# EKS workshop

## Basis instellingen en tools

- AWS CLI
- Kubectl / kubectx
- Helm 3
- Linux CLI
- K9s

- Groepnummer

Benodigdheden:

- Open de README file en
  - Voeg dit code block toe aan je **.aws/credentials** file

```
[odisee]
aws_access_key_id=AKIAYJY32O2VDZPUIPV4
aws_secret_access_key=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
region=eu-west-1
```

  - Voer deze commando's uit op de cli (vervang <groepnummer door jouw eigen nummer>)

```
export AWS_PROFILE=odisee
aws eks update-kubeconfig --name workshop-cluster --region eu-west-1
kubectl create namespace odisee-<groepnummer>
```

## Helm intro

## Basis Helm chart

Download de initiële versie van de files uit deze repo
https://github.com/SentiaBE/sentia_eks_workshop

In deze repo kan je een **helmbase** map vinden, vertrek hiervan.

## Opdracht 1

De kracht van een Helm chart is dat je gebruik kan maken van variabelen, maar dat je deze variabelen ook kan overschrijven via de cli. Open alvast even de deployment file die in de templates map te vinden is.

Zoals jullie gewoon zijn kan je in de deployment file deze waarden in plain tekst zetten. Het grote nadeel hieraan is dat je deze niet kan overschrijven wat de automatisatie moeilijker maakt.

```
spec:
  containers:
  - image: nginx:latest
    imagePullPolicy: Always
```

Een Helm chart deployen / updaten kan als volgt:
**helm upgrade --install <deployment_naam> . -n <namespace>**

Voor het gebruik van variabelen, maak je gebruik van de values.yaml file.
Probeer nu de variabelen in de deployment file aan te passen naar deze variabelen:

```yaml
image:
  repository: nginx
  tag: latest
  pullPolicy: Always
```

**Hint : Helm is achterliggend gebaseerd op Golang**

**Kan je de helm chart terug deployen?**
**Zie je iets veranderen?**
Je kan dit nakijken door kubectl get pods -n <namespace>, kijk zeker even naar de **AGE kolom**

## Opdracht 2

Momenteel hangt er een ingress aan de Helm chart, maar de webserver is momenteel niet publiek beschikbaar. Dit is een mogelijke output als we de ingress status opvragen adhv het volgende commando **kubectl get ingress -n <namespace>**

```
NAME          CLASS    HOSTS   ADDRESS                                                                  PORTS   AGE
web-ingress   <none>   *       internal-k8s-odisee-webingre-74a6806fa9-1003302405.eu-central-1.elb.amazonaws.com   80      2m33s
```

Standaard maakt de ingress een interne loadbalancer aan die via internet niet te bereiken is. Dit is vrij eenvoudig aan te passen in de **ingress file** door een **annotation** toe te voegen. Achterliggend maken we gebruik van de AWS Load Balancer Controller.
**Verander de interne loadbalancer naar een public facing loadbalancer en probeer de site te bereiken.**

Voer het commando **kubectl get ingress -n <namespace>** nogmaals uit om na te gaan of de loadbalancer aangepast is.

**Belangrijk om te onthouden, het opzetten van een ALB binnen AWS kan enkele minuten duren.**

## Opdracht 3

Als alles goed is zal je nu de Nginx default pagina zien, echter is deze enkel via http te bereiken. De volgende opdracht bestaat uit 3 grote delen

- Het linken van een domeinnaam aan de loadbalancer (odisee-<groepnummer>.training.sentiaws.cloud)
- Het koppelen van een certificaat aan de loadbalancer
- Het redirecten van al het HTTP verkeer naar HTTPS verkeer

Daarnaast gaan we ook een domeinnaam koppelen aan de ingress en een bijhorend certificaat. Gebruik de certificate ARN uit de benodigdheden en domeinnaam **odisee-<groepnummer>.training.sentiaws.cloud**

We gebruiken externalDNS, pas nu de ingress file aan. (we maken gebruik van de ALB Ingress controller)

**Probeer nu of je op de nieuwe hostname de test pagina kan bereiken. (dit kan even duren voor de DNS records aangemaakt zijn)**

Pas hierna de ingress file terug aan om al het HTTP verkeer te redirecten naar HTTPS en om een certificaat toe te voegen met de bovenstaande ARN. (gebruik hier de volledige arn te beginnen met **arn:aws:acm**)

**Surf nu naar je persoonlijke url, is de site beveiligd met een ssl certificaat?**
**Werkt de http site nog?**

## Opdracht 4

Als volgende stap is het de bedoeling dat je een extra deployment gaat toevoegen waarbij de echo-server (https://hub.docker.com/r/ealen/echo-server) gedeployed zal worden. Deze echo-server zal beschikbaar worden op jullie url onder pad **/echo**, de bedoeling hiervan wordt dat alle requests naar de nginx pod gaan met uitzondering van het /echo pad.

Om dit tot een goed einde te brengen moeten volgende bestanden aangepast worden:

- Ingress file
- Service file (of nieuwe service file)
- Deployment file (of nieuwe deployment file)

Dit artikel geeft een goed overzicht hoe een ingress / service en deployment samenwerken.

## Opdracht 5

Deze opdracht bestaat eruit om een health check op de ingress toe te voegen zodat de loadbalancer een specifiek pad gaat testen. Hierna voegen we een gelijkaardige test toe op de deployment zodat deze pas gebruikt kan worden als deze opgestart is.

Start met het toevoegen van de health check op de ingress, de documentatie van de AWS Load Balancer Controller kan je alvast verder helpen. Het healthcheck pad mag je aanpassen naar **/**, de healthcheck poort is **80**, gezien de containers op poort 80 luisteren.

Om ervoor te zorgen dat een container niet te snel in gebruik wordt genomen als deze nog niet klaar is kan gebruik gemaakt worden van een **readiness probe**. Deze gaat een tcp connectie doen naar de container om te zien of deze live is, van zodra een HTTP code 200 terug komt, pas dan zal deze in gebruik genomen worden.

Om na te kijken of de container nog steeds in een healthy state is, kan gebruik gemaakt worden van een **liveness probe**.

**Configureer nu de readiness probe en de liveness probe voor de beide deployments.**
**Deploy hierna je helm chart en verwijder als test een container.**

## Opdracht 6

Bij het deployen van de huidige Helm chart kan er downtime optreden omdat er bezoekers hun connectie gaan verliezen bij het vernieuwen van de containers. Om dit te verhinderen zijn er binnen de deployment meerdere deployment strategies die gebruikt kunnen worden.

Meer info kan je onder andere vinden op de website.

Bekijk de verschillende opties en kies voor de strategie met de minste impact voor de eindgebruiker.

Implementeer deze strategie en pas dan de versie van de nginx container aan, merk je tijdens het deployen downtime? Wat zie je precies gebeuren?

## Opdracht 7

Voor deze laatste opdracht moeten er limits en requests gezet worden op de deployments. Dit is nodig om de load op de cluster binnen de perken te houden.
Indien we geen limits en requests zouden zetten dan kan elke container onbeperkt gebruik maken van de resources in de cluster. Meestal heeft dit als gevolg dat er andere applicaties in de cluster hiervan impact ondervinden.

Zet nu op de 2 deployments voor zowel memory als cpu een limit en een request.

Meer info over limits en requests kan je vinden op de Kubernetes site.