Arkitektur og Konfigurasjon

Dette oppsettet definerer en multi-container-applikasjon ved hjelp av Docker, der hver container har en spesifikk rolle i applikasjonens stack. De viktigste komponentene er:

1. **API (catalog-api)**:
    - Denne containeren huser applikasjonens API, som håndterer produkt- og brukerforespørsler.
    - Den lytter på port `8080` internt i containeren, som er mappet til port `80` eksternt.
2. **Database (catalog-db)**:
    - Denne containeren kjører database-serveren (sannsynligvis MySQL eller MariaDB), som er ansvarlig for å lagre data relatert til produkter og brukere.
    - Den lytter på port `3306`, som er mappet til port `4444` på vertsmaskinen.
3. **Server (catalog-server-nginx)**:
    - Denne containeren kjører Nginx som en reverse proxy for å rute forespørsler mellom frontend og API.
    - Nginx videresender forespørsler til API-containeren basert på forhåndsdefinerte ruter.
4. **Nettverkskonfigurasjon**:
    - Alle tre containerne er koblet sammen gjennom et delt Docker-nettverk (`mynetwork`), slik at de kan kommunisere med hverandre.

Hvordan Komponentene Samhandler

1. **API**: `catalog-api` containeren leverer backend-API for produkter og brukere. Den behandler API-forespørsler som å hente en liste med produkter, hente detaljert informasjon om et produkt og spørre om brukerinformasjon.
2. **Database**: `catalog-db` containeren lagrer dataene som API-en får tilgang til. Den håndterer SQL-forespørsler fra API-en for å hente, sette inn eller oppdatere produkt- og brukerinformasjon.
3. **Nginx**: `catalog-server-nginx` containeren fungerer som en reverse proxy, og ruter innkommende HTTP-forespørsler fra klienten til de riktige backend API-rutene. Den håndterer:
    - `/api/products`: For å liste alle produkter.
    - `/api/products/{id}`: For å hente et spesifikt produkt etter ID.
    - `/api/users/{id}`: For å hente en bruker etter ID.
    - `/api/health`: For API-helse-sjekk endpointet, som bekrefter om API-en er operativ.
4. Reverse proxy-konfigurasjonen er definert i `nginx.conf`-filen, som videresender trafikk basert på rute-mønstre til `catalog-api` containeren.
5. **Docker Nettverk**: Alle containerne er koblet sammen på samme Docker bridge-nettverk (`mynetwork`), slik at de kan kommunisere med hverandre. Dette er viktig for at Nginx skal kunne sende forespørsler til API-et og API-et til databasen.

Testing med Postman

For å teste API-endepunktene, kan jeg bruke Postman til å sende forespørsler til den kjørende API-en. Kommandoene under antar at applikasjonen kjører og er tilgjengelig via `localhost`:

1. Hent alle produkter

- **Metode**: GET
- **URL**: `http://localhost:80/api/products`

2. Hent et produkt etter ID

- **Metode**: GET
- **URL**: `http://localhost:80/api/products/{productId}`
    - Erstatt `{productId}` med den faktiske ID-en til et produkt.

3. Hent en bruker etter ID

- **Metode**: GET
- **URL**: `http://localhost:80/api/users/{userId}`
    - Erstatt `{userId}` med den faktiske ID-en til en bruker.

4. API Helse Sjekk

- **Metode**: GET
- **URL**: `http://localhost:80/api/health`
    - Dette vil returnere en status som bekrefter om API-en er sunn og operativ.

Docker Konfigurasjon

Nginx Konfigurasjon (`nginx.conf`)

```
server {
    listen 81;
    root /usr/share/nginx/html;
    server_name localhost;

    # Reverse Proxy for å hente alle produkter
    location /api/products {
        proxy_pass http://catalog-api:8080/v1/products/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Reverse Proxy for å hente et produkt etter ID
```

```
    location ~* ^/api/products/(\d+)$ {
        proxy_pass http://catalog-api:8080/v1/products/$1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Reverse Proxy for å hente en bruker etter ID
    location ~* ^/api/users/(\d+)$ {
        proxy_pass http://catalog-api:8080/api/users/$1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # API Helse Sjekk endpoint
    location /api/health {
        proxy_pass http://catalog-api:8080/api/health;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Dockerfile (for API container)

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY *.csproj ./
RUN dotnet restore

COPY . ./
RUN dotnet publish -c Release -o /app/out

FROM mcr.microsoft.com/dotnet/aspnet:8.0
WORKDIR /app
COPY --from=build /app/out .

EXPOSE 80

ENTRYPOINT ["dotnet", "products-api.dll"]
```

Docker Compose Fil

```
services:
  api:
    container_name: catalog-api
    build: C:\Users\musta\products-api
    ports:
    - "8080:80"
    networks:
    - mynetwork

  db:
    container_name: catalog-db
    build: C:\Users\musta\products-api\sql-scripts
    ports:
    - "4444:3306"
    networks:
    - mynetwork

  server:
    container_name: catalog-server-nginx
    build: C:\Users\musta\products-api\nginx
    ports:
    - "80:81"
    depends_on:
      - api
    networks:
    - mynetwork

networks:
  mynetwork:
    driver: bridge
```