AWS 2 - Skalbar Dockermiljö

Index

Verktyg och tjänster

- Obsidian
- VSCode
- GitHub
- AWS + AWS CLI
- Docker
- ChatGPT

Inledning

Målet med denna uppgiften är att lägga up en fungerande skalbar dockerapplikation och en CI/CD pipeline för att driftsätta den i Amazon Web Services. Under hela projektet försökte jag hålla mig till namnstrukturen <tjänsttyp> och om inte det var tillräckligt så la jag till mer beskrivande ord.

Definitioner

- AWS = Amazon Web Services
- CI/CD = Continious Integration / Continious Deployment
- CLI = Command Line Interface
- AWS tjänster:
    - IAM = Identity and Access Management
    - ECS = Elastic Container Service
    - ECR = Elastic Container Registry
    - SG = Security Group
    - ALB = Application Load Balancer
    - TG = Target Group

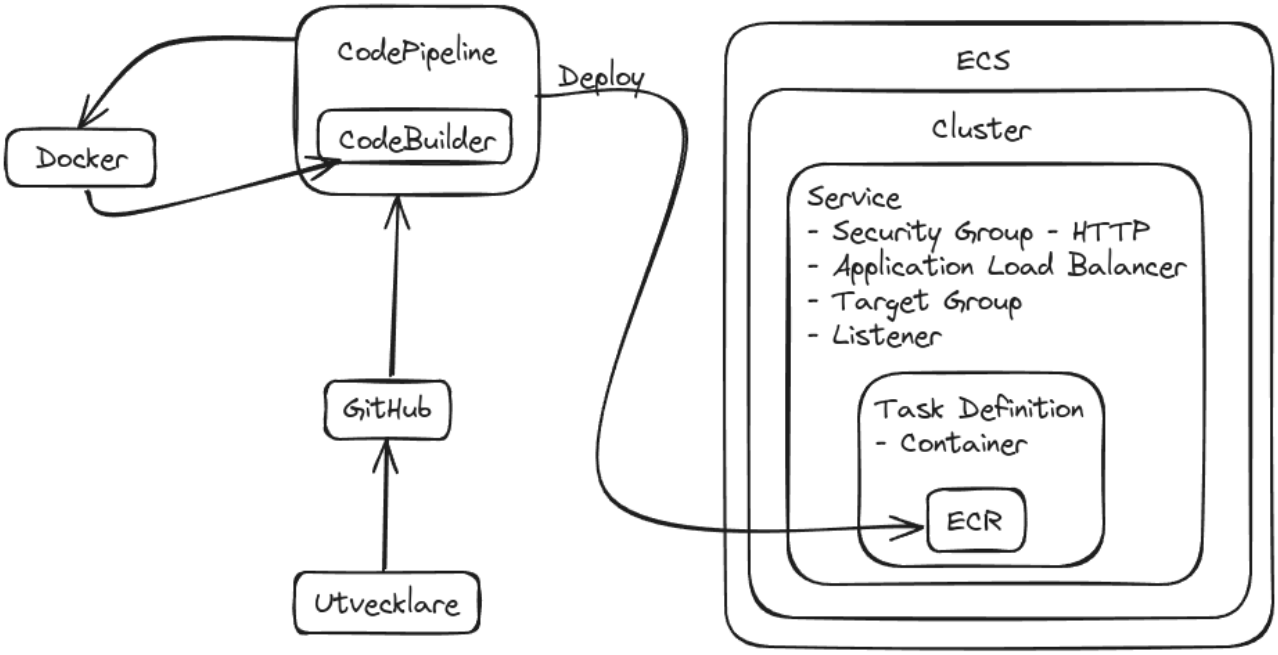
Krav

- Kan skalas automatiskt via load balancer.
- Applikationen ligger i en containerbaserad lösning som Docker.
- CI/CD pipeline som kan uppdatera applikationen via git push till sin repository.

- Använda så många Amazon Web Services som möjligt.

Systemarkitektur - Översikt

Så när jag försökte få en översikt av alla delar så kändes det som att det såg ut ungefär såhär:



Där Pipeline uppdaterar ECR och skickar tillbaka den nya filen till docker och när man triggar ECR så uppdateras hela ECS strukturen.

Utnyttjade molntjänster

- AWS:
  - ECR
  - ECS
  - Codebuild
  - CodePipeline
  - IAM
  - Target Groups
  - Security Groups
- GitHub
- ChatGPT

Provisionering av driftmiljö och driftsättning av applikation

Jag testar här ett nytt format där jag går igenom och gör själva uppgiften och sen lägger in min förståelse i ett steg för steg format.

1. Skapa en enkel hemsida som vi kan köra i Docker containern.
2. Initiera en docker container med Amazon Linux 2023, installera NGINX på den och öppna port 80 för webbtrafik.

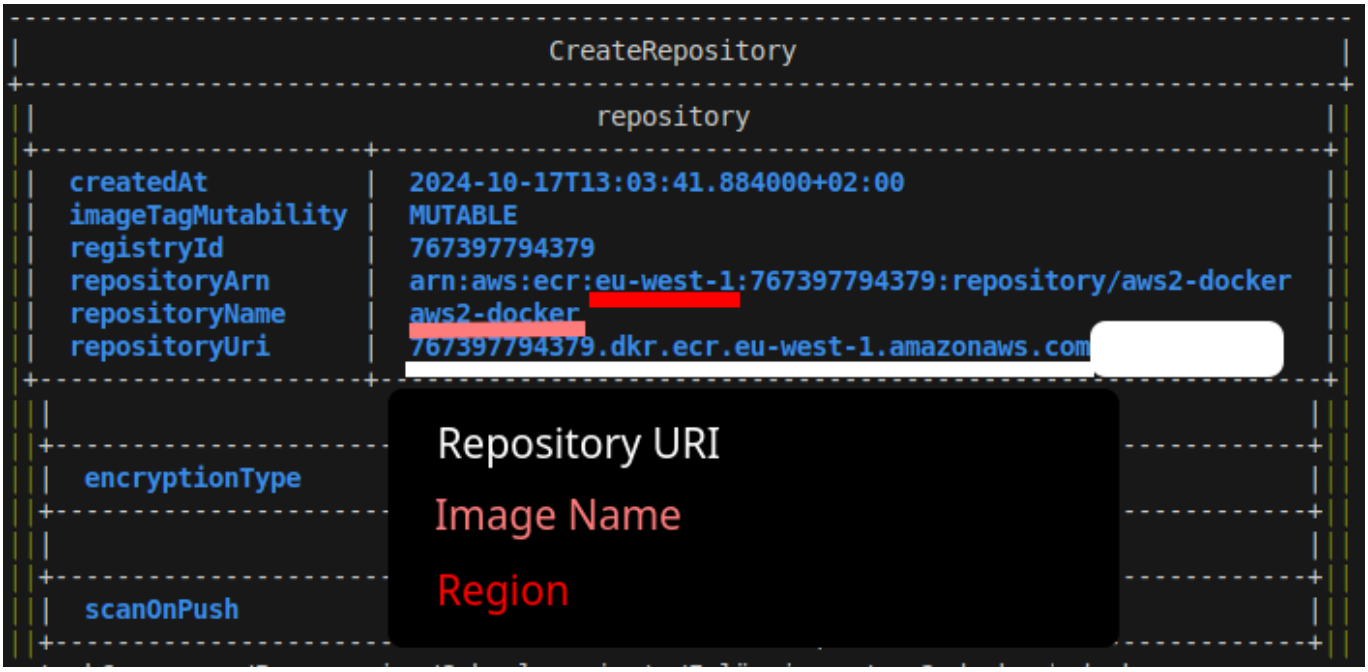
```
FROM amazonlinux:2023

RUN dnf update -y
RUN dnf install -y nginx
COPY ./html /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

3. Skapa en GitHub repository på deras hemsida och intitiera en git repository på datorn som du sen gör en commit och laddar upp till github.

```
git init
git add .
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:<git user>/<repo name>.git
git push origin main
```

4. Skapa en ECR med detta kommandot och skriv ner informationen som är understruken:

```
aws ecr create-repository --repository-name <projekt-namn>
```



5. Nu använder vi den informationen för att göra våran buildspec.yml som kommer hantera ordningen saker händer när vi lägger in Build och pipeline stegen.

```
version: 0.2

phases:
  pre_build:
    commands:
      # Fill in ECR information
      - REGISTRY_URI=767397794379.dkr.ecr.eu-west-1.amazonaws.com #
Registry URI
      - IMAGE_NAME=aws2-docker # Repository namnet
      - REGION=eu-west-1 # Hitta regionen från bildreferensen.
      # Fill in ECS information
      # Detta kommer bli namnet du använder i din Task definition Container
så spara detta.
      - CONTAINER_NAME=AWS2-DockerContainer
      # ----------------------
      - IMAGE=$REGISTRY_URI/$IMAGE_NAME
      - COMMIT=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-8)
      # Vi har en inlogg till docker här för annars kanske man inte kan
ladda ner amazon operativsystemfilen.
      - echo $DOCKERHUB_PASSWORD | docker login -u $DOCKERHUB_USERNAME --
password-stdin
      - aws ecr get-login-password --region $REGION | docker login --
username AWS --password-stdin $REGISTRY_URI
  build:
    commands:
      - docker build --tag $IMAGE .
      - docker tag $IMAGE $IMAGE:$COMMIT
  post_build:
    commands:
      - docker push $IMAGE
      - docker push $IMAGE:$COMMIT
      # Create imagedefinitions.json. This is used by ECS to know which
docker image to use.
      - printf '[{"name":"%s","imageUri":"%s"}]' $CONTAINER_NAME
$IMAGE:$COMMIT > imagedefinitions.json
artifacts:
  files:
    # Put imagedefinitions.json in the artifact zip file
    - imagedefinitions.json
```

5. Skapa ett CodeBuild projekt
    - Klicka på create project.
    - Source
        - Source borde vara GitHub
        - Anslut till GitHub genom att clicka på "Manage default source credential", Detta kommer öppna en ny flik.
        - Använd Github App och klicka på texten med länk som säger "create a new GitHub connection".

        - Ett nytt fönster kommer öppnas där du kan logga in till GitHub. När du har skapat den så väljer du den i dropdown och trycker "Save".
        - Klicka på sökrutan udner "Repository" så borde du få en lista av dina repositories som finns på ditt konto. Antingen sök eller scrolla igenom listan för att välja rätt repository för ditt projekt.
        - Under "Source version" se till att du väljer eller skriver branchen som du ska använda.
    - Använd en Webhook
    - Environment
        - Operating system ska vara Ubuntu
        - Runtime to Standard
        - Image to aws/codebuild/standard:7.0
        - Under **Additional configuration** så behöver du klicka i rutan för "Privileged" och lägga till två **Environment variables**. **DOCKERHUB_USERNAME** och **DOCKERHUB_PASSWORD**. Användarnamnet är Case Sensitive och **MÅSTE** vara små bokstäver för att kunna logga in.
    - Skapa en ny Service roll och notera namnet.
    - Under Buildspec sektionen behöver du ändra till att använda en fil istället för kommandorutan.
    - Klicka Create project.
    - För att avsluta denna sektionen behöver du gå till IAM, Roles, Klicka på service rollen du nyss skapat och lägga till policyn **AmazonEC2ContainerRegistryPowerUser** till den.
    - Starta Builden.
- Dubbelkolla att container imagen nu finns som en ECR.

6. Skapa Elastic container service med en task definition, cluster och service.
    1. ECS Task
        - Använd AWS fargate för task definitionen
        - Använd samma namn under Container details -> Name som CONTAINER_NAME i din buildspec.
        - När du lägger till Container details Image URI så behöver du / från den tidigare bilden i det formatet.
        - Klicka Create
    2. ECS Cluster
        - Cluster namn och skapa det.
    3. ECS Service
        - Skapa en service i ditt Cluster
        - Family namnet behöver du fylla i men gär ingen skillnad, jag tog projektets namn.
        - Service namn
        - Sätt Desired Tasks till mer än 1.
        - Öppna networking delen och använd eller skapa en ny SG som är öppen till HTTP.
        - Öppna upp delen för Load Balancing, Välj ALB och ge den ett namn.
        - Skapa Service.

7. Efter ett par minuter gå in i ditt ECS Cluster, Klicka på Service och kolla i tasks fliken. Där klickar du på task namnet så kommer du kunna hitta IP'n för din docker container. Verifiera så att när du klickar på IP'n kommer du till sidan du skapade innan. Sen går vi till EC2 -> Load Balancer -> Kopiera DNS namnet av din load balancer och öppna en ny flik för att gå till den sidan för att verifiera att Load balancern också fungerar.
8. Skapa en CodePipeline
    - Välj Build a custom pipeline sen Next.
    - Namnge den som vanligt och klicka Next.
    - Source
        - Github Version 1
        - Anslut till github, Borde vara två klick.
        - Välj Repositoryn för ditt projekt och branchen, Sen klicka Next.
    - Build
        - Välj Other Build providers och välj AWS Codebuild från dropdownen.
        - Välj Codebuild projektet vi gjorde tidigare.
        - Klicka next.
    - Deploy Provider borde vara Amazon ECS.
    - Cluster och Service name ska vara samma som dem du skapade tidigare i projektet.
    - Skapa pipelinen.
9. När Pipelinen blir klar, ändra något i din hemsida och skicka upp ändringen till GitHub, Vänta tills den blir klar igen och kolla på din ALB länk från tidigare för att verifiera att ändringar går igenom.

Slutsats

Ärligt talat, Dessa instruktioner visar inte mycket av problemen jag hade med detta projektet. Det finna många smådelar som enkelt missas. På hela alltet så är det trevligt att kunna få upp en CI/CD pipeline och se allting fungera.

Oturligt nog för mig så försvann Amazon CodeCommit innan jag gjorde detta projektet och jag var tvungen att gå runt det och testa flera olika lösningar som använde GitHub i större kapacitet än jag hade tänkt mig.

Kollade även lite dokumentation för att se ifall det fanns någon möjlighet att göra en mer IaC struktur på det hela men gav upp på det för AWS dokumentationen kändes sämre än Azure/Google's dokumentation för samma saker.