GOKSTAD AKADEMIET

Eksamen Emne 8 Cloud-teknikker, web-arkitektur og container teknologi

Utlevering	**14. februar 2025**
Innlevering	**28. februar 2025**
Hjelpemidler	**Alle**
Innlevering	**Leveres samlet i en zip-fil**
Vedlegg	**database_setup.sql**

✔ Oppgave 1: Enkel API-løsning med Docker (20%)

Oppgavebeskrivelse

Design og implementer en enkel mikrotjeneste-arkitektur med følgende Docker-tjenester:

Backend API

- Utvikle et REST API. Hvilket språk du velger for APIet er opp til deg
- API-et må støtte følgende funksjonalitet:
  - Hente en liste med produkter.
  - Hente et spesifikt produkt basert på ID.
  - En helsesjekk-endepunkt for å verifisere at tjenesten kjører.
- **Eksempel på JSON-respons fra API:**

```
{
  "id": 1,
  "name": "Laptop",
  "brand": "Dell",
  "price": 12999,
  "stock": 50
}
```

- API-et må kunne testes uavhengig på port `8080`.
- API-et må være containerisert med en `Dockerfile`.

MySQL Database

- Opprett en MySQL-container som en del av `docker-compose.yml`.
- Konfigurer MySQL med følgende detaljer:
  - Databasenavn: `product_db`
  - Brukernavn: `product-api`
  - Passord: `securepass`
- Konfigurer persistent lagring av MySQL-data.
- API-et må koble til MySQL-databasen for å hente og lagre produktdata.

Nginx Reverse Proxy

- Konfigurer en `nginx.conf`-fil slik at Nginx fungerer som en reverse proxy for API-et.
- URL-er som skal kunne testes via Nginx:
  - `http://localhost/api/products`
  - `http://localhost/api/products/{id}`
  - `http://localhost/api/health` (trenger ikke health fra database - kun return 'API OK' direkte fra Rest API)
- Nginx skal være containerisert og kjøres i samme Docker-nettverk.

Containeroppsett

- Bruk `docker-compose.yml` for å orkestrere tjenestene.
- Sikre at API-et, Nginx og MySQL kommuniserer internt i et felles Docker-nettverk.
- Start løsning med `docker-compose up -d`.
- **Alle konfigurasjonsfiler brukt i konfigurasjonen av `docker-compose` må leveres inn**.

Dokumentasjon

- Skriv en kort beskrivelse av arkitekturen og konfigurasjonen.
- Forklar hvordan løsningene samhandler.
- Legg ved kommandoer for testing med `curl` eller Postman.

✔ Oppgave 2: Publisering av Docker-images til Docker Hub (20%)

1. Test løsningen lokalt for å sikre at den fungerer som forventet.
2. Opprett en konto på **Docker Hub** hvis du ikke allerede har en.
3. Bygg og tag Docker-images for API og Nginx.
4. Push de byggede Docker-images til din private eller offentlige Docker Hub repository.
5. Lag nye `docker-compose.yml` slik at den ikke bruker `build`-kommandoen, men isteden refererer til image-tagene fra Docker Hub.
6. Dokumenter stegene og inkluder link til dine Docker Hub-repositories.

✔ Oppgave 3: Deploy API til AWS EC2 med Nginx (20%)

1. Opprett en **egen VPC** i AWS.
2. Opprett en **EC2-instans** i Free Tier (Amazon Linux eller Ubuntu) og sørg for at EC2-instansen er opprettet innenfor denne VPC-en.
3. Installer og konfigurer **Docker** og **Docker Compose**.
4. Konfigurer `docker-compose.yml` for å trekke ned API og Nginx-images fra Docker Hub.
5. Eksponer API-et via Nginx på port `80` slik at det er offentlig tilgjengelig.
6. For testing: Tillatt testing direkte på API på port 8080
7. Test API-et med `curl`, 'swagger', 'scalar' eller 'Postman'. Testene skal kjøres mot public IP'en til EC2-instansen.
8. Dokumenter stegene og IP-adressen til den kjørende tjenesten.
9. **Lag en video** (5-10 minutter) som viser hele prosessen, inkludert:
    - Opprettelse av EC2-instans i den opprettede VPC-en.
    - Hvordan man kobler til EC2-instansen med SSH.
    - Overføring av 'docker-compose.yml' til serveren med `scp`.
    - Hvordan `.pem`-filen brukes for sikker tilkobling.
    - Installasjon av Docker
    - Start docker-compose.yml (`docker-compose up -d`)
    - Test av API-et som nå kjører på en EC2 maskin med en public addresse.

✔ Oppgave 4: Migrering til AWS RDS (20%)

1. Opprett en **MySQL RDS-instans** i Free Tier innenfor samme VPC som EC2-instansen.
2. Konfigurer RDS med samme detaljer som i oppgave 1:

- Databasenavn: `product_db`
- Brukernavn: `product-api`
- Passord: `securepass`
- Tillat tilkobling fra EC2-instansens sikkerhetsgruppe

3. Modifiser `docker-compose.yml` på EC2-instansen:

- Fjern MySQL-containeren
- Endre APIet til å snakke med RDS-databasen; bruk RDS-endepunktet som hostnavn.

4. Sett opp den initielle `Products` tabellen i RDS. Hvordan den settes opp er opp til deg; om det gjøres direkte i app-koden eller gjennom f.eks. `mysql`-CLI. Bootstrapping-script som kan brukes ligger vedlagt i `database_setup.sql`.
5. Test API-et på nytt for å verifisere:

- Tilkobling mot RDS fungerer
- Data hentes korrekt fra RDS

6. Dokumenter:

- RDS-oppsett og konfigurasjon
- Endringer i `docker-compose.yml`
- Sikkerhetsgruppe-konfigurasjoner

7. **Utvid videoen** fra Oppgave 3 (eller lag en ny) som viser:

- Oppsett av RDS-instansen
- Konfigurering av sikkerhetsgrupper
- Testing av API mot RDS

✔ Oppgave 5: AWS CloudWatch Monitorering (20%)

1. Installer og konfigurer **CloudWatch Agent** på EC2-instansen:

- Last ned og installer CloudWatch Agent
- Konfigurer nødvendige IAM-roller for EC2-instansen
- Verifiser at agenten kjører korrekt

2. Implementer en **custom CloudWatch metric** i API-et:

- Lag en mekanisme for å telle antall API-kall
    - Send denne informasjonen til CloudWatch som en custom metric
    - Metrikken skal ha navnet `ApiCallCount`
    - Bruk namespace `ProductApi`

3. Konfigurer CloudWatch Dashboard:

- Opprett et Cloudwatch dashboard
- Legg til graf som visualiserer antall API-kall som gjøres per tid ved å bruke metrikken `ApiCallCount` i en Cloudwatch Dashboard Widget.

4. Dokumenter:

- IAM-rolle konfigurasjon
- CloudWatch Agent oppsett
- Dashboard konfigurasjon
- Kodeendringer i API-et for metrikk-logging

5. **Utvid videoen** fra tidligere oppgaver (eller lag en ny) som viser:

- Installasjon av CloudWatch Agent
- Konfigurasjon av custom metrics
- Demonstrasjon av Cloudwatch dashboard med metrikkdata

Husk å fjerne alle AWS-ressurser når du er ferdig for å unngå unødvendige kostnader.

Vurdering

For Oppgave 1:

- API-funksjonalitet og korrekt JSON-respons.
- Korrekt oppsett av Docker-containerne.
- Nginx reverse proxy fungerer som forventet.
- MySQL-databasen er satt opp korrekt og kommuniserer med API-et.
- Tjenestene fungerer sammen i `docker-compose.yml`.
- **Alle konfigurasjonsfiler for `docker-compose` er levert inn**.

For Oppgave 2:

- Lokalt testing av løsningen er gjennomført.
- Images er bygget, tagget og publisert på Docker Hub.
- `docker-compose.yml` er oppdatert for å bruke Docker Hub-images.
- Dokumentasjon av stegene er levert.

For Oppgave 3:

- Korrekt oppsett av EC2-instans i opprettet VPC.
- Nginx fungerer og eksponerer API-et.
- AWS-infrastruktur satt opp korrekt (VPC, sikkerhetsgrupper, osv.).
- Dokumentasjon av stegene og IP-adressen til den kjørende løsningen.
- Videoen viser en tydelig demonstrasjon av oppgaven, inkludert SSH, SCP og `.pem`-filbruk.

For Oppgave 4:

- RDS-instansen er korrekt konfigurert i VPC
- Sikkerhetsgruppe er satt opp til å tillate trafikk inn fra kun EC2-instansen sin sikkerhetsgruppe
- API kommuniserer godt med RDS
- Dokumentasjon av RDS-oppsett er fullstendig
- Demonstrasjonsvideo viser tydelig hvordan RDS er blitt tatt i bruk mot APIet

For Oppgave 5:

- CloudWatch Agent er korrekt installert og konfigurert
- Custom metrics sendes og vises i CloudWatch
- Dashboard er satt opp med relevante grafer
- Video demonstrerer alle aspekter av oppgaven

Lykke til! 🎯