3.1 - GitHub Actions: Kubernetes (poging 3)

Introduction

GitHub Actions also have the ability to execute commands on our local host machine, as we have explored in the previous assignment.

We are going to use this functionality to work with Kubernetes commands in order to update production-style deployments in a professional way.

Objective

Setting up GitHub Actions to deploy applications in a Docker Container in Kubernetes Pods in a scalable Deployment, with Services configured.

Extra: Implementing Helm to deploy our application all in one.

Knowledge

- GitHub Actions
- Kubernetes Deployments & Services
- Helm Charts

Skills

- Deploying Kubernetes applications on a local cluster using GitHub Actions

Necessities

- A Kubernetes Cluster as set up in
    - Docker Desktop
    - Installed `kubectl` (and accessible in your path)
    - Installed `k3d` (and accessible in your path)
    - Installed `helm` (and accessible in your path)
- A Local Runner as set up in

- A terminal
- A Git Repository on GitHub
- An IDE such as Visual Studio Code

GitHub Actions + Kubectl

First trials

GitHub Actions has a way of connecting to your local Kubernetes instance by using `kubectl`, the same interface you are using yourself and using the self-hosted GitHub Actions runner.

To test out if your local runner can access your Kubernetes cluster, you can try to install a Kubernetes object with the `kubectl` commands.

👉 If it does not work, try and give the path to your `~/.kube/config` file as a flag in all the `kubectl` commands.

1. Create a new workflow
2. Trigger it manually `workflow_dispatch` or through a `push` on the main-branch
3. Make sure it uses the `self-hosted` runner this time.
4. Add the right actions:

- The action to pull your source code
- A `kubectl` action which will apply a Kubernetes object to your cluster
- A `kubectl` action to check for the pods in your namespace

5. Find a way to generate a random name for the Kubernetes Pod object. You can find an example file in here:

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-{ID} # Try to make this dynamically generated (random)
  namespace: <your-namespace> # Enter your namespace here
  labels:
    name: mypod
spec:
  containers:
  - name: mypod
    image: rancher/hello-world
```

❓ **QUESTION 1 - Random name**
How did you make the name randomised?

ANSWER 1 - Random name

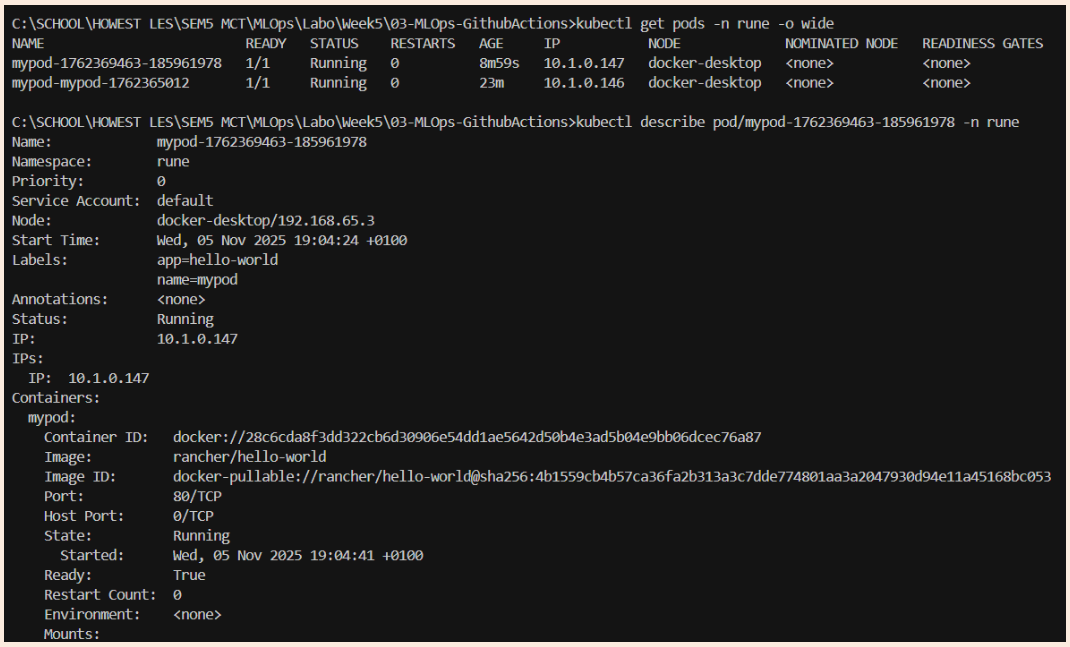
```
name: Kubernetes Pod Deployment

on:
  workflow_dispatch:  # Handmatig triggeren
  push:
    branches:
      - main

jobs:
  deploy-pod:
    runs-on: self-hosted  # Gebruik je self-hosted runner

    steps:
      - name: Check out code
        uses: actions/checkout@v4

      # Genereer een random ID voor de pod naam (Windows Powe
rshell)
      - name: Generate random ID (Windows)
        id: random-id
        shell: powershell
        run: |
          $random = Get-Random
          $timestamp = [int][double]::Parse((Get-Date -UFormat %s))
          $random_id = "$timestamp-$random"
          echo "random_id=$random_id" >> $env:GITHUB_OUTPUT
          Write-Host "Generated random ID: $random_id"

      # Vervang de placeholder in pod.yaml met random ID (Powers
hell!)
      - name: Update pod name with random ID
        shell: powershell
        run: |
          $file = "kubernetes-pod.yaml"
          $content = Get-Content $file
          $random_id = "${{ steps.random-id.outputs.random_id }}"
          $content = $content -replace "PLACEHOLDER", $random_i
d
          Set-Content $file $content
          Get-Content $file

      # Deploy de pod naar Kubernetes
      - name: Apply Kubernetes Pod
        run: |
          kubectl apply -f kubernetes-pod.yaml

      # Controleer de pods in je namespace
      - name: Check pods in namespace
        run: |
          kubectl get pods -n rune
```

Uitleg van de randomisatie:

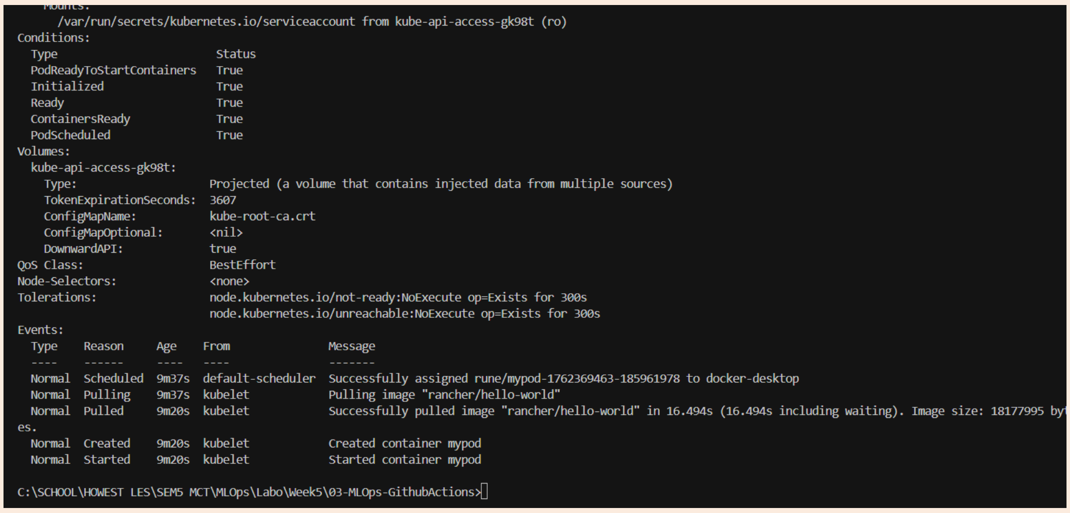
- `date +%s` geeft de huidige Unix timestamp (aantal seconden sinds 1 januari 1970)
- `$RANDOM` geeft een random nummer tussen 0 en 32767
- Door ze te combineren krijg je een unieke naam zoals `mypod-1730067890-12345`
- `sed -i` vervangt "PLACEHOLDER" in het bestand met de random ID
- De pod naam wordt zo bijvoorbeeld `mypod-1730067890-12345`

❓ **QUESTION 2 - Screenshot & logs**
Show the result of the pod running in the terminal.

ANSWER 2 - Screenshot & logs



```
C:\SCHOOL\HOWEST LES\SEM5 MCT\MLOps\Labo\Week5\03-MLOps-GithubActions>kubectl get pods -n rune -o wide
NAME                         READY   STATUS    RESTARTS   AGE     IP           NODE             NOMINATED NODE   READINESS GATES
mypod-1762369463-185961978   1/1     Running   0          8m59s   10.1.0.147   docker-desktop   <none>           <none>
mypod-mypod-1762365012       1/1     Running   0          23m     10.1.0.146   docker-desktop   <none>           <none>

C:\SCHOOL\HOWEST LES\SEM5 MCT\MLOps\Labo\Week5\03-MLOps-GithubActions>kubectl describe pod/mypod-1762369463-185961978 -n rune
Name:             mypod-1762369463-185961978
Namespace:        rune
Priority:         0
Service Account:  default
Node:             docker-desktop/192.168.65.3
Start Time:       Wed, 05 Nov 2025 19:04:24 +0100
Labels:           app=hello-world
                  name=mypod
Annotations:      <none>
Status:           Running
IP:               10.1.0.147
IPs:
  IP:  10.1.0.147
Containers:
  mypod:
    Container ID:   docker://28c6cda8f3dd322cb6d30906e54dd1ae5642d50b4e3ad5b04e9bb06dcec76a87
    Image:          rancher/hello-world
    Image ID:       docker-pullable://rancher/hello-world@sha256:4b1559cb4b57ca36fa2b313a3c7dde774801aa3a2047930d94e11a45168bc053
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 05 Nov 2025 19:04:41 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
```



```
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-gk98t (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-gk98t:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  9m37s  default-scheduler  Successfully assigned rune/mypod-1762369463-185961978 to docker-desktop
  Normal  Pulling    9m37s  kubelet            Pulling image "rancher/hello-world"
  Normal  Pulled     9m20s  kubelet            Successfully pulled image "rancher/hello-world" in 16.494s (16.494s including waiting). Image size: 18177995 byt
es.
  Normal  Created    9m20s  kubelet            Created container mypod
  Normal  Started    9m20s  kubelet            Started container mypod

C:\SCHOOL\HOWEST LES\SEM5 MCT\MLOps\Labo\Week5\03-MLOps-GithubActions>
```

Deployments

Similar to how a Pod can be deployed in `kubectl`, we can also use a Deployment. Set this up by creating a deployment and service with the necessary settings, such as NodePort to open up the application to the host machine.

However, when using a deployment, we are usually having a new Docker image to deploy. This image can be automated in the pipeline as we will do later. But when you're choosing to use the same image name every time, you'll need to restart your deployment so it gets the new image update.

Search for the command to rollout a new update using the same image.

❓ **QUESTION 3 - Deployment rollout**
Which command did you use for the deployment rollout?

💻 **ANSWER 3 - Deployment rollout**

```
# GitHub Actions snippet
- name: Restart deployment to pull latest image
  run: |
    kubectl rollout restart deployment/hello-world-deployment -n r
une
Uitleg: Het kubectl rollout restart commando forceert Kubernetes
om alle
pods in de deployment opnieuw te starten. Dit is nodig wanneer j
e
dezelfde image tag gebruikt (zoals :latest), maar de image inhoud
is
veranderd. Kubernetes detecteert dit normaal niet omdat de tag h
etzelfde
blijft, dus moet je expliciet een rollout doen.
```

Getting Helm involved

As we know, Helm is great to do automations of the Kubernetes installations. We simply need to enter a command like `helm upgrade --install ...` with all the necessary parameters to install a chart into our cluster.

The useful thing to combine with GitHub Actions is that we can override the `values.yaml` file from the command line as well.

We can even use a special Action / Task that is specialised in editing YAML files. That way we can easily specify a new version of the Docker image.

Try to install a Helm chart into your Kubernetes cluster, with the right GitHub Actions that you can confirm. You can use the same logic as you used above for the Kubernetes part.

❓ **QUESTION 4 - Helm install**
How does the command for a Helm install look like with the flags added ?

💻 **ANSWER 4 - Helm install**

```
helm upgrade --install hello-world-nginx ./helm-chart \
  --namespace rune \
  --create-namespace \
  --set image.tag=latest \
  --wait \
  --timeout 2m
```

Uitleg van de flags:

- `-install` : Installeert als de release nog niet bestaat (combineert upgrade en install)
- `-namespace rune` : Specificeert de Kubernetes namespace
- `-create-namespace` : Maakt de namespace aan als deze niet bestaat
- `-set image.tag=latest` : Override de image tag waarde uit values.yaml
- `-wait` : Wacht tot alle resources in een ready state zijn
- `-timeout 2m` : Maximale wachttijd van 2 minuten

Complete automation

This quick checklist / manual can help you in figuring out a good structure for the GitHub Actions.

1. Clone the GitHub repository
2. Optional: Perform some automated checks on the code if needed (Unit Test, Integration Test, Cyclomatic Complexity, Code Coverage …)

3. Build the Docker Image with a new tag (`:latest`, `:<git-sha>`). We like to refer to the Git-SHA as it's always unique and refers back to the Git commit we performed.
4. Push the Docker Image (to GHCR or Docker Hub).
5. Deploy using plain Kubernetes (without Helm) or with Helm

Kubernetes without Helm

1. Create a Kubernetes namespace, if needed
2. Change the Kubernetes YAML files
3. Apply the Kubernetes YAML files
4. Test the final result

Kubernetes with Helm

1. Create a Kubernetes namespace, if needed
2. Override the `values.yaml` file using a command line interface flag or with a specialised task.
3. Upgrade (or install) the Helm chart
4. Test the final result

A few notes

A few things to note using the automation with GitHub Actions:

- Make sure it works when you have just installed a clean Kubernetes cluster: Cold start
  - [x] ~~Create namespace **If doesn't exist**~~
  - [x] ~~Change properties if needed, or use default properties~~
  - [x] ~~Update **OR install**. Don't forget that your `:latest` tag will not update the Kubernetes cluster, as it will only have an effect if the Kubernetes' **Desired State** is different than it was before. That's why we prefer to use **Semantic Versioning** for our Docker Image tags. If you want to overcome that issue, you can use the `ImagePullPolicy: Always` property in your Kubernetes deployment container specs.~~
- Don't have any secrets into your GitHub repository
- See if your GitHub Actions runner is active
- Try automated tests in your project

What did you learn?

Fill in something that you learned during this lesson

- Hoe je GitHub Actions integreert met Kubernetes voor geautomatiseerde deployments
- Het verschil tussen Pods en Deployments in Kubernetes
- Hoe je dynamisch pod namen genereert met timestamps en random IDs
- Het gebruik van kubectl rollout restart voor het updaten van deployments
- Hoe Helm charts werken en hoe je waarden overschrijft met --set flags
- Het belang van imagePullPolicy: Always bij het gebruik van :latest tags
- Hoe je namespaces aanmaakt en beheert in Kubernetes
- Het verschil tussen declaratieve (Helm/YAML) en imperatieve (kubectl commands) deployment strategieën
- Hoe je een complete CI/CD pipeline opzet: build → push → deploy → verify

Give three interesting exam questions

1. Leg uit waarom een Kubernetes Deployment met :latest tag niet automatisch update wanneer je een nieuwe Docker image pusht met dezelfde tag. Welke twee oplossingen zijn er voor dit probleem?
2. Wat is het verschil tussen "helm install" en "helm upgrade --install"? In welk scenario zou je voor de tweede optie kiezen en waarom?
3. Beschrijf de volledige flow van een rolling update in Kubernetes. Wat gebeurt er stap voor stap wanneer je "kubectl rollout restart deployment/my-app" uitvoert? Waarom zorgt dit voor zero-downtime?

Handing in this assignment

You can hand this in by duplicating this document on Notion, print this document as a `.pdf` and submit that document on Leho.

Also hand in the written Source Code in a `.zip` file please.

Checkboxes:

- [x] ~~I have duplicated this file~~

- [x] ~~I have filled in all the answers~~
- [x] ~~I have added something that I learned~~
- [x] ~~I have added three interesting exam questions~~
- [x] ~~I have zipped my project and uploaded it to Leho~~