Group 3 DevOps Document

Gemaakt door:

Renas Khalil

Sadek Al Mousawi

Sam Maijer

Daniel Pustjens

Module: DevOps | WFSDAD.DO.23

Versie: 3.0

Datum: 23-06-2024

Inhoudsopgave

1. Projectkeuze

We hebben ervoor gekozen om **SL2B** te gebruiken voor ons projectkeuze, omdat we veel van de voorbeelden die voorkomen in lessen van **SL2B & DevOps** toegepast kunnen worden in het project zelf. Op die manier begrijpen we niet alleen de lesstof beter, maar kunnen we ook sneller en efficiënter werken, waarbij we meer kennis opbouwen.

Daarnaast biedt SL2B handige ingebouwde tools aan om onze code te testen. Door het gebruik van deze tools kunnen wij ervoor zorgen dat onze applicatie vanaf het begin stabiel is en dat eventuele fouten vroegtijdig worden ontdekt en opgelost. Hierdoor kunnen we een sterke applicatie bouwen en ervoor zorgen dat alles soepel blijft draaien.

Verder maken we gebruik van het volgende voor het project:

Sprints: waar wij elke **twee weken** een demo opleveren. Deze demo gaat hand in hand met de werkprincipes van **DevOps**.

Unit tests en build tests: Docker build images en geautomatiseerde testen.

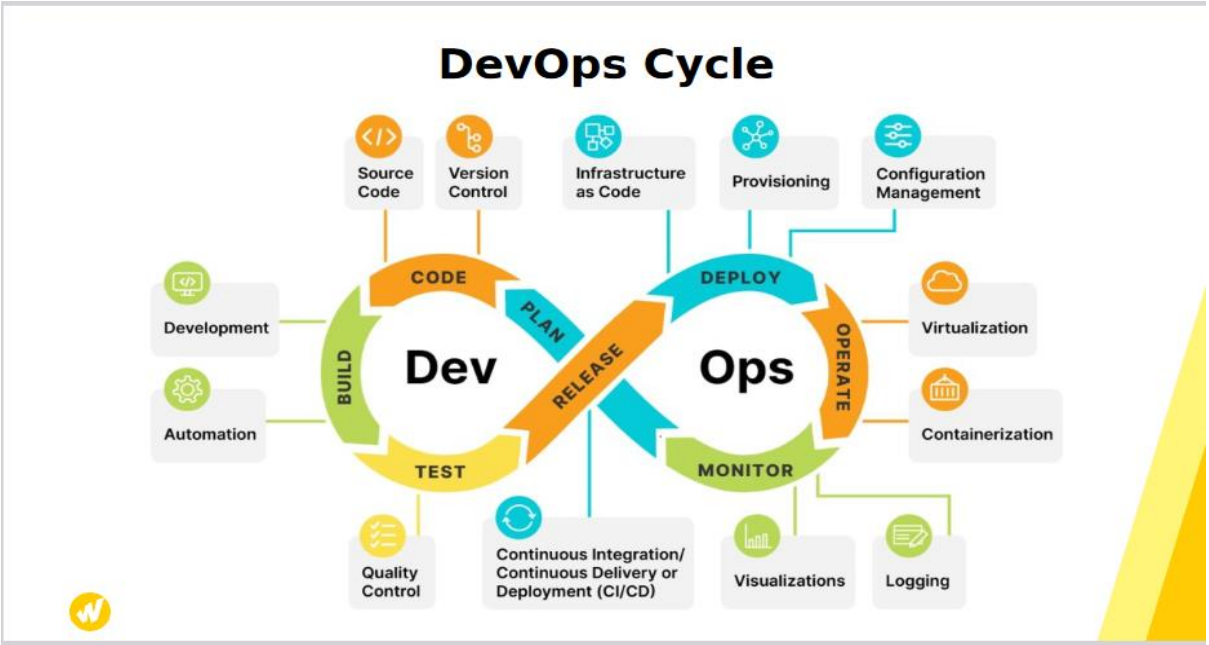
CI/CD-pipelines: Git branches die automatisch worden getest en gemerged als ze de testen met succes voltooien

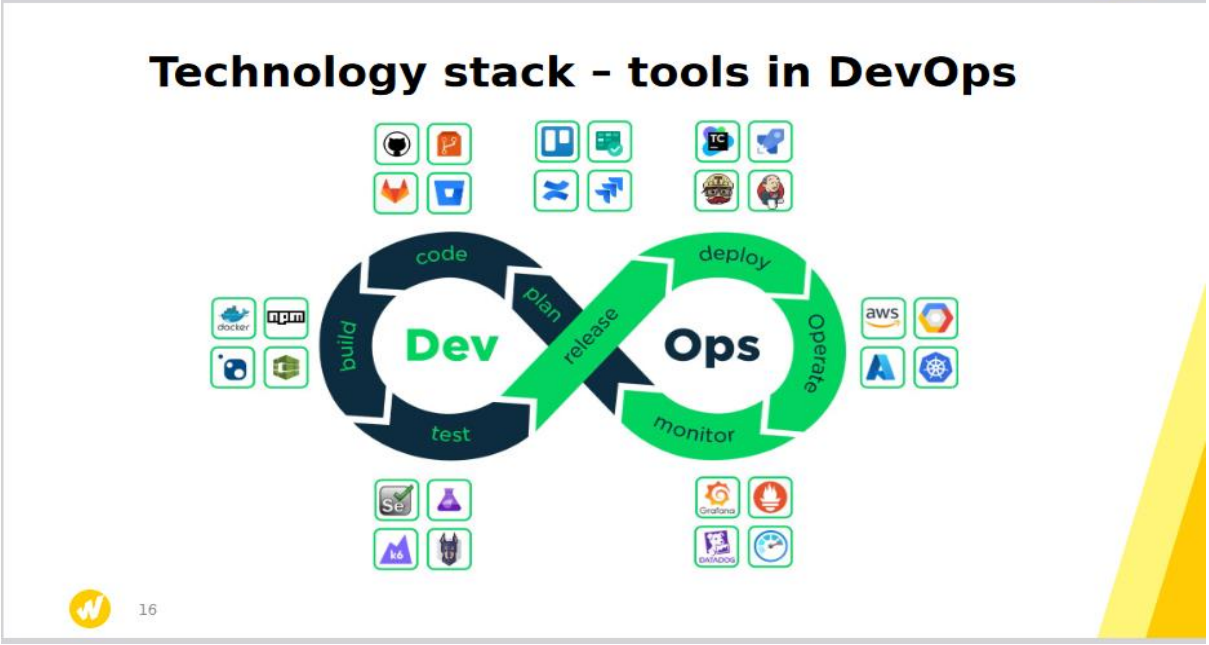
SL2B voldoet ook aan alle eisen om gehost te worden op de Kubernetes cluster en we kunnen het project monitoren qua performances.

Dit zijn de algemene redenen die ons hebben overtuigd om **SL2B** te gebruiken voor het **DevOps** traject.

2. DevOps

1. Plan: Trello board
2. Code: Github
3. Build: Docker
4. Test: C# unit test + Docker unit test
5. Release: Kubernetes cluster
6. Deploy: Docker
7. Operate: Kubernetes cluster
8. Monitor: Grafana





3. Praktijken & Technieken

1. **Plan: Trello board**

Wij hebben ervoor gekozen om Trello te gebruiken omdat het een gebruiksvriendelijke gebruikersinterface heeft en we er veel meer ervaring mee hebben vergeleken met Azure. Het geeft ons de vrijheid om de board te designen naar onze keuze. Hierdoor is de trelloboard zelf overzichtelijk en fijn om mee te werken.

2. **Code: Github**

Ons team heeft al veel ervaring met GitHub, dus de keuze was snel gemaakt om ermee te gaan werken. Het zorgt ervoor dat wij branches kunnen maken en hierdoor continu aan het project kunnen werken zonder dat wij elkaar in de weglopen.

3. **Build: Docker**

Docker is een tool die ervoor zorgt dat wij onze applicatie op elke operating system kunnen runnen.

4. **Test: C# unit test + Docker unit test**

C# unit test maakt gebruik van de ASP.NET core systeem, wat ervoor zorgt dat er al zelf veel testen aanwezig zijn en wij dit kunnen gebruiken voor het automatiseren van onze testen.

Docker unit test zorgt ervoor dat wij onze builds kunnen testen en configureren voor elke operating systeem.

5. **Release: Kubernetes cluster**

In ons traject wordt er van ons verwacht dat wij ons project releasen en operate op de Kubernetes cluster die aanwezig zou zijn tijdens onze lessen.

6. **Deploy: Docker**

We maken gebruik van docker om onze build image te creëren en aan de hand van deze image het project te operaten.

7. **Operate: Kubernetes cluster**

In ons traject wordt er van ons verwacht dat wij ons project releasen en operate op de Kubernetes cluster die aanwezig zou zijn tijdens onze lessen.

8. **Monitor: Grafana**

We hebben onderzoek gedaan voor monitoring en hieruit kwam Grafana omdat het opensource is en het ons een mogelijkheid geeft om ene dashboard te creëren.

4. Aandachtspunten & Verbeterstrategieën

1. Overzichtelijker maken van epics en user story's. Voorheen wisten we niet perfect hoe je epics en user story's moest maken/koppelen, waardoor het onoverzichtelijk was welke user story bij welke epic hoorde. Verder wisten we niet precies wanneer de user story helemaal klaar zou zijn, omdat we niet veel acceptatiecriteria hadden genoteerd.

   Oplossing: Elke epic heeft een nummer en bijhorende user story met daarbij ook een nummer. De branches worden bijvoorbeeld vernoemd ”1.2-inlog-systeem”. Verder wordt er voor elke user story acceptatiecriteria gemaakt, zodat de persoon die eraan werkt weet wanneer het af zou zijn.

2. Er wordt gelet op uitstelgedrag. Voorheen hadden we in het begin moeite om meteen aan de eindopdrachten te werken, wat aan het einde van de periode tot veel werk en veel stress leidde.

   Oplossing: We gaan het verminderen doormiddel van Sprints, Daily stand-ups en een planning bijhouden.

3. Er wordt gelet op taakverdeling. Voorheen hadden we moeite met het verdelen van taken, waardoor één persoon veel meer deed dan de rest.

   Oplossing: We letten op dat iedereen aan het werk is en actief meedoet aan elke fase.

4. Er zal meer onderzoek worden gedaan naar de concepten van DevOps zodat wij een beter begrip krijgen over de verschillende fases van DevOps en wellicht verandering van de tools die wij gebruiken in het project.

   Toelichting: Voor vrijwel iedereen in ons projectgroepje zijn de devops-concepten nieuw, dit betekent dat wij als groepje gezamenlijk nog veel onderzoek moeten doen om erachter te komen hoe wij het willen aanpakken voor ons project.

5. Actieplan

5.1 Structuur epics en user story's

Actie 1.1: Nummeringssysteem implementeren. Elk epic en user story krijgt een uniek nummer. Voorbeeld: Epic 1 kan hebben User Stories 1.1, 1.2, enz.

Actie 1.2: Standaard branch-namen aanhouden. Gebruik het nummeringssysteem in de naam van de branches om duidelijkheid en vindbaarheid te verbeteren, zoals 1.2-inlog-systeem.

Actie 1.3: Acceptatiecriteria definiëren. Duidelijke en meetbare acceptatiecriteria voor elke user story op te stellen, zodat het duidelijk is wanneer een taak als voltooid beschouwd kan worden.

5.2 Verminderen van uitstelgedrag

Actie 2.1: Sprints inplannen. Een sprintduur van twee weken aanhouden en strikt aan de geplande sprint reviews en planningssessies houden. Er worden daar o.a. verantwoordelijken toegewezen, deadlines vastgesteld en eventuele aanpassingen gemaakt.

Actie 2.2: Dagelijkse stand-ups implementeren. Elke schooldag een korte stand-up meeting houden om de voortgang te bespreken en problemen/valkuilen snel aan te pakken.

Actie 2.3: Voortgang visueel bijhouden. Gebruik maken van Trello board om taken te visualiseren en de status van elk item bij te werken.

5.3 Verbetering van taakverdeling

Actie 3.1: Rollen en verantwoordelijkheden definiëren. Elk teamlid moet duidelijke verantwoordelijkheden hebben en deze evenwichtig verdeeld moeten zijn.

5.4 Verdieping in DevOps concepten

Actie 4.1: Gezamenlijke leersessies organiseren. Regelmatige sessies plannen waarin teamleden onderzoek doen en hun resultaten over DevOps concepten delen.

Actie 4.2: Feedback en evaluatie van gebruikte tools.

6. Docker

6.1 Wat is Docker?

Docker is een platform dat softwarepakketten, genaamd containers, creëert, distribueert en uitvoert. Containers bevatten alles wat nodig is om een applicatie te draaien, inclusief code, runtime, bibliotheken en systeemhulpmiddelen, waardoor applicaties consistent en geïsoleerd op verschillende omgevingen kunnen worden uitgevoerd.

Voor **Docker** hebben we 3 gerelateerd bestanden in ons project

1. .dockerignore
2. Dockerfile
3. Dockerfile-sqlite

6.2 Dockerignore

De .dockerignore-file voorkomt dat bepaalde bestanden en mappen meegenomen worden in de Docker-image. Hiermee kan de grootte van de image verminderd worden en de buildtijd van de image verbeterd en beveiligd worden.

We hebben gekozen voor een standaardtemplate voor ASP.NET-projecten. Hieronder is een voorbeeld van de onze .dockerignore-file:

6.3 Dockerfile

Een Dockerfile is een tekstbestand met instructies en commando's die Docker gebruikt om een Docker-image te bouwen. Het bevat de stappen die nodig zijn om de applicatie en de bijbehorende omgeving te configureren en uit te voeren.

Het doel van een Dockerfile is om het bouwproces van de Docker-image te automatiseren en te vereenvoudigen, zodat de omgeving consistent en reproduceerbaar is.

Dit is onze dockerfile:

```
# Use the .NET SDK image with multi-platform support
FROM --platform=$BUILDPLATFORM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG TARGETARCH
ARG BUILD_CONFIGURATION=Release
WORKDIR /src

# Copy and build Dierentuin-App
COPY ["Dierentuin-App/Dierentuin-App.csproj", "Dierentuin-App/"]
RUN dotnet restore -a $TARGETARCH "Dierentuin-App/Dierentuin-App.csproj"
COPY Dierentuin-App/ Dierentuin-App/
WORKDIR "/src/Dierentuin-App"
RUN dotnet build "Dierentuin-App.csproj" -c $BUILD_CONFIGURATION -o /app/build

# Copy the appsettings.json file
COPY Dierentuin-App/appsettings.json .

# Switch back to /src
WORKDIR /src

# Copy and test Dierentuin-unit-test
COPY ["Dierentuin-unit-test/Dierentuin-unit-test.csproj", "Dierentuin-unit-test/"]
RUN dotnet restore -a $TARGETARCH "Dierentuin-unit-test/Dierentuin-unit-test.csproj"
COPY Dierentuin-unit-test/ Dierentuin-unit-test/
WORKDIR "/src/Dierentuin-unit-test"
RUN dotnet test --logger:trx

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
WORKDIR "/src/Dierentuin-App"
RUN dotnet publish "Dierentuin-App.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

# Final stage
FROM --platform=$TARGETARCH mcr.microsoft.com/dotnet/aspnet:8.0
WORKDIR /app
```

```
# Set root user to install dependencies
USER root

COPY --from=publish /app/publish .

# Create a non-root user and group
RUN addgroup --system appgroup && adduser --system --ingroup appgroup appuser

# Set the user to the newly created non-root user
USER appuser

# Set environment variable for ASP.NET Core to use port 8080
ENV ASPNETCORE_URLS=http://+:8080

# Expose port 8080
EXPOSE 8080

ENTRYPOINT ["dotnet", "Dierentuin-App.dll"]
```

6.4 Uitleg dockerfile:

Build Stage

De Dockerfile begint met het gebruik van mcr.microsoft.com/dotnet/sdk:8.0 als basis, wat zorgt voor een omgeving met .NET SDK 8.0, geschikt voor het bouwen van .NET-applicaties. Met de optie --platform=$BUILDPLATFORM wordt de image geschikt gemaakt voor meerdere platforms, waardoor het buildproces op verschillende architecturen kan worden uitgevoerd. Het .csproj-bestand van de "Dierentuin-App" wordt gekopieerd naar de container, waarna dotnet restore de benodigde afhankelijkheden installeert. Daarna worden de overige bronbestanden gekopieerd en wordt de applicatie gebouwd.

Kopiëren van Configuratiebestand

Na de buildfase wordt het appsettings.json-bestand, dat de applicatieconfiguraties van .NET bevat, gekopieerd naar de werkdirectory.

Test Stage

Het .csproj-bestand van de "Dierentuin-unit-test" wordt gekopieerd, en vervolgens wordt dotnet restore uitgevoerd om de benodigde dependencies voor de testomgeving te installeren. Daarna worden de testbestanden gekopieerd en uitgevoerd met dotnet test, waarbij de testresultaten worden vastgelegd in een .trx-bestand.

Publish Stage

De applicatie wordt gepubliceerd naar de map /app/publish in de container, waarbij de configuratie en afhankelijkheden worden vastgelegd voor productie.

Final Stage

In de laatste fase wordt aspnet:8.0 gebruikt als basisimage om de runtime-omgeving voor onze .NET-applicatie op te zetten. Vervolgens wordt een niet-root gebruiker aangemaakt voor betere beveiliging. ASPNETCORE_URLS wordt ingesteld om de applicatie te laten luisteren op poort 8080. De container start de applicatie op met: dotnet Dierentuin-App.dll.

6.5 Dockerfile-sqlite

Helaas is het ons niet gelukt om de database live te krijgen op de Kubernetes-cluster zelf. We hebben hiervoor verschillende technieken gebruikt, waarvan Dockerfile-sqlite er een was. Ons doel was om een image te creëren voor de database zelf en deze in dezelfde omgeving te laten draaien en communiceren met onze applicatie.

Dockerfile:

```
Dockerfile-sqlite > ...
1   # Dockerfile.sqlite
2   FROM alpine:latest
3
4   RUN apk --no-cache add sqlite
5
6   WORKDIR /app
7
8   COPY Dierentuin_AppContext-8936df66-a63e-4347-9020-18b0e3d246f1.db /data/Dierentuin_AppContext-8936df66-a63e-4347-9020-18b0e3d246f1.db
9
10  CMD ["sqlite3", "Dierentuin_AppContext-8936df66-a63e-4347-9020-18b0e3d246f1.db"]
```

6.6 Nawoord:

Echter, we liepen tegen problemen aan, zoals het ontbreken van permissies om bestanden aan te maken op de pod zelf. Uiteindelijk hebben we een Persistent Volume (PV) en Persistent Volume Claim (PVC) opgezet, maar dit is ons ook niet gelukt.

We hebben ook geprobeerd om een tijdelijke pod aan te maken en het databasebestand van de lokale omgeving naar de pods te kopiëren, maar ook hier liepen we helaas tegen dezelfde problemen aan.

7. Kubernetes

7.1 Wat is Kubernetes?

Kubernetes is een open-source platform dat helpt bij het beheren en automatiseren van de uitrol en het beheer van container-gebaseerde applicaties. Kubernetes helpt bij het coördineren van containers op verschillende machines

Voor **Kubernetes** hebben we 3 geraltered bestanden in ons project

1. deploy.yaml
2. kubeconfig_group03.yaml
3. pv.yaml

7.2 deploy.yaml

Een deploy.yaml is een YAML-bestand dat wordt gebruikt om een Kubernetes Deployment te beschrijven en te configureren. Een Kubernetes Deployment definieert hoe applicaties worden uitgerold, geschaald en beheerd in een Kubernetes-cluster.

Zo ziet onze **deploy.yaml** eruit:

kubernetes-config > ! deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: group03
  namespace: group03
  labels:
    app: group03
spec:
  replicas: 1
  selector:
    matchLabels:
      app: group03
  template:
    metadata:
      labels:
        app: group03
    spec:
      containers:
      - name: group03
        image: renyas/yourimagename:latest
        imagePullPolicy: Always
        ports:
        - name: web
          containerPort: 8080
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        env:
        - name: ASPNETCORE_ENVIRONMENT
          value: "Development"
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: group03
  namespace: group03
spec:
  ports:
    - name: web
      port: 80
      targetPort: 8080
  selector:
    app: group03

---
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: group03route
  namespace: group03
spec:
  entryPoints:
    - websecure
  routes:
  - match: Host(`meow.web.dops.tech`) && PathPrefix(`/`)
    kind: Rule
    services:
    - name: group03
      port: 80
  tls:
    certResolver: myresolver
```

7.3 Uitleg Deploy.yaml

Deployment:

maakt een applicatie genaamd group03 in namespace group03.

Draait één pod met een container gebaseerd op de image renyas/yourimagename:latest. Container luistert op poort 8080, met een beperkt gebruik van CPU en geheugen. Zet de omgeving op "Development".

Service:

Biedt toegang tot de group03 applicatie via poort 80, die verkeer doorstuurt naar poort 8080 van de pod.

IngressRoute:

Richt verkeer van meow.web.dops.tech door naar de group03 service op poort 80, beveiligd met TLS.

7.4 Kubeconfig_group03.yaml

Bevat de configuratie instellingen om toegan te krijgen tot een specifieke cluster. Zo ziet onze **Kubeconfig_group03.yaml** eruit:

Cluster Configuratie:

microk8s-cluster met server https://kube.dops.tech:16443 en een certificaat voor beveiligde verbindingen.

Context:

Definieert group03-context voor interactie met de microk8s-cluster, in de namespace group03, door gebruiker group03-user.

Gebruiker:

Specificeert group03-user met een toegangs-token voor authenticatie.

7.5 pv.yaml

Een PersistentVolume (PV) is een Kubernetes-object dat opslagvoorzieningen in een cluster vertegenwoordigt.

Zo ziet once **pv.yaml** eruit:

kubernetes-config > ! pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: sqlite-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/app/sqlite"
    type: DirectoryOrCreate
```

ApiVersion

Dit geeft aan dat het YAML-bestand gebruikmaakt van de eerste versie van de Kubernetes API voor PersistentVolumes.

Kind PersitendVolume

specificeert dat het object dat wordt gecreëerd een PersistentVolume (PV) is.

Metadata

geeft de naam van de PersistentVolume als sqlite-pv.

Spec

specificeert dat de PersistentVolume 1 Gigabyte opslagruimte heeft.

AccessModes

Dit betekent dat de opslag door één pod tegelijkertijd kan worden gelezen en beschreven.

HostPath

"/mnt/app/sqlite": geeft het pad aan op de host waar de opslag is gevestigd.

DirectoryOrCreate: betekent dat de directory /mnt/app/sqlite wordt gemaakt als deze nog niet bestaat.

8. Github/workflows

8.1 Wat zijn Github/workflows?

GitHub Workflows zijn geautomatiseerde processen die taken uitvoeren binnen een GitHub repository. Ze worden gedefinieerd in YAML-bestanden onder de directory .github/workflows. Workflows kunnen worden geconfigureerd om automatisch te starten bij specifieke gebeurtenissen, zoals het pushen van code naar een repository of het openen van een pull request.

Voor **Github/workflows** hebben we 2 geraltered bestanden in ons project

1. runner.yml
2. test.yml

8.2 Runner.yml

.github > workflows > ! runner.yml

```yaml
name: CI

# Event triggers
on:
  push:
    branches: [ "main" ]
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v3
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: renyas/yourimagename:latest
          platforms: linux/arm64

```

```yaml
deploy:
  runs-on: ubuntu-latest
  needs: validate_and_tests

  steps:
    - uses: actions/checkout@v3

    - name: Setup kubectl
      run: |
        curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/st
        chmod +x ./kubectl
        sudo mv ./kubectl /usr/local/bin/kubectl
    - name: Set up KUBECONFIG
      run: |
        mkdir -p $HOME/.kube
        echo "${{ secrets.KUBE_CONFIG }}" > $HOME/.kube/config

    - name: Deploy to Kubernetes
      run: |
        kubectl apply -f kubernetes-config/deploy.yaml
        kubectl rollout status deployment/group03 -n group03
    - name: Restart Deployment
      run: kubectl rollout restart deployment/group03 -n group03
```

```yaml
validate_and_tests:
  runs-on: ubuntu-latest
  needs: build

  steps:
    - uses: actions/checkout@v3

    - name: Setup kubectl
      run: |
        curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/st
        chmod +x ./kubectl
        sudo mv ./kubectl /usr/local/bin/kubectl
    - name: Create .kube directory
      run: mkdir -p $HOME/.kube

    - name: Set up KUBECONFIG
      run: echo "${{ secrets.KUBE_CONFIG }}" > $HOME/.kube/config

    - name: Infrastructure Test - Check Kubernetes Nodes
      run: kubectl get nodes

    - name: Infrastructure Test - Check Kubernetes Pods
      run: kubectl get pods --all-namespaces

    - name: Infrastructure Test - Check Kubernetes Services
      run: kubectl get services --all-namespaces
```

De GitHub workflow is ontworpen om automatisch een reeks acties uit te voeren op basis van bepaalde triggers, zoals het pushen naar de main branch. Het bestaat uit drie hoofdtaken: **build, validate_and_tests, en deploy.**

Triggers:

Push: Deze workflow wordt uitgevoerd wanneer er een push-actie plaatsvindt op de `main` branch.

Workflow_Dispatch: Hiermee kunnen gebruikers de workflow handmatig starten vanuit de GitHub interface.

De taken van de **Build** zijn als volgt**:**

Platform: Draait op ubuntu-latest.

Acties:

Code ophalen: Checkt de code uit de repository.

QEMU instellen: Voor emulatie van verschillende CPU-architecturen.

Docker Buildx instellen: Voor geavanceerde Docker build mogelijkheden.

Inloggen bij Docker Hub: Logt in bij Docker Hub met behulp van geheime variabelen.

Docker image bouwen en pushen: Bouwt en pushed een Docker image naar Docker Hub voor linux/arm64 platform.

De taken van de **Validate en Tests** zijn als volgt**:**

Platform: Draait op ubuntu-latest.

Voorwaarden: Wacht tot de build taak voltooid is.

Acties:

kubectl instellen: Downloadt en installeert kubectl voor interactie met Kubernetes clusters.

.kube map aanmaken: Maakt een directory aan voor Kubernetes configuratie.

KUBECONFIG instellen: Stelt de Kubernetes configuratie in met geheime variabelen.

Infrastructuur testen: Controleert de status van Kubernetes nodes, pods en services.

De taken van de **Deploy** zijn als volgt**:**

Platform: Draait op ubuntu-latest.

Voorwaarden: Wacht tot de validate_and_tests taak voltooid is.

Acties:

kubectl instellen: Installeert kubectl voor het beheer van Kubernetes.

KUBECONFIG instellen: Configureert toegang tot het Kubernetes cluster.

Deploy naar Kubernetes: Past de deployment configuratie toe op het Kubernetes cluster en controleert de status.

Deployment herstarten: Voert een herstart van de deployment uit om eventuele wijzigingen door te voeren.

8.3 Samenvatting:

De workflow automatiseert het proces van het bouwen en pushen van een Docker image, het valideren van de infrastructuur door middel van Kubernetes, en het implementeren en deployen van de applicaties naar een Kubernetes cluster.

8.4 Test.yml

.github > workflows > ! test.yml

```yaml
name: Test

# Event triggers
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
  workflow_dispatch:

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up .NET
        uses: actions/setup-dotnet@v2
        with:
          dotnet-version: '8.0.x'

      - name: Restore dependencies
        run: dotnet restore ./Dierentuin-App/Dierentuin-App.sln

      - name: Run tests
        run: dotnet test ./Dierentuin-App/Dierentuin-App.sln
```

De GitHub workflow genaamd **Test** is ontworpen om automatisch tests uit te voeren voor een .NET applicatie. De workflow wordt geactiveerd door verschillende triggers en voert een reeks stappen uit om de code uit te checken, .NET in te stellen, afhankelijkheden te herstellen en tests uit te voeren.

Triggers:

push: De workflow wordt uitgevoerd wanneer er een push-actie plaatsvindt op de main branch.

pull_request: De workflow wordt ook uitgevoerd wanneer er een pull request wordt geopend of bijgewerkt op de main branch.

workflow_dispatch: Hiermee kunnen gebruikers de workflow handmatig starten vanuit de GitHub interface.

Taken:

Code uit repository ophalen: Deze stap gebruikt de actions/checkout actie om de code uit de GitHub repository te halen.

.NET instellen: Deze stap gebruikt de actions/setup-dotnet actie om de gewenste versie van .NET (hier versie 8.0.x) in te stellen.

Afhankelijkheden herstellen: Voert dotnet restore uit om alle vereiste pakketten en bibliotheken te downloaden die nodig zijn voor de applicatie.

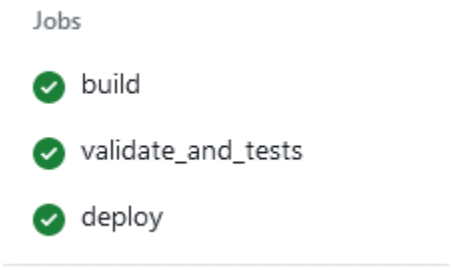
Tests uitvoeren: Voert dotnet test uit op de opgegeven oplossing (.sln bestand) om de eenheidstests van de applicatie uit te voeren.

8.5 Samenvatting:

De workflow zorgt voor geautomatiseerde tests voor een .NET applicatie. Het biedt een betrouwbare en herhaalbare manier om ervoor te zorgen dat de codebase goed functioneert.

9. Feedback peer review:

Als groep hebben wij feedback gekregen om te laten zien in welke Github action job's fases de runner zich bevindt. Deze feedback hebben wij verwerkt en geïmplementeerd als volgt:



We hebben nu een build , validate_and_tests en deploy fase.\

10. Infrastructure Tests

10.1 Infrastructure Test 1 – Check Kubernetes Nodes

```
# Infrastructure Test 1 - Check nodes
- name: Infrastructure Test - Check Kubernetes Nodes
  run: kubectl get nodes
```

Deze test bevestigt de beschikbaarheid en status van de nodes in de Kubernetes-cluster.

10.2 Infrastructure Test 2 – Check Kubernetes Pods

```
# Infrastructure Test 2 - Check pods
- name: Infrastructure Test - Check Kubernetes Pods
  run: kubectl get pods --all-namespaces
```

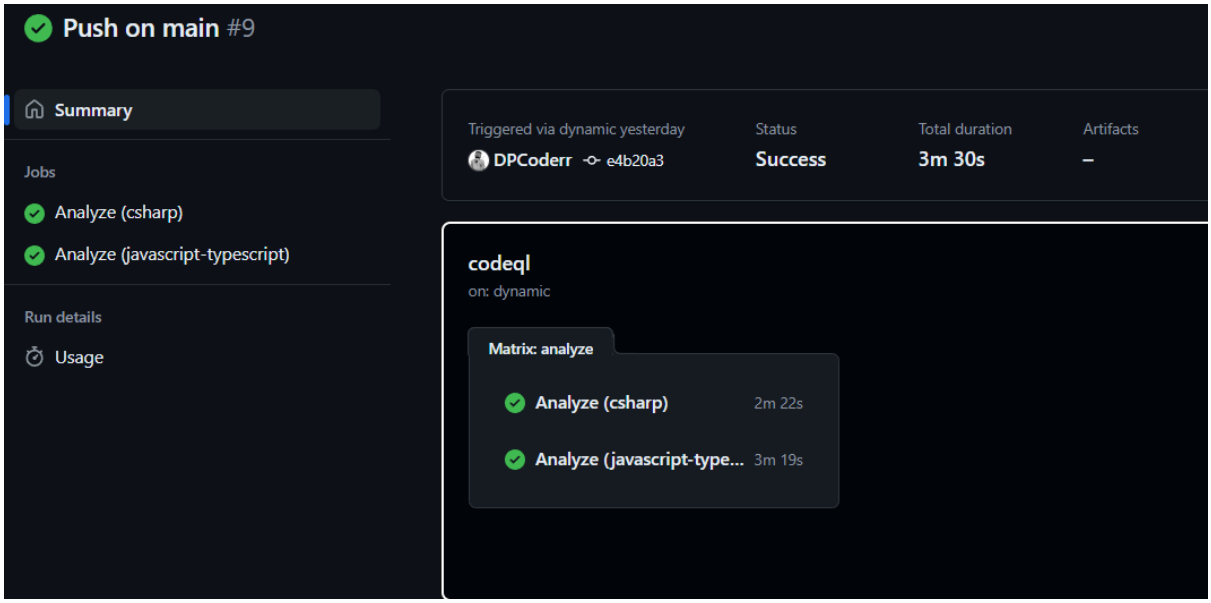
Deze test checkt de status van de pods die zijn gedeployed over alle namespaces in de kuberenetes-cluster

10.3 Infrastructure Test 3 - Check Kubernetes Services

```
# Infrastructure Test 3 - Check services
- name: Infrastructure Test - Check Kubernetes Services
  run: kubectl get services --all-namespaces
```

Deze test verzekert dat alle services, gedefinieerd in de kubernetes-cluster operationeel zijn.

11. Code Quality

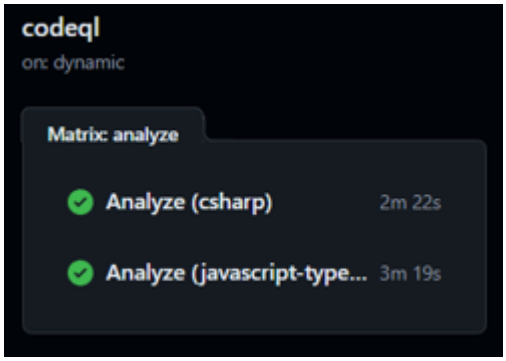


Er is gebruik gemaakt van **GitHub Code Scanning** met behulp van **CodeQL** om de kwaliteit van de code te verbeteren. CodeQL voert analyses uit op de codeconvensies in verschillende programmeertalen, waaronder C#, JavaScript en TypeScript. Dit helpt bij het vinden van beveiligingsproblemen, kwetsbaarheden en andere fouten, waardoor de software betrouwbaarder en veiliger wordt.

12. Code Analyse KPI

Door gebruik te maken van Github Code Scanning zijn er automatisch 2 **code analysis KPI's** uitgevoerd en een **Security KPI.**

12.1 Code Analysis KPI's:



Zoals eerder is vermeld bij het kopje Code Quality, wordt er geanalyseerd op de codeconvensies van C# en Javascript-typescript.

12.2 Security KPI:

GitHub Code Scanning (CodeQL) analyseert automatisch beveiligingskwetsbaarheden en codefouten. Dit zorgt ervoor dat potentiële veiligheidsrisico's op tijd worden geïdentificeerd en aangepakt.

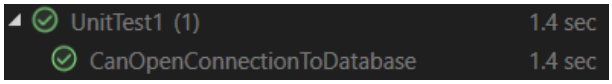
13. Unit testen

```
#21 13.58   Dierentuin-unit-test -> /src/Dierentuin-unit-test/bin/Debug/net8.0/Dierentuin-unit-test.dll
#21 13.61 Test run for /src/Dierentuin-unit-test/bin/Debug/net8.0/Dierentuin-unit-test.dll (.NETCoreApp,Version=v8.0)
#21 13.71 Microsoft (R) Test Execution Command Line Tool Version 17.10.0 (x64)
#21 13.71 Copyright (c) Microsoft Corporation.  All rights reserved.
#21 13.72
#21 13.89 Starting test execution, please wait...
#21 13.94 A total of 1 test files matched the specified pattern.
#21 17.23 Results File: /src/Dierentuin-unit-test/TestResults/_buildkitsandbox_2024-06-23_12_19_11.trx
#21 17.23
#21 17.23 Passed!  - Failed:     0, Passed:    10, Skipped:     0, Total:    10, Duration: 486 ms - Dierentuin-unit-test.dll (net8.0)
#21 DONE 17.4s

#22 [publish 1/2] WORKDIR /src/Dierentuin-App
#22 DONE 0.0s
```

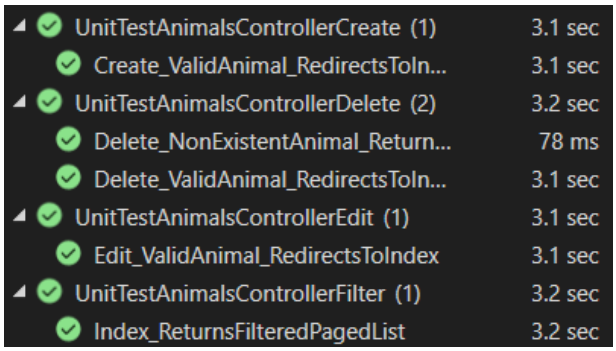
We hebben in totaal 10 unit testen gemaakt die worden uitgevoerd tijdens het deployen van een image naar de main branch. Dit wordt gedaan in de “build” job bij de actions.

13.1 Connection Unit Test



Deze Unit Test checkt of de connectie succesvol is met de database.

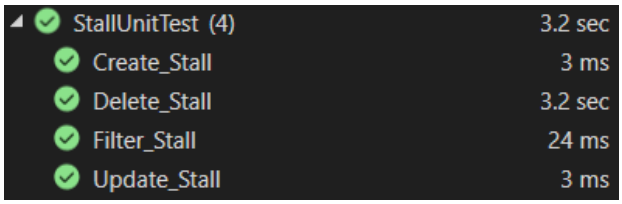
13.2 Animal controller(create/edit/delete/filter)



Deze Unit Testen voeren de volgende testen uit:

- Create Valid Animal
- Delete Non Existing Animal
- Delete Valid Animal
- Edit Valid Animal
- Filter Animal

13.3 Stall controller(create/edit/delete/filter)



Deze Unit Testen voeren de volgende testen uit:

- Create Stall
- Delete Stall
- Edit Stall
- Filter Stall

14. Release Registry

Onze CI/CD-pipeline werkt geautomatiseerd bij code pushes en mergen. Het begint met het bouwen van een Docker-image vanuit onze code. Daarna wordt deze image veilig opgeslagen op Docker Hub. Vervolgens wordt het automatisch geïmplementeerd in ons Kubernetes-cluster. Dit proces garandeert snelle updates, stabiele releases en een betere gebruikerservaring.

15. Monitoring:

We hebben een poging gedaan om monitoring toe te voegen maar dit is gefaald. Wat we gedaan hadden is een yaml file gemaakt die alles aanmaakt wat nodig is om grafana te runnen, maar de Persistent Volume Claim wordt niet goed aangemaakt waardoor de rest vast komt te zitten. We hebben geprobeerd dit op te lossen maar kwamen uiteindelijk er niet uit.

We hadden geprobeerd de PVC te linken aan een PV door middel van een storage ClassName maar deze konden we niet aanmaken en linken.

Ook had een de grafana service en een deployment gemaakt maar die konden niet verder zonder de PVC.

```
PS C:\WINDOWS\system32> kubectl get all --namespace=grafana
Warning: Use tokens from the TokenRequest API or manually created secret-based tokens
NAME                           READY   STATUS    RESTARTS   AGE
pod/grafana-6756f6587b-9hmjn   0/1     Pending   0          11h

NAME              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/grafana   LoadBalancer   10.152.183.213   <pending>     3000:32275/TCP   11h

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana   0/1     1            0           11h

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/grafana-6756f6587b   1         1         0       11h
```

```
Warning: Use tokens from the TokenRequest API or manually created secret-based tokens instead of auto-gen
Name:             grafana-6756f6587b-9hmjn
Namespace:        grafana
Priority:         0
Service Account:  default
Node:             <none>
Labels:           app=grafana
                  pod-template-hash=6756f6587b
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Controlled By:    ReplicaSet/grafana-6756f6587b
Containers:
  grafana:
    Image:      grafana/grafana:latest
    Port:       3000/TCP
    Host Port:  0/TCP
    Requests:
      cpu:        250m
      memory:     750Mi
    Liveness:     tcp-socket :3000 delay=30s timeout=1s period=10s #success=1 #failure=3
    Readiness:    http-get http://:3000/robots.txt delay=10s timeout=2s period=30s #success=1 #failure=3
    Environment:  <none>
    Mounts:
      /var/lib/grafana from grafana-pv (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-4z75q (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  grafana-pv:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  grafana-pvc
    ReadOnly:   false
  kube-api-access-4z75q:
    Type:                     Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:   3607
    ConfigMapName:            kube-root-ca.crt
    ConfigMapOptional:        <nil>
    DownwardAPI:              true
QoS Class:                    Burstable
Node-Selectors:               <none>
Tolerations:                  node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                              node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:                       <none>
```

Het doel van een Kubernetes-monitoren is het observeren, verzamelen en analyseren van gegevens over de prestaties, gezondheid en functionaliteit van de Kubernetes-cluster. Het doel van monitoring is om een duidelijk inzicht te krijgen in de werking van de applicaties en infrastructuur, zodat problemen vroegtijdig gedetecteerd en opgelost kunnen worden, wat resulteert in een stabielere en efficiëntere omgeving.

15.1 Wat is monitoring?

Monitoring in Kubernetes houdt in dat je de hele tijd gegevens verzamelt over de verschillende componenten binnen je cluster, zoals pods, nodes, en services. Deze gegevens kunnen betrekking hebben op CPU-gebruik, geheugengebruik, netwerkverkeer, en andere systeem- en applicatie data.

15.2 Wat doet monitoring?

Monitoring biedt real-time inzicht in de staat van de Kubernetes-omgeving. Het helpt bij:

- **Detecteren van problemen:** Door bugs en fouten snel te identificeren.
- **Analyseren van prestaties:** Door de prestatie bij te houden en te zien waar en wanneer het slomer dan verwacht runt.
- **Optimaliseren van resources:** Door inzicht te bieden in het gebruik van resources, wat helpt bij het efficiënter gebruiken van middelen.

15.3 Doel van monitoring

Het primaire doel van monitoring binnen een Kubernetes-omgeving is om de betrouwbaarheid, prestaties en beschikbaarheid van applicaties te bekijken. Dit wordt gedaan door:

- **Vroegtijdige probleemdetectie:** Zodat bugs snel kunnen worden aangepakt voordat ze eindgebruikers bereiken.
- **Proactieve systeemverbetering:** Door continu inzicht te bieden in de systeemstatus, kunnen verbeteringen worden gemaakt om toekomstige problemen te voorkomen.
- **Ondersteunen van DevOps-praktijken:** Door feedback loops te creëren die belangrijk zijn voor continue integratie en continue levering (CI/CD), wat zorgt voor een snellere en betrouwbaardere levering van nieuwe features en updates.

16. Scrum

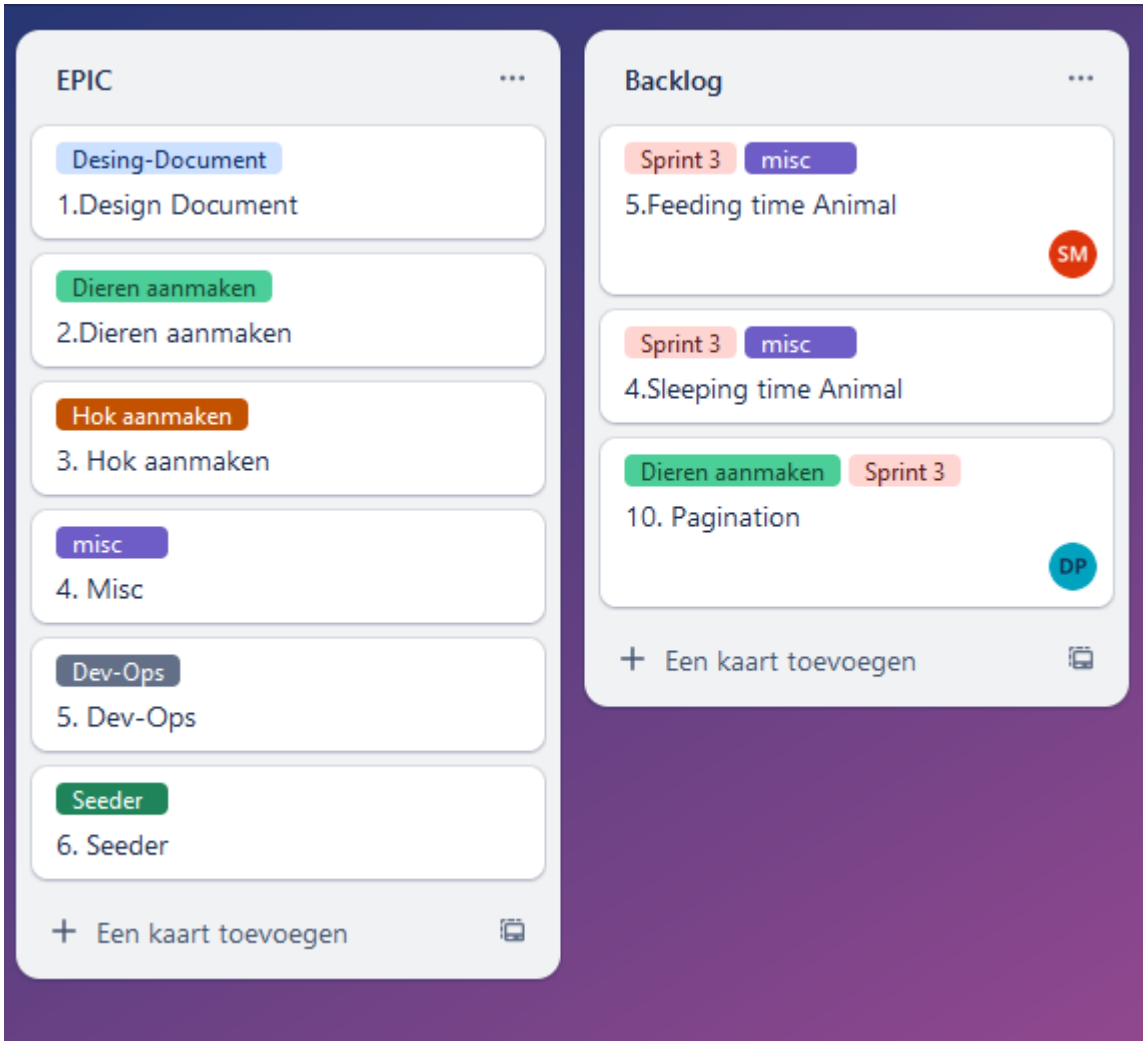
De planning , userstories, backlog hebben we uitgewerkt in **Trello**.

Voor communicatie hebben gekozen voor Discord en op school tijdens onze colleges.

Backlogbeheer:

Aan het begin van de dag gaan we in Discord, of op school onze daily stand up doen, hierbij houden we bij waar idereen mee bezig is of als ze tegen iets aan lopen.

Voorbeeld van onze Backlog:



Sprint Oplevering/ Sprintplanning:

Elke 2 weeken hebben wij een **sprintreview** waarin we bespreken wat we hebben gedaan, wat er goed ging, en wat nog aandag nodig heeft. Hierna volgt ook direct the **sprintreview** op hier bespreken we waar we tegen aan liepen en wat beter kan en hoe we dit gaan aanpakken.