Lab 5 – Containers and Kubernetes

Basics:

- Introduction to containers, e.g. Docker: https://docs.docker.com/get-started/
    - Lightweight virtualization
    - Key concepts:
        - Dockerfile, Image, Container, Image repositories
- Kubernetes, container orchestration: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
    - Cluster, Pods, Services etc.
    - Used to manage a cluster of hosts/containers
    - Why do we need orchestration? What functionalities are provided by k8s, which exceed those of a simple dockerd daemon?
- Alternative implementations:
    - Containers: apptainer, podman, lxc
    - Container orchestration: OpenShift, Rancher, Nomad, commercial services: e.g. AWS ECS, Azure AKS, Google GKE

Prerequisites

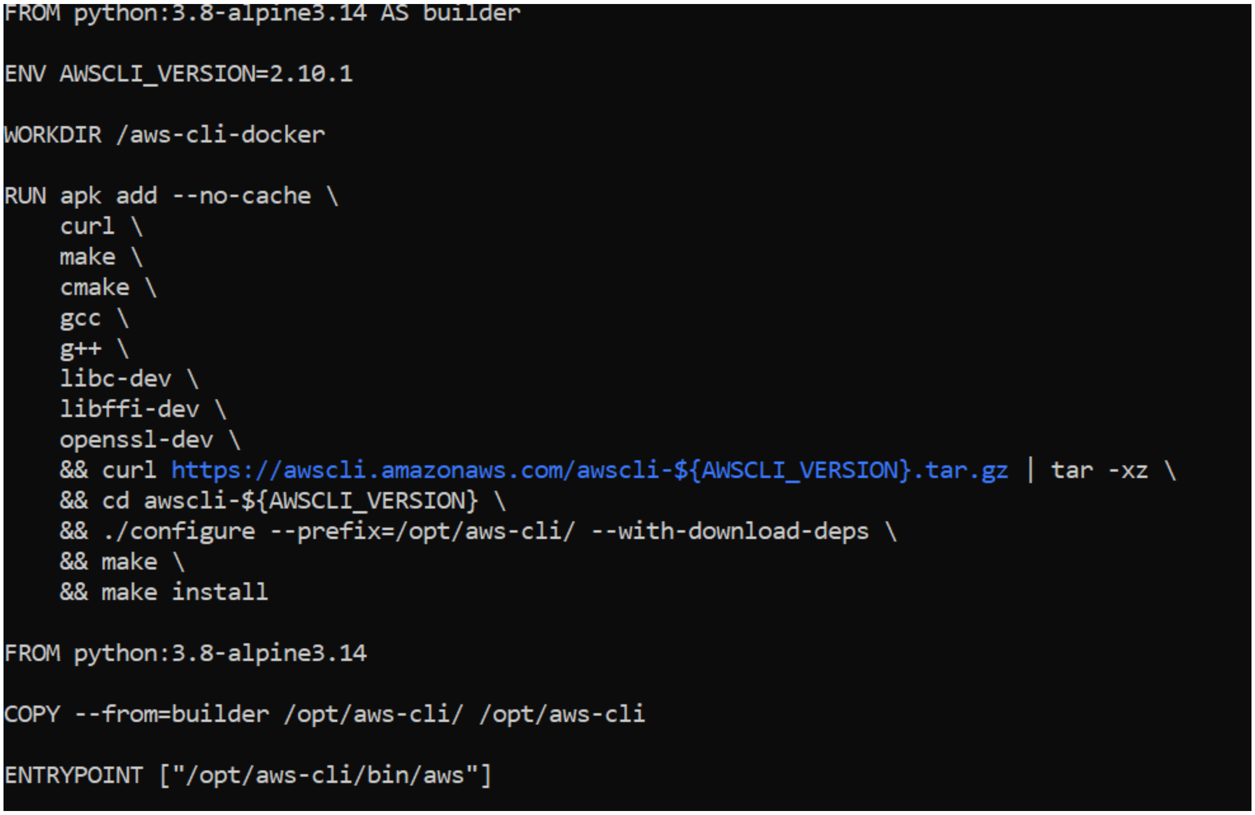
- Preferably a Linux environment (Linux VM, or Windows/WSL 2)
- Docker
- Kubernetes cluster. While installing a full-fledged cluster is quite complicated, it's possible to prepare a development "mini k8s cluster" with the help of
    - Minikube: https://minikube.sigs.k8s.io/docs/start (simple, only 1-node cluster)
    - Kind: https://kind.sigs.k8s.io (Kubernetes in Docker, possible to emulate multiple nodes on a single machine)
    - Or others…

Assignments

Assignments were performed in **WSL** in the **Ubuntu** operating system

1) Dockerize AWS-CLI (3p)
    a) Create a Dockerfile with aws-cli, built from source files (using tar.gz file / make).
        - Use a multi-stage build to create a small image. Also use a small base image, e.g., based on Alpine Linux.
        - Read more on Python and Docker multistage build.
        - Follow best practices for writing Dockerfiles. In particular read about ADD or COPY.

nano Dockerfile



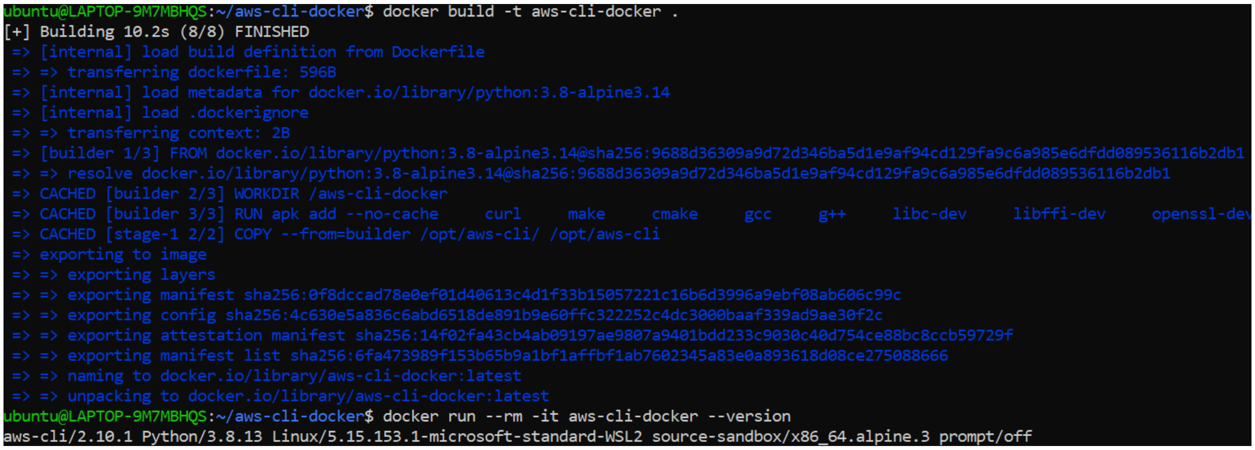
```
FROM python:3.8-alpine3.14 AS builder

ENV AWSCLI_VERSION=2.10.1

WORKDIR /aws-cli-docker

RUN apk add --no-cache \
    curl \
    make \
    cmake \
    gcc \
    g++ \
    libc-dev \
    libffi-dev \
    openssl-dev \
    && curl https://awscli.amazonaws.com/awscli-${AWSCLI_VERSION}.tar.gz | tar -xz \
    && cd awscli-${AWSCLI_VERSION} \
    && ./configure --prefix=/opt/aws-cli/ --with-download-deps \
    && make \
    && make install

FROM python:3.8-alpine3.14

COPY --from=builder /opt/aws-cli/ /opt/aws-cli

ENTRYPOINT ["/opt/aws-cli/bin/aws"]
```

My main example for creating the image was the AWS instructions. To improve the Dockerfile, I concretized the Alpine version and removed the groff installation

b) Build an image based on your Dockerfile and test it.

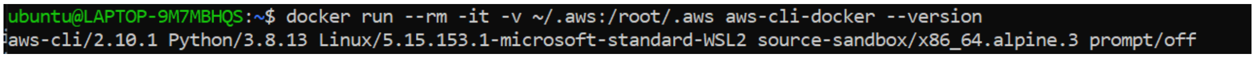
- Use a volume (-v option) to make AWS credentials available in the container.

Running image locally



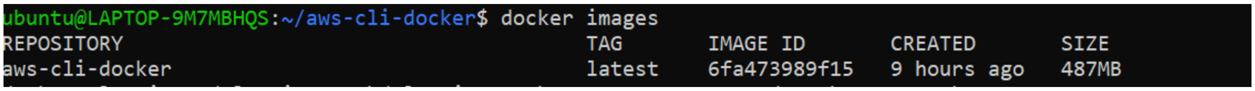
```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ docker build -t aws-cli-docker .
[+] Building 10.2s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 596B
 => [internal] load metadata for docker.io/library/python:3.8-alpine3.14
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [builder 1/3] FROM docker.io/library/python:3.8-alpine3.14@sha256:9688d36309a9d72d346ba5d1e9af94cd129fa9c6a985e6dfdd089536116b2db1
 => => resolve docker.io/library/python:3.8-alpine3.14@sha256:9688d36309a9d72d346ba5d1e9af94cd129fa9c6a985e6dfdd089536116b2db1
 => CACHED [builder 2/3] WORKDIR /aws-cli-docker
 => CACHED [builder 3/3] RUN apk add --no-cache     curl     make     cmake     gcc     g++     libc-dev     libffi-dev     openssl-dev
 => CACHED [stage-1 2/2] COPY --from=builder /opt/aws-cli/ /opt/aws-cli
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:0f8dccad78e0ef01d40613c4d1f33b15057221c16b6d3996a9ebf08ab606c99c
 => => exporting config sha256:4c630e5a836c6abd6518de891b9e60ffc322252c4dc3000baaf339ad9ae30f2c
 => => exporting attestation manifest sha256:14f02fa43cb4ab09197ae9807a9401bdd233c9030c40d754ce88bc8ccb59729f
 => => exporting manifest list sha256:6fa473989f153b65b9a1bf1affbf1ab7602345a83e0a893618d08ce275088666
 => => naming to docker.io/library/aws-cli-docker:latest
 => => unpacking to docker.io/library/aws-cli-docker:latest
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ docker run --rm -it aws-cli-docker --version
aws-cli/2.10.1 Python/3.8.13 Linux/5.15.153.1-microsoft-standard-WSL2 source-sandbox/x86_64.alpine.3 prompt/off
```

Checking buckets on AWS with container



```
ubuntu@LAPTOP-9M7MBHQS:~$ docker run --rm -it -v ~/.aws:/root/.aws aws-cli-docker --version
aws-cli/2.10.1 Python/3.8.13 Linux/5.15.153.1-microsoft-standard-WSL2 source-sandbox/x86_64.alpine.3 prompt/off
```

Size of the container image

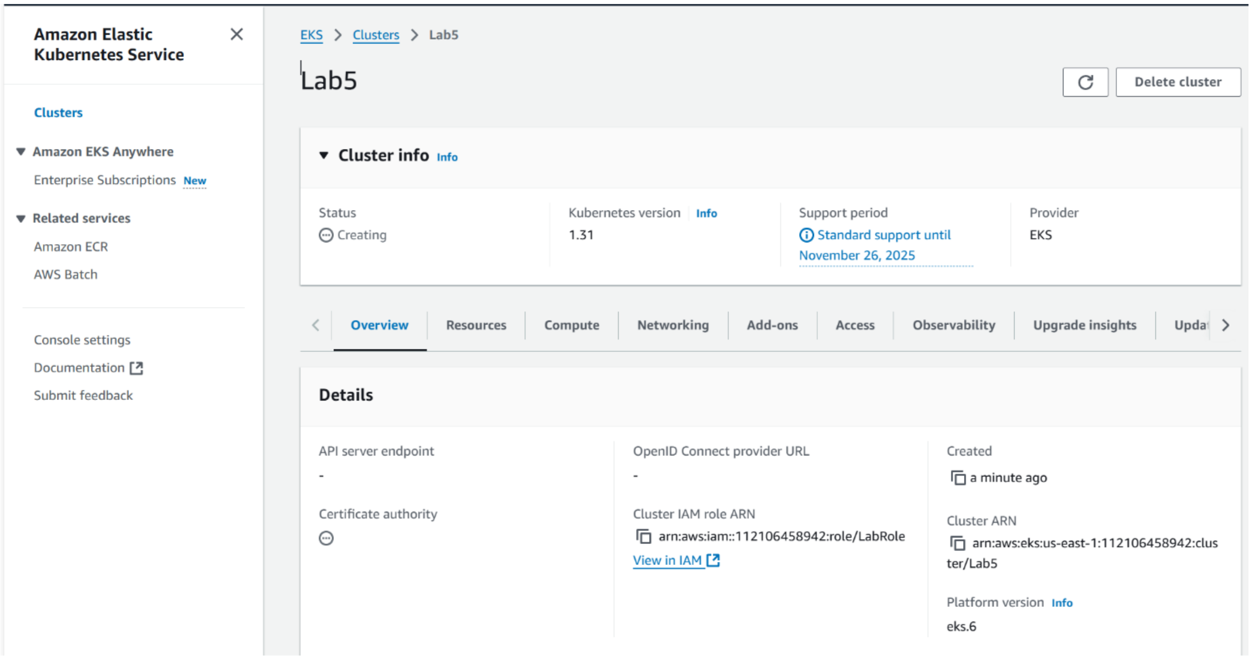


```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ docker images
REPOSITORY                                        TAG       IMAGE ID       CREATED        SIZE
aws-cli-docker                                    latest    6fa473989f15   9 hours ago    487MB
```

2) Kubernetes deployment (5p)
   a) Create a k8s cluster using Amazon Elastic Kubernetes Service (EKS)
      - You can also use minikube, kind, or any other Kubernetes distribution, or existing cluster.
      - Minikube, by default, uses its own internal Docker daemon. This daemon doesn’t know anything about images built previously. Prepare your environment by directing it to access the internal docker daemon by using the $(minikube docker-env) command and rebuild your images. This way images will be available within the k8s cluster. (https://medium.com/bb-tutorials-and-thoughts/how-to-use-own-local-doker-images-with-minikube-2c1ed0b0968)

My main example was the AWS instructions for creating cluster with AWS console

Cluster



Lab5

Cluster info Info

Status: Creating

Kubernetes version Info: 1.31

Support period: Standard support until November 26, 2025

Provider: EKS

Details

API server endpoint: -

OpenID Connect provider URL: -

Created: a minute ago

Certificate authority

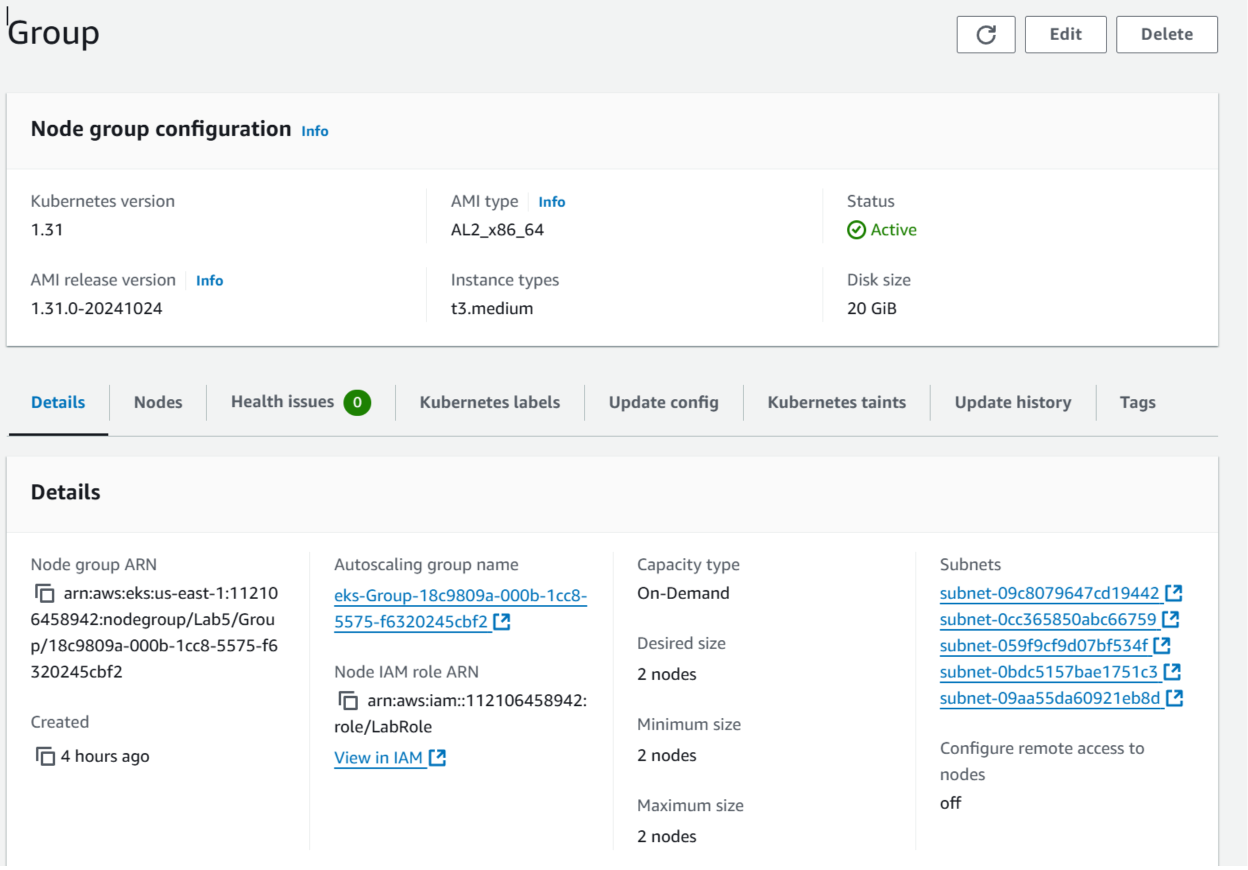
Cluster IAM role ARN: arn:aws:iam::112106458942:role/LabRole

View in IAM

Cluster ARN: arn:aws:eks:us-east-1:112106458942:cluster/Lab5

Platform version Info: eks.6

Nodes group



Group

Node group configuration Info

Kubernetes version
1.31

AMI type | Info
AL2_x86_64

Status
Active

AMI release version | Info
1.31.0-20241024

Instance types
t3.medium

Disk size
20 GiB

Details

Node group ARN
arn:aws:eks:us-east-1:112106458942:nodegroup/Lab5/Group/18c9809a-000b-1cc8-5575-f6320245cbf2

Created
4 hours ago

Autoscaling group name
eks-Group-18c9809a-000b-1cc8-5575-f6320245cbf2

Node IAM role ARN
arn:aws:iam::112106458942:role/LabRole
View in IAM

Capacity type
On-Demand

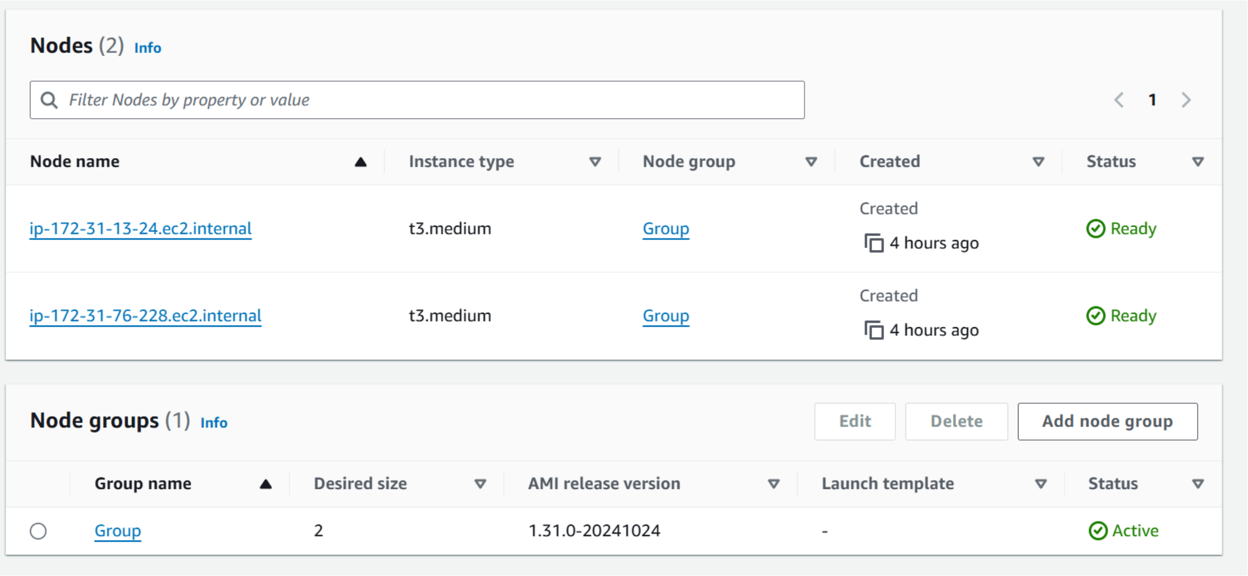
Desired size
2 nodes

Minimum size
2 nodes

Maximum size
2 nodes

Subnets
subnet-09c8079647cd19442
subnet-0cc365850abc66759
subnet-059f9cf9d07bf534f
subnet-0bdc5157bae1751c3
subnet-09aa55da60921eb8d

Configure remote access to nodes
off



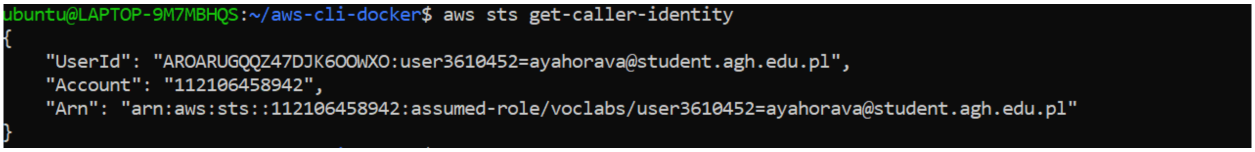
Nodes (2) Info

Node name	Instance type	Node group	Created	Status
ip-172-31-13-24.ec2.internal	t3.medium	Group	Created 4 hours ago	Ready
ip-172-31-76-228.ec2.internal	t3.medium	Group	Created 4 hours ago	Ready

Node groups (1) Info

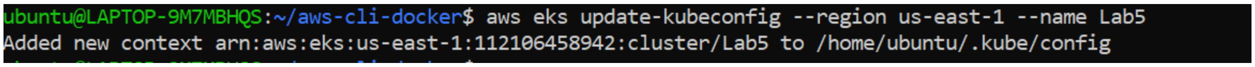
	Group name	Desired size	AMI release version	Launch template	Status
○	Group	2	1.31.0-20241024	-	Active

Cluster authentication



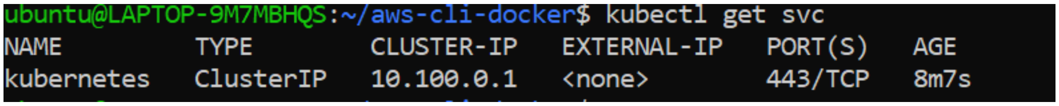
```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ aws sts get-caller-identity
{
    "UserId": "AROARUGQQZ47DJK6OOWXO:user3610452=ayahorava@student.agh.edu.pl",
    "Account": "112106458942",
    "Arn": "arn:aws:sts::112106458942:assumed-role/voclabs/user3610452=ayahorava@student.agh.edu.pl"
}
```

Create a kubeconfig file for the cluster



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ aws eks update-kubeconfig --region us-east-1 --name Lab5
Added new context arn:aws:eks:us-east-1:112106458942:cluster/Lab5 to /home/ubuntu/.kube/config
```

Testing configuration

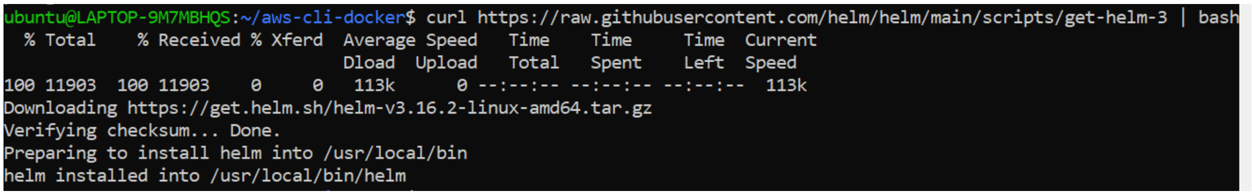


```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   8m7s
```

b) Using Helm, install an NFS server and provisioner in the cluster.

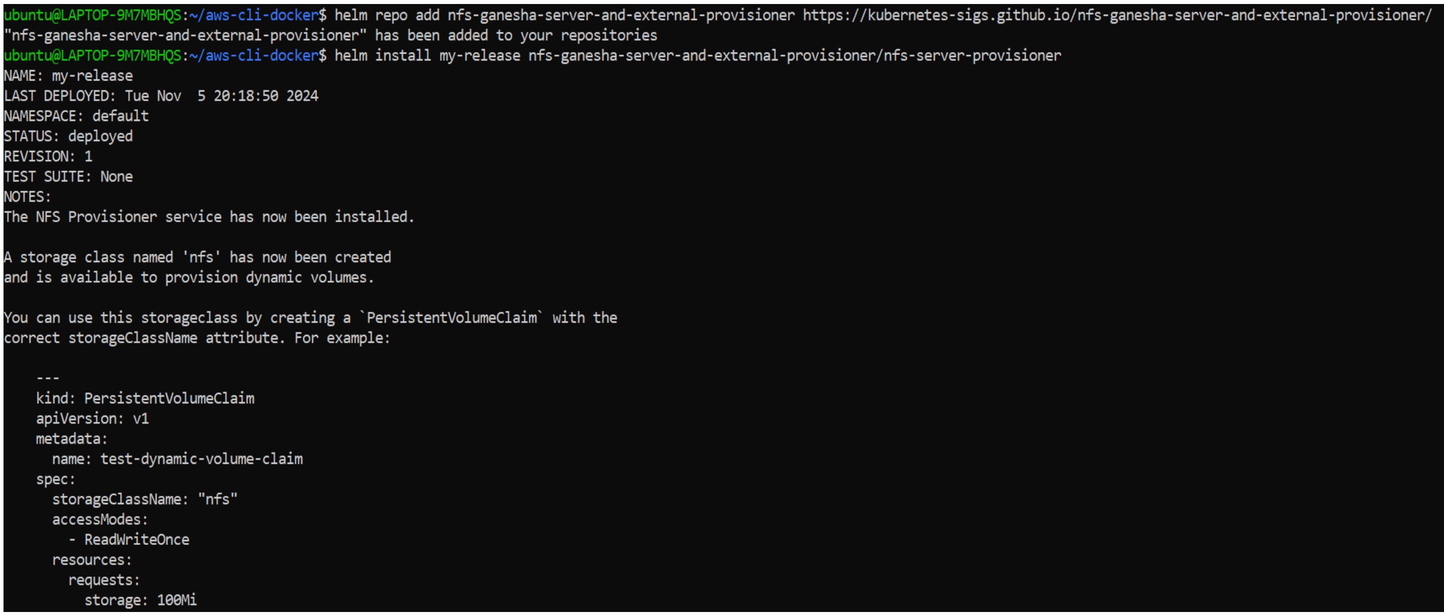
- Go to charts/nfs-server-provisioner for a README.
- Pay attention to configuration parameters, in particular, override `storageClass.name` which denotes the name of the StorageClass that you'll have to use when creating Persistent Volume Claims.

Installing Helm



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11903  100 11903    0     0   113k      0 --:--:-- --:--:-- --:--:--  113k
Downloading https://get.helm.sh/helm-v3.16.2-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

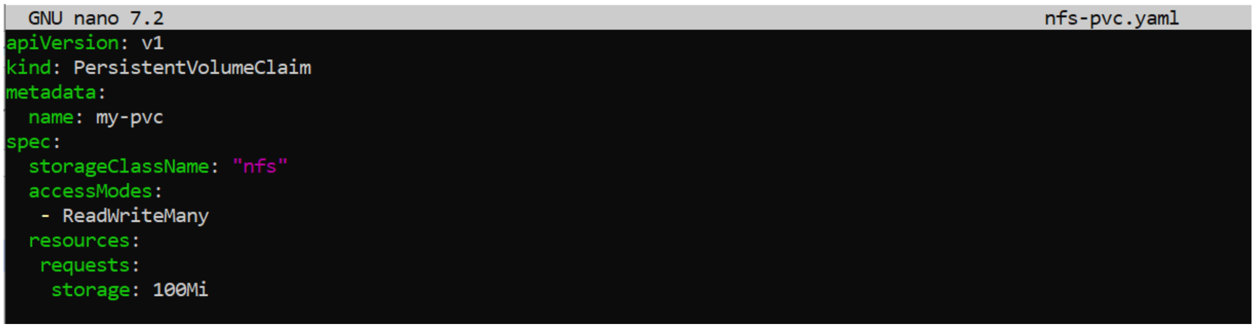
Installing nfs



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ helm repo add nfs-ganesha-server-and-external-provisioner https://kubernetes-sigs.github.io/nfs-ganesha-server-and-external-provisioner/
"nfs-ganesha-server-and-external-provisioner" has been added to your repositories
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ helm install my-release nfs-ganesha-server-and-external-provisioner/nfs-server-provisioner
NAME: my-release
LAST DEPLOYED: Tue Nov  5 20:18:50 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The NFS Provisioner service has now been installed.

A storage class named 'nfs' has now been created
and is available to provision dynamic volumes.

You can use this storageclass by creating a `PersistentVolumeClaim` with the
correct storageClassName attribute. For example:

    ---
    kind: PersistentVolumeClaim
    apiVersion: v1
    metadata:
      name: test-dynamic-volume-claim
    spec:
      storageClassName: "nfs"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 100Mi
```

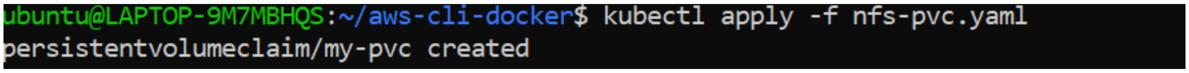
c) Create a Persistent Volume Claim which will bind to a NFS Persistent Volume provisioned dynamically by the provisioner installed in the previous step.

nano nfs-pvc.yaml



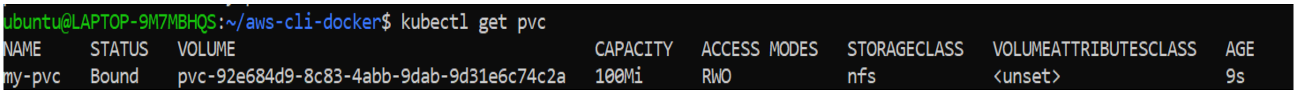
```
  GNU nano 7.2                                                                  nfs-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: "nfs"
  accessModes:
   - ReadWriteMany
  resources:
   requests:
    storage: 100Mi
```

Running nfs-pvc.yaml



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl apply -f nfs-pvc.yaml
persistentvolumeclaim/my-pvc created
```

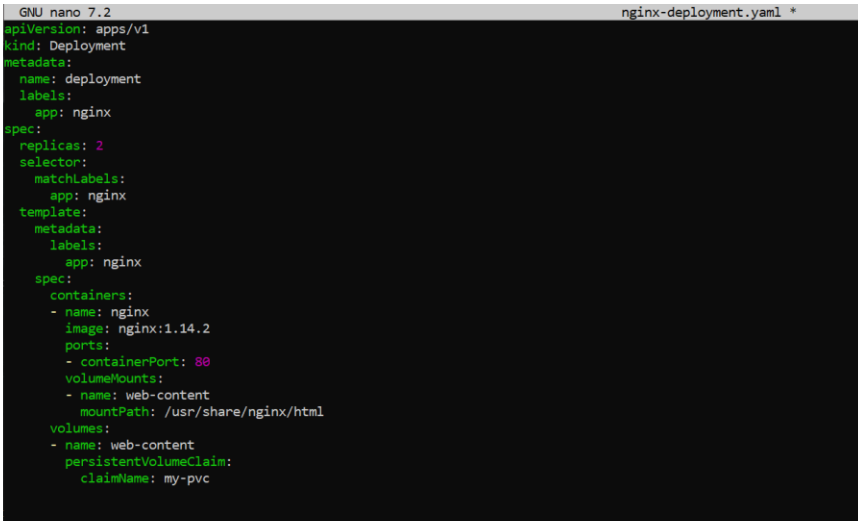
Checking pvc



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get pvc
NAME     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
my-pvc   Bound    pvc-92e684d9-8c83-4abb-9dab-9d31e6c74c2a   100Mi      RWO            nfs            <unset>                 9s
```

d) Create a Deployment with a HTTP server (e.g., apache or nginx). The web content directory should be mounted as a volume using the PVC created in the previous step.

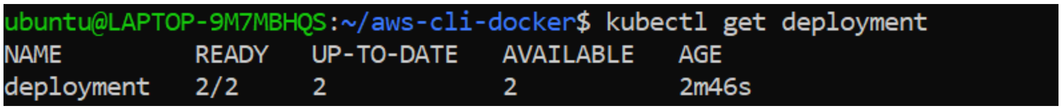
nano nginx-deployment.yaml



```
  GNU nano 7.2                                       nginx-deployment.yaml *
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
        volumeMounts:
        - name: web-content
          mountPath: /usr/share/nginx/html
      volumes:
      - name: web-content
        persistentVolumeClaim:
          claimName: my-pvc
```

Running deployment

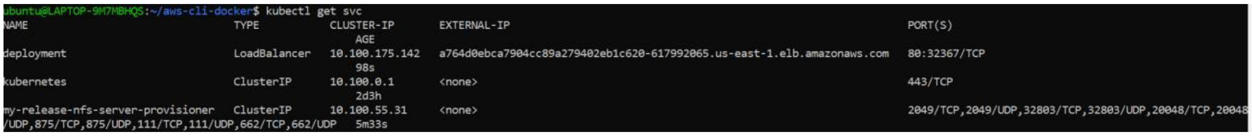
kubectl apply -f nginx-deployment.yaml



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get deployment
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
deployment   2/2     2            2           2m46s
```

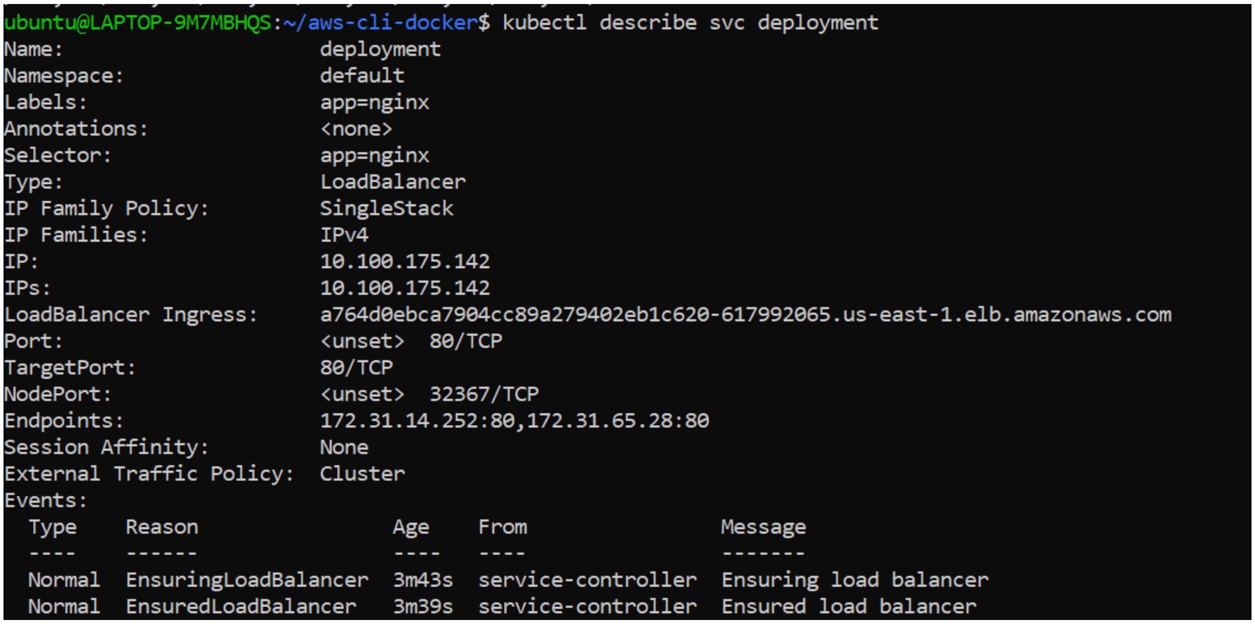
e) Create a Service associated with the Pod(s) of the HTTP server Deployment.

kubectl expose deployment/deployment --type=LoadBalancer --port=80



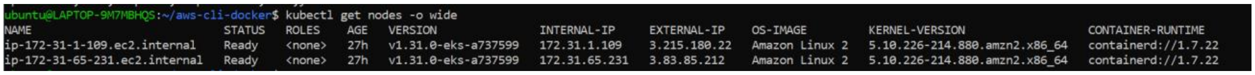
```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get svc
NAME                              TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)
                                                    AGE
deployment                        LoadBalancer   10.100.175.142   a764d0ebca7904cc89a279402eb1c620-617992065.us-east-1.elb.amazonaws.com   80:32367/TCP
                                                    98s
kubernetes                        ClusterIP      10.100.0.1       <none>                                                                    443/TCP
                                                    2d3h
my-release-nfs-server-provisioner   ClusterIP      10.100.55.31     <none>                                                                    2049/TCP,2049/UDP,32803/TCP,32803/UDP,20048/TCP,20048
/UDP,875/TCP,875/UDP,111/TCP,111/UDP,662/TCP,662/UDP   5m33s
```

Details about svc



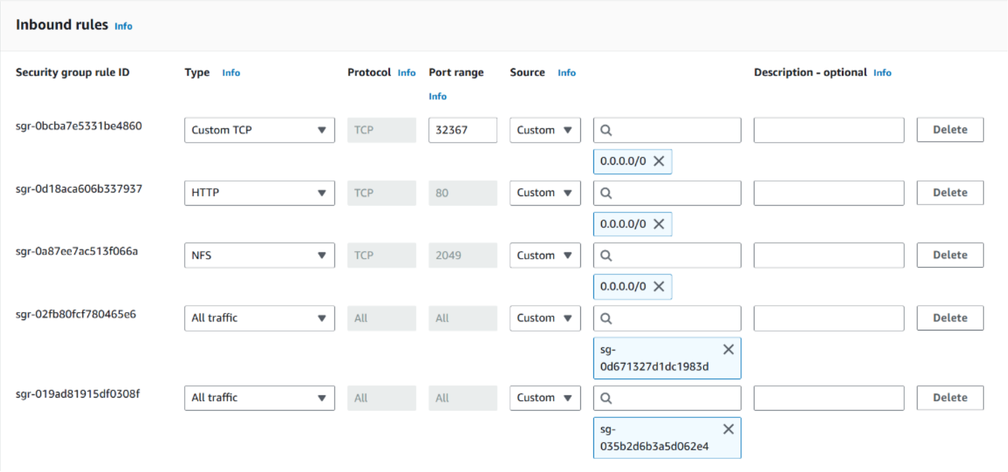
```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl describe svc deployment
Name:                     deployment
Namespace:                default
Labels:                   app=nginx
Annotations:              <none>
Selector:                 app=nginx
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.100.175.142
IPs:                      10.100.175.142
LoadBalancer Ingress:     a764d0ebca7904cc89a279402eb1c620-617992065.us-east-1.elb.amazonaws.com
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32367/TCP
Endpoints:                172.31.14.252:80,172.31.65.28:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason                Age    From                Message
  ----    ------                ----   ----                -------
  Normal  EnsuringLoadBalancer  3m43s  service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer   3m39s  service-controller  Ensured load balancer
```

Details about nodes



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get nodes -o wide
NAME                          STATUS   ROLES    AGE   VERSION               INTERNAL-IP     EXTERNAL-IP     OS-IMAGE         KERNEL-VERSION                   CONTAINER-RUNTIME
ip-172-31-1-109.ec2.internal   Ready    <none>   27h   v1.31.0-eks-a737599   172.31.1.109    3.215.180.22    Amazon Linux 2   5.10.226-214.880.amzn2.x86_64   containerd://1.7.22
ip-172-31-65-231.ec2.internal  Ready    <none>   27h   v1.31.0-eks-a737599   172.31.65.231   3.83.85.212     Amazon Linux 2   5.10.226-214.880.amzn2.x86_64   containerd://1.7.22
```

Security group rules

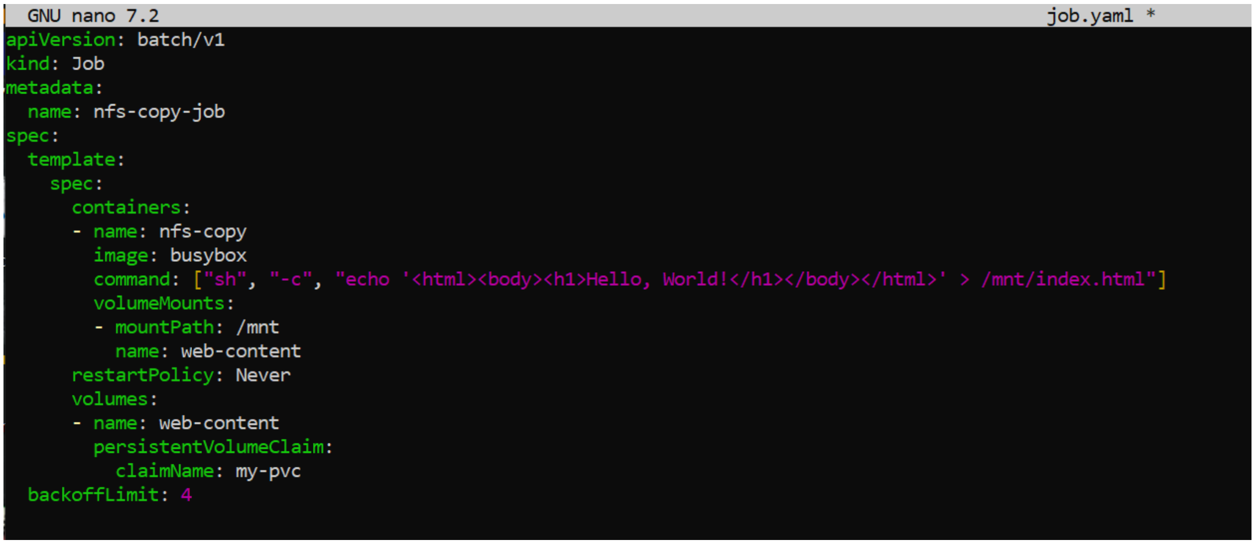


Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-0bcba7e5331be4860	Custom TCP	TCP	32367	Custom	0.0.0.0/0		Delete
sgr-0d18aca606b337937	HTTP	TCP	80	Custom	0.0.0.0/0		Delete
sgr-0a87ee7ac513f066a	NFS	TCP	2049	Custom	0.0.0.0/0		Delete
sgr-02fb80fcf780465e6	All traffic	All	All	Custom	sg-0d671327d1dc1983d		Delete
sgr-019ad81915df0308f	All traffic	All	All	Custom	sg-035b2d6b3a5d062e4		Delete

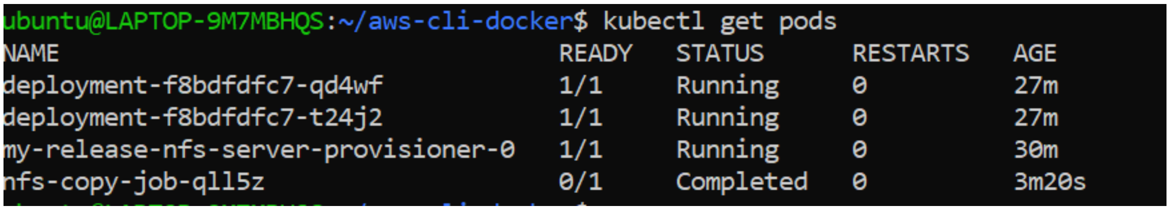
f) Create a Job which mounts the PVC and copies a sample content through the shared NFS PV.

Job configuration



```
  GNU nano 7.2                                                                                     job.yaml *
apiVersion: batch/v1
kind: Job
metadata:
  name: nfs-copy-job
spec:
  template:
    spec:
      containers:
      - name: nfs-copy
        image: busybox
        command: ["sh", "-c", "echo '<html><body><h1>Hello, World!</h1></body></html>' > /mnt/index.html"]
        volumeMounts:
        - mountPath: /mnt
          name: web-content
      restartPolicy: Never
      volumes:
      - name: web-content
        persistentVolumeClaim:
          claimName: my-pvc
  backoffLimit: 4
```

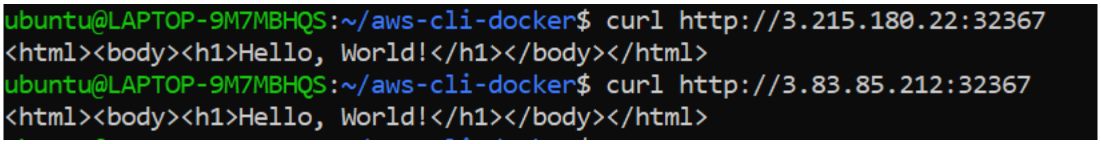
Running job



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ kubectl get pods
NAME                                READY   STATUS      RESTARTS   AGE
deployment-f8bdfdfc7-qd4wf          1/1     Running     0          27m
deployment-f8bdfdfc7-t24j2          1/1     Running     0          27m
my-release-nfs-server-provisioner-0 1/1     Running     0          30m
nfs-copy-job-ql15z                  0/1     Completed   0          3m20s
```

g) Test the HTTP server by showing the sample web content in a browser.

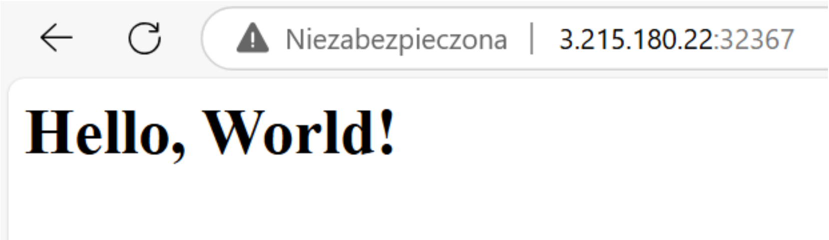
Test in console



```
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ curl http://3.215.180.22:32367
<html><body><h1>Hello, World!</h1></body></html>
ubuntu@LAPTOP-9M7MBHQS:~/aws-cli-docker$ curl http://3.83.85.212:32367
<html><body><h1>Hello, World!</h1></body></html>
```

Test HTTP

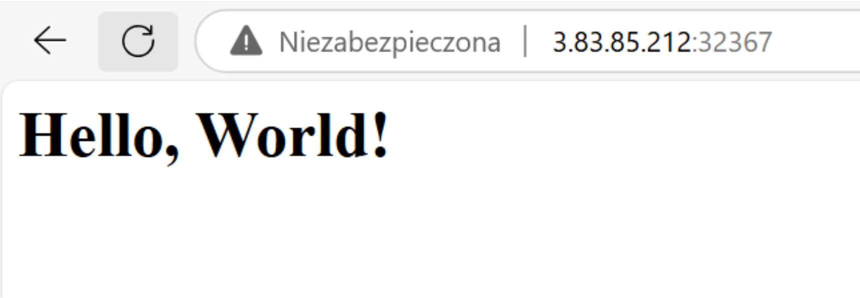
First replica



Niezabezpieczona | 3.215.180.22:32367

Hello, World!

Second replica



Hello, World!

Repository: https://github.com/honeyAsya/LSC

Command list:

```
nano ~/.aws/credentials or aws configure

nano Dockerfile

docker build -t aws-cli-docker .

docker run --rm -it aws-cli-docker –version

docker run --rm -it -v ~/.aws:/root/.aws aws-cli-docker -

aws sts get-caller-identity

aws eks update-kubeconfig --region us-east-1 --name Lab5

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

helm repo add nfs-ganesha-server-and-external-provisioner https://kubernetes-sigs.github.io/nfs-ganesha-server-and-external-provisioner/

helm install my-release nfs-ganesha-server-and-external-provisioner/nfs-server-provisioner

nano nfs-pvc.yaml

kubectl apply -f nfs-pvc.yaml

kubectl get pvc

nano nginx-deployment.yaml

kubectl apply -f nginx-deployment.yaml
kubectl expose deployment/deployment --type=LoadBalancer --port=80

kubectl get nodes

nano job.yaml

kubectl apply -f job.yaml

curl http://3.215.180.22:32367
```