

Research: Overgang van Handmatig Deployen naar Geautomatiseerd Deployen naar Docker Hub in het Buzzloop Project

SAMMOUR,REDOUAN R.

Inhoud

Inleiding	2
1. Identificeren van de behoefte aan automatisering:	3
Welke taken in het deployproces kunnen worden geautomatiseerd om efficiëntie te verbeteren en fouten te verminderen?	3
Hoe kan ik de CI/CD-pijplijn gebruiken om het deployproces te automatiseren en te stroomlijnen?	3
Welke voordelen kan ik verwachten van geautomatiseerd deployen ten opzichte van handmatig deployen naar Docker Hub?	3
2. Implementatie van Continuous Integration (CI) Tools:	4
Welke CI-tools zijn geschikt voor mijn project en hoe kan ik ze integreren in mijn ontwikkelproces?	4
Hoe configureer ik de CI/CD-pijplijn om automatisch te reageren op wijzigingen in mijn GitHub-repository?	4
Welke rol speelt GitHub Actions in het automatiseren van het deployproces?	4
3. Configuratie van de CI/CD Pipeline:	5
Hoe stel ik een CI/CD-pijplijn in die bestaat uit verschillende stappen zoals bouwen, testen en deployen?	5
Welke specifieke configuratieopties zijn beschikbaar binnen mijn CI-tool voor het bouwen en pushen van Docker-images naar Docker Hub?	5
Hoe kan ik ervoor zorgen dat de pipeline efficiënt en betrouwbaar draait en dat fouten snel worden opgelost?	6
Screenshots:	7
Conclusie:	9

Inleiding

In het kader van het Buzzloop-project is mijn primaire doelstelling de optimalisatie van het ontwikkelingsproces door de introductie van efficiënte, geautomatiseerde methoden voor het implementeren en bijwerken van de applicatie. Het overbruggen van de kloof tussen ontwikkeling en operationele uitvoering, voornamelijk het deployment naar Docker Hub, staat centraal in deze verbeteringslag. Deze research ontvouwt zich als een exploratie van de noodzakelijke transformatieve stappen richting automatisering, waarbij ik gedetailleerd de voordelen, uitdagingen en strategische overwegingen in kaart breng. Ik zal de huidige manuele processen evalueren, inzicht verschaffen in de beschikbare automatiseringstools en -technieken, en een plan ontwikkelen om de bestaande methoden te herzien. Door dit pad te volgen, beoog ik een gestroomlijnde, geautomatiseerde pipeline te implementeren die niet alleen de deployment versnelt maar ook zorgt voor een verhoogde consistentie en betrouwbaarheid van de releases die naar Docker Hub worden gepusht. Het einddoel is een robuust systeem dat wendbaar genoeg is om te voldoen aan de dynamische eisen van het hedendaagse softwareontwikkelingslandschap en dat bijdraagt aan de continue groei en succes van het Buzzloop-project.

1. Identificeren van de behoefte aan automatisering:

Welke taken in het deployproces kunnen worden geautomatiseerd om efficiëntie te verbeteren en fouten te verminderen?

Om de efficiëntie te verbeteren en fouten te verminderen, kunnen verschillende taken in het deployproces worden geautomatiseerd. Allereerst kunnen het bouwen en pushen van Docker-images naar Docker Hub worden geautomatiseerd. Dit omvat het automatisch bouwen van de applicatie op basis van de nieuwste code wijzigingen in het versiebeheersysteem, het creëren van Docker-images en het pushen van deze images naar een gedeelde repository op Docker Hub. Door dit proces te automatiseren, kunnen we de doorlooptijd verkorten en de kans op fouten verminderen die kunnen ontstaan door handmatige interventie. Een andere taak die kan worden geautomatiseerd, is het implementeren van de applicatie naar verschillende omgevingen. Met behulp van een CI/CD-pijplijn kunnen we automatisch de gebouwde Docker-images implementeren naar de gewenste omgevingen, zoals staging en productie. Dit elimineert de noodzaak voor handmatige implementatie en vermindert de kans op menselijke fouten tijdens het implementatieproces.

Hoe kan ik de CI/CD-pijplijn gebruiken om het deployproces te automatiseren en te stroomlijnen?

De CI/CD-pijplijn biedt een krachtig framework voor het automatiseren en stroomlijnen van het deployproces. Allereerst kunnen we de pijplijn configureren om te reageren op code wijzigingen in het versiebeheersysteem, zoals GitHub. Dit kan worden bereikt door triggers in te stellen die de pijplijn starten telkens wanneer er nieuwe code wordt gepusht naar de repository. Vervolgens kunnen we de stappen in de pijplijn definiëren om het deployproces te automatiseren. Dit omvat het bouwen van de applicatie, het uitvoeren van tests om de kwaliteit te waarborgen, het bouwen van Docker-images, en het implementeren van de applicatie naar de gewenste omgevingen. Door deze stappen te automatiseren, kunnen we de doorlooptijd verkorten en de betrouwbaarheid van het deployproces verbeteren.

Welke voordelen kan ik verwachten van geautomatiseerd deployen ten opzichte van handmatig deployen naar Docker Hub?

Geautomatiseerd deployen biedt verschillende voordelen ten opzichte van handmatig deployen naar Docker Hub. Ten eerste leidt automatisering tot consistente en voorspelbare implementaties. Doordat het deployproces wordt gestandaardiseerd en herhaalbaar is, minimaliseert automatisering de kans op fouten die kunnen optreden bij handmatige interventie. Dit resulteert in een verhoogde betrouwbaarheid van de implementaties en een vermindering van de downtime van de applicatie.

Bovendien maakt geautomatiseerd deployen het gemakkelijker om schaalbaarheid en flexibiliteit te bereiken. Door het implementatieproces te automatiseren, kunnen we gemakkelijk opschalen naar meerdere omgevingen en configuraties beheren met minimale inspanning. Dit stelt ons in staat om de applicatie uit te rollen in verschillende omgevingen, zoals staging en productie, en om snel te reageren op fluctuaties in de vraag.

Al met al biedt geautomatiseerd deployen aanzienlijke voordelen op het gebied van betrouwbaarheid, snelheid en schaalbaarheid, waardoor ontwikkelteams efficiënter kunnen werken en sneller waarde kunnen leveren aan klanten.

2. Implementatie van Continuous Integration (CI) Tools:

Welke CI-tools zijn geschikt voor mijn project en hoe kan ik ze integreren in mijn ontwikkelproces?

Voor mijn project zijn verschillende CI-tools geschikt, afhankelijk van de behoeften en vereisten van het project. GitHub Actions is een populaire keuze vanwege de naadloze integratie met GitHub-repositories en de mogelijkheid om workflows te definiëren met behulp van YAML-bestanden, zoals mijn CI/CD.yaml. Deze workflows kunnen verschillende taken automatiseren, zoals het bouwen van de applicatie, het uitvoeren van tests en het implementeren van updates. Daarnaast biedt Jenkins een krachtig en flexibel CI/CD-platform met een breed scala aan plugins en integratiemogelijkheden. Jenkins kan worden geconfigureerd om complexe CI/CD-pijplijnen te ondersteunen en te beheren, waardoor het geschikt is voor projecten van elke omvang.

Om een CI-tool te integreren in mijn ontwikkelproces, moet ik eerst de CI/CD-workflow definiëren op basis van de vereisten van het project. Vervolgens kan ik de CI-tool configureren om deze workflow uit te voeren bij elke wijziging in de GitHub-repository. Dit kan worden bereikt door de CI-tool te koppelen aan de GitHub-repository en triggers in te stellen om de workflow automatisch te activeren bij elke push, pull request of merge naar de repository. Ten slotte kan ik de resultaten van de CI/CD-pijplijn gebruiken om feedback te genereren en de voortgang van het ontwikkelproces te volgen.

Hoe configureer ik de CI/CD-pijplijn om automatisch te reageren op wijzigingen in mijn GitHub-repository?

Om de CI/CD-pijplijn te configureren om automatisch te reageren op wijzigingen in mijn GitHub-repository, gebruik ik de 'on'-trigger in mijn CI/CD.yaml-bestand. Hierin definieer ik dat de workflow moet worden uitgevoerd bij elke push naar de 'main'-branch van de repository. Dit zorgt ervoor dat de CI/CD-pijplijn automatisch wordt geactiveerd telkens wanneer er nieuwe code wordt toegevoegd aan de repository. Vervolgens definieer ik de stappen die moeten worden uitgevoerd in de 'jobs'-sectie van het YAML-bestand, zoals het bouwen van de applicatie, het uitvoeren van tests en het implementeren van updates. Door deze configuratie wordt de CI/CD-pijplijn automatisch gestart en uitgevoerd bij elke wijziging in de GitHub-repository.

Welke rol speelt GitHub Actions in het automatiseren van het deployproces?

GitHub Actions speelt een cruciale rol in het automatiseren van het deployproces door het faciliteren van continue integratie en continue implementatie. Met GitHub Actions kan ik workflows definiëren in YAML-bestanden, zoals mijn CI/CD.yaml, om verschillende taken automatisch uit te voeren, waaronder het bouwen van de applicatie, het uitvoeren van tests en het implementeren van updates. Deze workflows kunnen worden geconfigureerd om te reageren op triggers zoals pushes naar de GitHub-repository, pull requests of schema-evenementen. Door het gebruik van GitHub Actions kan ik het deployproces volledig automatiseren, waardoor de efficiëntie wordt verbeterd en de kans op fouten wordt verminderd.

3. Configuratie van de CI/CD Pipeline:

Hoe stel ik een CI/CD-pijplijn in die bestaat uit verschillende stappen zoals bouwen, testen en deployen?

Om een CI/CD-pijplijn in te stellen die bestaat uit verschillende stappen zoals bouwen, testen en deployen, maak ik gebruik van de mogelijkheden van mijn CI-tool, zoals GitHub Actions. Ik definieer een YAML-bestand waarin ik de verschillende stappen van de pijplijn beschrijf, zoals het bouwen van de applicatie, het uitvoeren van tests en het implementeren van updates. Elke stap wordt afzonderlijk gedefinieerd met de nodige configuratieopties en vereisten. Bijvoorbeeld, voor het bouwen van de applicatie, voer ik de nodige commando's uit om de broncode te compileren en het artefact te genereren. Voor het testen van de applicatie, voer ik tests uit om de functionaliteit en kwaliteit van de code te verifiëren. Ten slotte configureer ik de pijplijn om automatisch updates te implementeren naar de productieomgeving na succesvolle tests en validatie.

Stap 1: Definieren van de Workflow

Om een CI/CD-pijplijn op te zetten met stappen voor bouwen, testen, en deployen, begin ik met het definiëren van de workflow in een YAML-configuratiebestand zoals '**main.yaml**' voor GitHub Actions. Deze workflow specificeert automatische acties gebaseerd op GitHub-events zoals pushes naar bepaalde branches.

Stap 2: Specifieke Jobs Instellen

In het YAML-bestand verdeel ik de workflow in verschillende jobs:

- **Build Job:** Dit compileert de code, bouwt de executables, en bereidt alle artefacten voor (zoals Docker-images).
- **Test Job:** Voert unit tests, integration tests, en andere verificaties uit om de kwaliteit van de code te waarborgen.
- **Deploy Job:** Implementeert de succesvolle builds naar productie- of testomgevingen. Dit kan via directe deployment scripts of door Docker-images naar Docker Hub te pushen.

Elke job wordt uitgevoerd op triggers zoals een push naar de main branch, zoals gespecificeerd onder de on sleutel in de YAML-file.

Welke specifieke configuratieopties zijn beschikbaar binnen mijn CI-tool voor het bouwen en pushen van Docker-images naar Docker Hub?

Binnen mijn CI-tool, zoals GitHub Actions, zijn er specifieke configuratieopties beschikbaar voor het bouwen en pushen van Docker-images naar Docker Hub. In mijn CI/CD.yaml-bestand configureer ik de stappen voor het bouwen en pushen van Docker-images met behulp van de Docker CLI-opdrachten. Ik definieer een specifieke job die wordt uitgevoerd op een runner met de juiste omgeving en vereisten. Binnen deze job definieer ik de stappen voor het bouwen van de Docker-image met het '**docker build**'-commando en het taggen van de image met de juiste versie. Vervolgens gebruik ik het '**docker push**'-commando om de gebouwde image naar Docker Hub te pushen, waarbij ik de juiste referenties en autorisatietokens configureer om toegang te krijgen tot mijn Docker Hub-repository.

Stap 1: Docker Image Bouwen en Taggen

Gebruikmakend van de YAML-configuratie voor GitHub Actions, specificeer ik een step binnen een job om de Docker image te bouwen. Dit gebruik ik met de standaard Docker commands ('**docker build -t username/repository:tag**'). Hier, '**username/repository:tag**' moet worden vervangen door specifieke waarden die wijzen naar mijn Docker Hub repository en de gewenste tag.

Stap 2: Inloggen op Docker Hub

Voordat ik de image kan pushen, moet ik inloggen op Docker Hub via de CI-tool. Dit doe ik door de '**docker/login-action@v2**' GitHub Action te gebruiken, waarbij de gebruikersnaam en het wachtwoord als secrets ('**DOCKER_USERNAME**' en '**DOCKER_PASSWORD**') zijn opgeslagen in de GitHub repository settings.

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

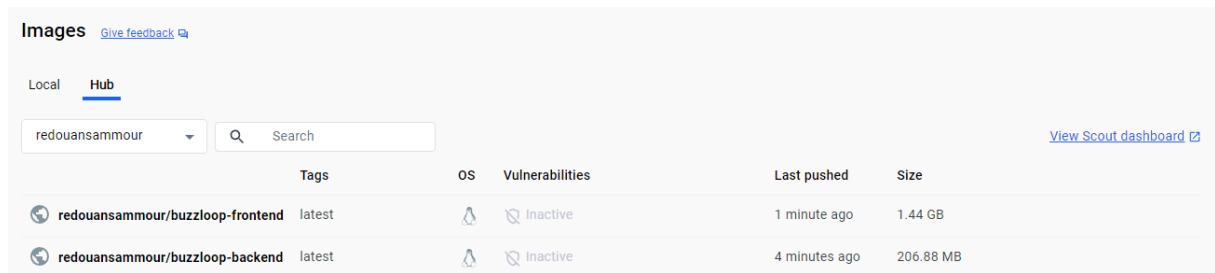
Stap 3: Pushen van de Docker Image

Na het bouwen en taggen van de image, voeg ik een stap toe om de image naar Docker Hub te pushen. Dit commando ('**docker push username/repository:tag**') zorgt ervoor dat de nieuwste image beschikbaar is op Docker Hub voor deployment.

Hoe kan ik ervoor zorgen dat de pipeline efficiënt en betrouwbaar draait en dat fouten snel worden opgelost?

Om ervoor te zorgen dat de CI/CD-pijplijn efficiënt en betrouwbaar draait en dat fouten snel worden opgelost, implementeer ik verschillende best practices en strategieën. Ten eerste voer ik regelmatig tests uit op elke stap van de pijplijn om ervoor te zorgen dat eventuele fouten of problemen snel worden gedetecteerd en opgelost. Ik configureer de pijplijn om automatisch te reageren op wijzigingen in de GitHub-repository, waardoor updates en wijzigingen snel kunnen worden geïmplementeerd en getest. Daarnaast maak ik gebruik van logging en monitoring om de voortgang van de pijplijn te volgen en eventuele fouten of problemen te identificeren. Bij het optreden van fouten ontvang ik onmiddellijk meldingen, zodat ik snel actie kan ondernemen om ze op te lossen. Tot slot voer ik regelmatig onderhoud uit aan de pijplijn om ervoor te zorgen dat deze up-to-date blijft en blijft voldoen aan de vereisten van het project.

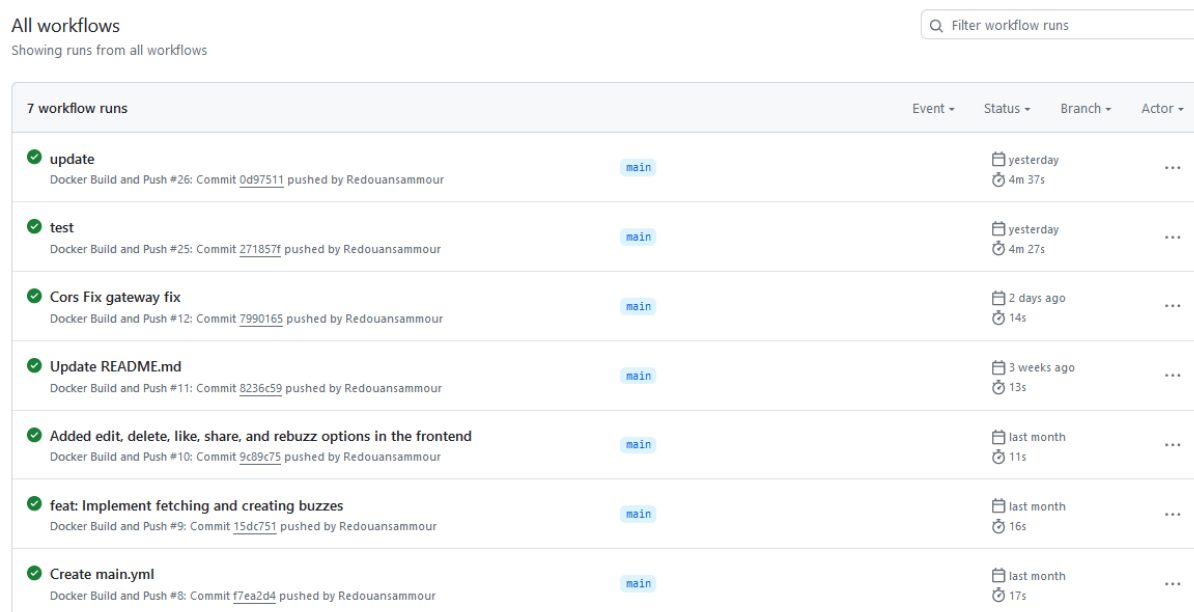
Screenshots:



The screenshot shows the Docker Hub interface for the user 'redouansammour'. It displays two Docker images: 'redouansammour/buzzloop-frontend' and 'redouansammour/buzzloop-backend'. Both images are tagged as 'latest' and were pushed recently. The 'buzzloop-frontend' image is 1.44 GB and was pushed 1 minute ago. The 'buzzloop-backend' image is 206.88 MB and was pushed 4 minutes ago. Both images show 'Inactive' vulnerabilities.

Tags	OS	Vulnerabilities	Last pushed	Size
redouansammour/buzzloop-frontend latest	linux	Inactive	1 minute ago	1.44 GB
redouansammour/buzzloop-backend latest	linux	Inactive	4 minutes ago	206.88 MB

Afbeelding 1: twee Docker Hub repositories 'buzzloop-frontend' en 'buzzloop-backend', beide recent gepusht.



The screenshot shows a list of 7 workflow runs for a GitHub Actions CI/CD process. The runs are triggered by Docker Build and Push events. The workflow runs include tasks like 'update', 'test', 'Cors Fix gateway fix', 'Update README.md', 'Added edit, delete, like, share, and rebuzz options in the frontend', 'feat: Implement fetching and creating buzzes', and 'Create main.yml'. Each run shows the commit hash, the branch (main), and the time taken to complete the run.

Event	Status	Branch	Actor
update Docker Build and Push #26: Commit 0d97511 pushed by Redouansammour	✓	main	yesterday 4m 37s
test Docker Build and Push #25: Commit 271857f pushed by Redouansammour	✓	main	yesterday 4m 27s
Cors Fix gateway fix Docker Build and Push #12: Commit 7990165 pushed by Redouansammour	✓	main	2 days ago 14s
Update README.md Docker Build and Push #11: Commit 8236c59 pushed by Redouansammour	✓	main	3 weeks ago 13s
Added edit, delete, like, share, and rebuzz options in the frontend Docker Build and Push #10: Commit 9c89c75 pushed by Redouansammour	✓	main	last month 11s
feat: Implement fetching and creating buzzes Docker Build and Push #9: Commit 15dc751 pushed by Redouansammour	✓	main	last month 16s
Create main.yml Docker Build and Push #8: Commit f7ea2d4 pushed by Redouansammour	✓	main	last month 17s

Afbeelding 2: een overzicht van workflow runs voor een GitHub Actions CI/CD-proces, met verschillende taken zoals builds en pushes


```

name: CI/CD

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK
        uses: actions/setup-java@v2
        with:
          java-version: '17'
          distribution: 'adopt'

      - name: Build with Gradle
        run: |
          ls -l # Just to make sure we're in the correct directory
          ./gradlew build --no-daemon

      - name: Check contents of build/libs folder
        run: |
          ls -l build/libs

      - name: Copy JAR files to Docker context
        run: cp build/libs/*.jar $GITHUB_WORKSPACE

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

      - name: Build Docker image
        run: |
          docker build -t redouansammour/buzzloop-spring-backend .
          docker push redouansammour/buzzloop-spring-backend:latest

```

Afbeelding 3: een GitHub Actions workflow YAML-bestand voor een CI/CD proces, geactiveerd door pushes naar de main branch, met taken voor code checkout, JDK setup, Gradle build, JAR files check, kopiëren naar Docker context, Docker login en Docker build & push.

build-and-push
succeeded 1 minute ago in 4m 20s

- > Set up job
- > Checkout repository
- > Set up JDK
- > Build with Gradle
- > Build frontend
- > Build and push backend image
- > Build and push frontend image
- > Post Set up JDK
- > Post Checkout repository
- > Complete job

Afbeelding 4: We zien een succesvol voltooide GitHub Actions workflow met taken zoals setup job, checkout repository, JDK setup, Gradle build, build en push voor zowel frontend als backend images, en afsluitende stappen zoals complete job.

Conclusie:

De transitie naar geautomatiseerde deployments met Docker Hub heeft een significante verbetering betekend voor mijn ontwikkel workflow, merkbaar door een substantiële toename in efficiëntie en een sterke afname van menselijke fouten. Het gebruik van geavanceerde CI/CD-tools heeft geleid tot een meer gestroomlijnd en responsief ontwikkel- en deploymentproces, waardoor ik snel en effectief kan inspelen op projectupdates. Dankzij de invoering van een geautomatiseerde CI/CD-pijplijn ben ik nu in staat om met grotere consistentie en betrouwbaarheid Docker-images te construeren en te distribueren naar Docker Hub. Dit verzekert niet alleen een vlottere schaalbaarheid en onderhoud van mijn applicatie, maar draagt ook bij aan een robuustere en meer agile ontwikkelomgeving.