imej Docker | docker rmi nginx |
| docker exec -it | Akses terminal dalam kontena (interactive) | docker exec -it my\_container bash |
| docker build -t nama:tag . | Bina imej daripada Dockerfile | docker build -t myapp:latest . |
| docker compose up | Jalankan semua servis dalam fail docker-compose.yml | docker compose up |
| docker compose up -d | Jalankan docker-compose dalam latar belakang | docker compose up -d |
| docker compose down | Hentikan dan buang semua servis dari docker-compose | docker compose down |

Nota Tambahan:

* -d = detached (jalan di belakang tanpa keluar output di terminal)
* -p = port mapping, contohnya -p 8080:80 bermakna akses aplikasi dari localhost:8080
* -it = interactive mode + terminal (untuk masuk dalam kontena)
* **docker compose** digunakan bila anda mahu urus banyak kontena (multi-container apps) seperti frontend + backend + database serentak.

LATIHAN AMALI

# 1. PROSEDUR PEMASANGAN DOCKER DESKTOP (WINDOWS)

1. **Akses laman rasmi Docker.**  
   Layari laman web rasmi Docker di pautan berikut:  
   <https://www.docker.com/products/docker-desktop>
2. **Muat turun pemasang.**  
   Klik pada butang muat turun untuk sistem operasi Windows. Pemasang biasanya berformat .exe.  
     
   
3. **Jalankan pemasang.**  
   Klik dua kali pada fail .exe untuk memulakan proses pemasangan.
4. **Ikuti arahan pemasangan.**  
   Semasa pemasangan, pastikan pilihan untuk memasang WSL 2 diaktifkan (jika menggunakan Windows Home). Teruskan mengikut arahan yang diberikan.
5. **Selesaikan pemasangan.**  
   Tunggu sehingga pemasangan selesai sepenuhnya. Sistem mungkin akan meminta untuk dimulakan semula (restart).
6. **Buka Docker Desktop.**  
   Setelah selesai, buka aplikasi Docker Desktop dari menu Start.  
     
   
7. **Sahkan pemasangan Docker.**  
   Buka Command Prompt atau PowerShell dan taip arahan berikut:  
   docker –version  
     
   

# 2. DEPLOY APLIKASI MENGGUNAKAN DOCKER

# 2.1 Objektif

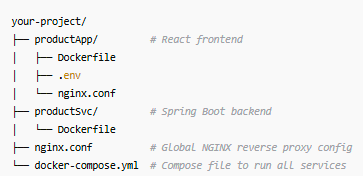
* Memahami proses pembungkusan (containerization) aplikasi React dan Spring Boot ke dalam Docker.
* Membina dan menjalankan sistem penuh secara automatik dengan docker-compose.

# 2.2 Keperluan

* **Docker Desktop** terpasang
* **Git** dan **VS Code** (atau teks editor lain)

# 2.3 Struktur folder

Untuk memudahkan latihan, dua projek React (frontend) dan Spring Boot (backend) yang telah disediakan boleh dimuat turun dari pautan berikut:

<http://gitlab.cybersolution.com.my/mbdk-tot/product-management-docker/repository/master/archive.zip>  
  
Selepas fail zip dimuat turun, **extract (nyahzip)** kandungan fail tersebut ke dalam direktori kerja anda, contohnya ke dalam folder:  
  
C:\Users\<nama-anda>\Documents\DockerProductApp\  
  
Pastikan struktur folder selepas extract adalah seperti berikut:  
  


# 2.4 Deploy React App

# 2.4.1 Langkah Menyediakan Dockerfile untuk Aplikasi React

Sila pastikan fail **Dockerfile** berada di dalam folder yourproject/productApp dan mengandungi skrip berikut:

|  |
| --- |
| FROM node:16-alpine AS build  WORKDIR /app  COPY package\*.json ./  RUN npm install  COPY . .  RUN npm run build  FROM nginx:stable-alpine  COPY --from=build /app/build /usr/share/nginx/html  COPY nginx.conf /etc/nginx/conf.d/default.conf  EXPOSE 80  CMD ["nginx", "-g", "daemon off;"] |

Jika fail ini tiada, sila cipta fail baru bernama Dockerfile secara manual dalam direktori productApp, kemudian salin dan tampal skrip di atas ke dalamnya.

Penjelasan setiap arahan Dockerfile:

|  |  |
| --- | --- |
| **Arahan** | **Fungsi** |
| FROM node:16-alpine AS build | Gunakan image Node.js versi ringan (alpine) untuk membina aplikasi React. |
| WORKDIR /app | Tetapkan direktori kerja dalam kontena. |
| COPY package\*.json ./ | Salin fail konfigurasi npm untuk pemasangan dependencies. |
| RUN npm install | Pasang dependencies projek React. |
| COPY . . | Salin semua kod sumber ke dalam kontena. |
| RUN npm run build | Bina (compile) projek React ke dalam folder build. |
| FROM nginx:stable-alpine | Gunakan NGINX versi ringan untuk hidangkan aplikasi yang telah dibina. |
| COPY --from=build /app/build /usr/share/nginx/html | Salin output binaan ke direktori NGINX. |
| COPY nginx.conf /etc/nginx/conf.d/default.conf | Salin fail konfigurasi NGINX ke lokasi yang sesuai. |
| EXPOSE 80 | Dedahkan port 80 (port HTTP standard) untuk diakses dari luar. |
| CMD ["nginx", "-g", "daemon off;"] | Jalankan NGINX sebagai servis utama dalam kontena. |

# 2.4.2 Langkah Menyediakan nginx.conf untuk Aplikasi React

Sila pastikan fail **nginx.conf** berada di dalam folder yourproject/productApp dan mengandungi skrip berikut:

|  |
| --- |
| server {  listen 80;  location / {  root /usr/share/nginx/html;  index index.html;  try\_files $uri $uri/ /index.html;  }  } |

Jika fail ini tiada, sila cipta fail baru bernama **nginx.conf** secara manual dalam direktori productApp, kemudian salin dan tampal skrip di atas ke dalamnya  
  
**nota**: Konfigurasi ini menetapkan pelayan NGINX untuk mendengar di port 80 dan menyajikan fail React dari direktori /usr/share/nginx/html. Jika pengguna akses URL yang tidak wujud secara fizikal, NGINX akan kembalikan index.html, membolehkan React Router mengendalikan routing aplikasi.

# 2.4.3 Menjalankan React App dalam Docker

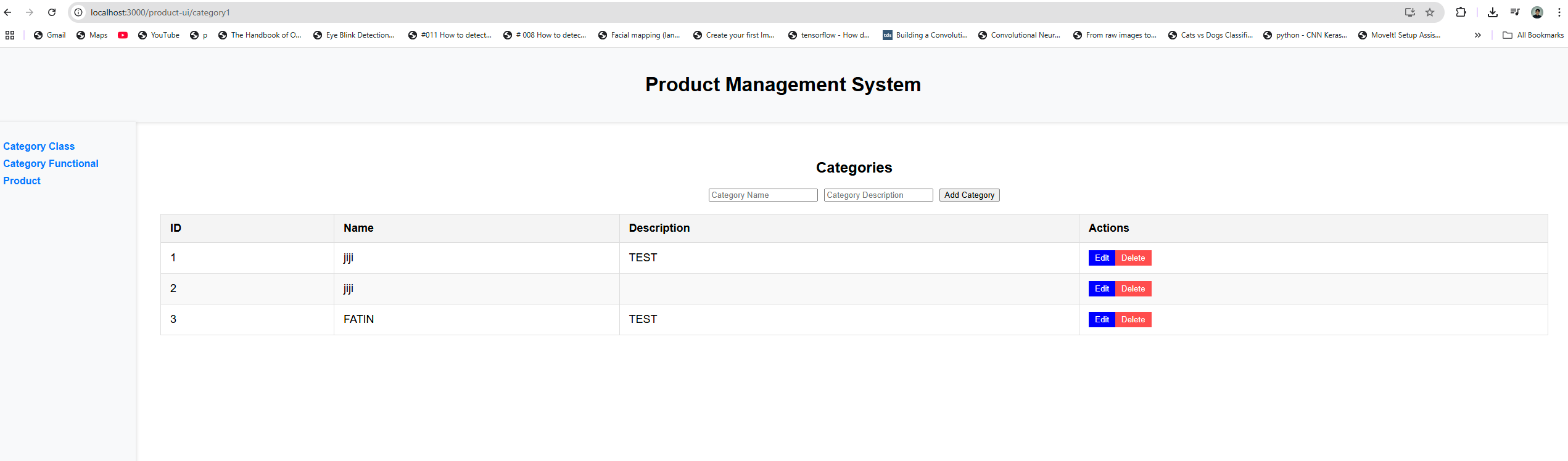
Jalankan arahan ini di terminal:

|  |
| --- |
| cd productApp  docker build -t product-ui .  docker run -d -p 3000:80 --name product-ui product-ui  docker ps |

Setelah semua arahan dijalankan, sila pastikan container product-ui sedang berjalan dengan menjalankan arahan docker ps. container product-ui sepatutnya tersenarai dalam output tersebut sebagai petunjuk bahawa aplikasi React anda telah berjaya dideploy. Contoh:



Dan boleh Akses aplikasi anda di [http://localhost:3000/](http://localhost:3000/product-ui) .



# 2.5 Deploy Java Spring Boot

# 2.5.1 Dockerfile

Sila pastikan fail **Dockerfile** berada di dalam folder yourproject/ productSvc dan mengandungi skrip berikut:

|  |
| --- |
| FROM eclipse-temurin:21-jdk-alpine AS builder  WORKDIR /app  COPY . .  RUN ./gradlew clean build -x test  FROM eclipse-temurin:21-jdk-alpine  WORKDIR /app  COPY --from=builder /app/build/libs/\*.jar app.jar  EXPOSE 8080  ENTRYPOINT ["java", "-jar", "app.jar"] |

Jika fail ini tiada, sila cipta fail baru bernama Dockerfile secara manual dalam direktori productSvc, kemudian salin dan tampal skrip di atas ke dalamnya.

Penjelasan setiap arahan Dockerfile:

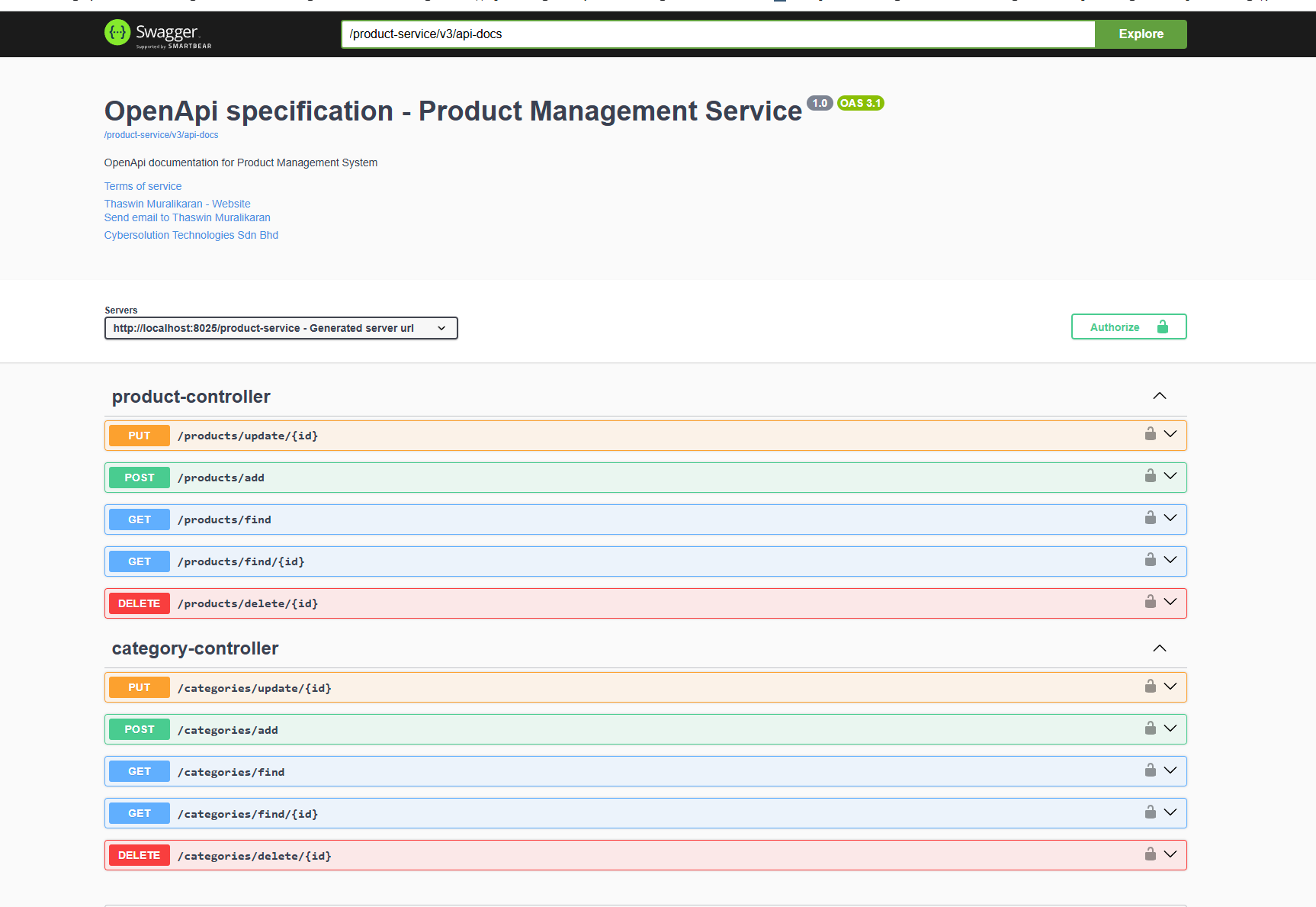
|  |  |
| --- | --- |
| **Arahan Dockerfile** | **Penerangan** |
| FROM eclipse-temurin:21-jdk-alpine AS builder | Gunakan imej base JDK 21 berasaskan Alpine untuk proses bina, dan beri nama stage ini sebagai builder. |
| WORKDIR /app | Tetapkan direktori kerja dalam kontena kepada /app. |
| COPY . . | Salin semua fail dari folder projek ke dalam direktori /app di dalam kontena. |
| RUN ./gradlew clean build -x test | Jalankan Gradle untuk membersihkan dan membina projek tanpa menjalankan ujian (untuk percepatkan bina). |
| FROM eclipse-temurin:21-jdk-alpine | Mulakan imej baru JDK 21 berasaskan Alpine sebagai imej akhir, tanpa fail binaan. |
| WORKDIR /app | Tetapkan direktori kerja untuk imej akhir juga ke /app. |
| COPY --from=builder /app/build/libs/\*.jar app.jar | Salin fail .jar yang telah dibina dari stage builder ke direktori kerja imej akhir dan namakan sebagai app.jar. |
| EXPOSE 8080 | Beritahu Docker bahawa kontena akan menggunakan port 8080. |
| ENTRYPOINT ["java", "-jar", "app.jar"] | Tetapkan arahan lalai untuk menjalankan aplikasi Java dengan fail app.jar apabila kontena dimulakan. |

# 2.5.2 Menjalankan Java Spring Boot dalam Docker

Jalankan arahan ini di terminal:

|  |
| --- |
| cd productSvc  docker network create my-network  docker run -d --network my-network --name mysql-db -e MYSQL\_ROOT\_PASSWORD=S3cret -e MYSQL\_DATABASE=productmngm -p 3306:3306 mysql:latest  docker build -t product-svc .  docker run -d --network my-network -p 8025:8025 --name product-service -e APP\_INTERNAL\_PORT=8025 -e DB\_ADDRESS=mysql-db -e DB\_PORT=3306 -e DB\_NAME=productmngm -e DB\_USERNAME=root -e DB\_PASSWORD=S3cret product-svc  docker ps |

Setelah semua arahan dijalankan, sila pastikan kontena sedang berjalan dengan menjalankan arahan docker ps. Kontena product-svc sepatutnya tersenarai dalam output tersebut sebagai petunjuk bahawa aplikasi Java Spring Boot anda telah berjaya dideploy. Contoh:

  
  
Dan boleh Akses aplikasi anda di [http://localhost:8025/product-service/swagger-ui/index.html#/](http://localhost:8025/product-service/swagger-ui/index.html%23/)  
  


# 3. Langkah deploy aplikasi multi-container menggunakan Docker Compose dan NGINX sebagai reverse proxy

# 3.1 Kandungan docker-compose.yml

Fail docker-compose.yml menyenaraikan semua servis yang anda ingin jalankan. Contohnya:

|  |
| --- |
| version: '3.8'  services:  product-svc:  container\_name: productservice  build: ./productSvc  ports:  - "8020:8020"  environment:  APP\_INTERNAL\_PORT: 8020  DB\_ADDRESS: mysql  DB\_PORT: 3306  DB\_NAME: productmngm  DB\_USERNAME: root  DB\_PASSWORD: S3cret  depends\_on:  - mysql  networks:  - my-network  mysql:  container\_name: mysql  image: mysql:latest  restart: always  environment:  MYSQL\_ROOT\_PASSWORD: S3cret  MYSQL\_DATABASE: productmngm  ports:  - "3306:3306"  volumes:  - mysql-data:/var/lib/mysql  networks:  - my-network  nginx:  container\_name: nginx  image: nginx:latest  ports:  - "80:80"  volumes:  - ./nginx.conf:/etc/nginx/nginx.conf:ro  depends\_on:  - product-svc  networks:  - my-network  product-ui:  container\_name: product-ui  build: ./productApp # path to your React app Dockerfile folder  ports:  - "8030:80"  networks:  - my-network  volumes:  mysql-data:  networks:  my-network:  driver: bridge |

# 3.2 NGINX sebagai reverse proxy

Apa Itu **Reverse Proxy**?

Secara ringkas:

**Reverse proxy** ialah *“orang tengah”* antara client (browser) dengan backend server (API, database, frontend app dll).

Analogi Mudah:  
Bayangkan anda pergi ke kaunter servis pelanggan di pejabat kerajaan:

* Anda (client) tak terus jumpa pegawai belakang.
* Anda jumpa **kaunter depan** dulu (reverse proxy).
* Kaunter ini tengok permintaan anda, dan **salurkan** ke jabatan yang betul (backend server).

Fungsi Reverse Proxy:

* **Terima permintaan dari client** (browser).
* **Salurkan ke backend** server tertentu (Spring Boot, Node.js, React, dsb).
* Hantar semula **jawapan dari backend** ke client.

Kandungan nginx.conf:

|  |
| --- |
| events {}  http {  server {  listen 80;  location /product-service/ {  proxy\_pass http://productservice:8020/product-service/;  }  location /product-ui/ {  proxy\_pass http://product-ui:80/;  }  }  }  } |

# 3.3 Bina dan Jalankan Semua Servis

Dalam terminal, jalankan arahan:

|  |
| --- |
| docker-compose up -d --build |

* **Build** semua servis (kalau ada Dockerfile),
* Jalankan dalam **background (-d)**

# 3.4 Periksa Sama Ada Semua Container Berjalan

Gunakan perintah:

|  |
| --- |
| docker ps |

Pastikan semua container seperti productservice, mysql, product-ui, dan nginx aktif. Seperti:

# 3.5 Akses Aplikasi Anda di Pelayar (Browser)

* Frontend (React App)  
  <http://localhost/product-ui>
* API (Swagger UI)  
  <http://localhost/product-service/swagger-ui/index.html>

# 