IKT210 - Cloud Infrastructure

Group 8 - Robin Meier

Final Project

Autumn 2023

Fakultet for teknologi og realfag

Universitetet i Agder

Contents

1 Introduction

This project concerned itself with setting up a new k8s cluster and configuring multiple different deployments.

1.1 Requirements

The following requirements were given:

1. Setup a new k8s stack.
2. Setup a CI stack with ArgoCD, ArgoCD Image Update and Sealed Secrets
3. Setup a Monitoring stack with Blackbox Exporter, Prometheus, Alert Manager, Node Exporter, Grafana and Thanos.
4. Setup Bitwarden
5. Setup Cryptpad
6. Follow the NSA hardening guide for Kubernetes

2 How did the group solve this task

2.1 Cluster

2.1.1 Terraform Configuration

The nodes for the Kubernetes cluster were created using Terraform and deployed on Openstack. A new network including subnet, router and subnet route was configured for the nodes to reside in.

```
resource "openstack_networking_network_v2" "IKT210-G-23H-g8-final-network" {
    name                  = "IKT210-G-23H-g8-final"
    admin_state_up        = "true"
    port_security_enabled = "false"
}

resource "openstack_networking_subnet_v2" "IKT210-G-23H-g8-final-subnet" {
  name       = "IKT210-G-23H-g8-final"
  network_id = openstack_networking_network_v2.IKT210-G-23H-g8-final-network.id
  cidr       = "192.210.8.0/24"
  allocation_pool {
    start = "192.210.8.10"
    end   = "192.210.8.254"
  }
  dns_nameservers = ["158.37.218.20", "158.37.218.21", "128.39.54.10"]
  gateway_ip      = "192.210.8.1"
}

resource "openstack_networking_router_v2" "IKT210-G-23H-g8-final-router" {
  name                = "IKT210-G-23H-g8-final"
  admin_state_up      = true
  external_network_id = "9992655d-0892-4fe0-8a62-d9dac9044be2" # provider network
}

resource "openstack_networking_router_interface_v2" "IKT210-G-23H-g8-final-interface" {
  router_id = openstack_networking_router_v2.IKT210-G-23H-g8-final-router.id
  subnet_id = openstack_networking_subnet_v2.IKT210-G-23H-g8-final-subnet.id
}
```

Listing 5: Terraform network configuration

A variables.tf file was created, to define the variables used to store the connection details to Openstack and to define a variable setting the names of the nodes.

```
variable "OS_USERNAME" {}
variable "OS_PROJECT_NAME" {}
variable "OS_PASSWORD" {}
variable "OS_USER_DOMAIN_NAME" {}
variable "OS_PROJECT_DOMAIN_NAME" {}
variable "OS_AUTH_URL" {}
variable "OS_IDENTITY_API_VERSION" {}

variable "servers" {
  type    = list
  default = ["final-master", "final-worker1", "final-worker2"]
}
```

Listing 6: Terraform variables.tf file

For each node, a floating IPv4 address and a block storage volume of 1TB was created.

```
resource "openstack_networking_floatingip_v2" "IKT210-G-23H-g8-final-fip" {
  for_each = toset(var.servers)
  pool       = "provider"
}

resource "openstack_blockstorage_volume_v3" "IKT210-G-23H-g8-final-storage" {
    for_each = toset(var.servers)
    size = 1024
}
```

Listing 7: Terraform floating address and block storage volume creation

The nodes were created with the names defined in the `servers` variable and were added to the previously defined network. Finally the floating address and storage volume is assigned.

```
resource "openstack_compute_instance_v2" "IKT210-G-23H-g8-final-nodes" {
  for_each = toset(var.servers)
  name = each.key
  image_id = "d8e27e72-42b0-4c5c-890e-04fce014e83b"
  flavor_id = "42"
  key_pair = "ikt210"

  network {
    name = "IKT210-G-23H-g8-final"
  }
}

resource "openstack_compute_volume_attach_v2" "IKT210-23H-g8-final-attach" {
  for_each = openstack_compute_instance_v2.IKT210-G-23H-g8-final-nodes
  volume_id = openstack_blockstorage_volume_v3.IKT210-G-23H-g8-final-storage[each.key].id
  instance_id = each.value.id
}

resource "openstack_compute_floatingip_associate_v2" "IKT210-G-23H-g8-final-fip-associate" {
  for_each = openstack_compute_instance_v2.IKT210-G-23H-g8-final-nodes
  floating_ip = openstack_networking_floatingip_v2.IKT210-G-23H-g8-final-fip[each.key].address
  instance_id = each.value.id
}
```

Listing 8: Terraform node creation and assignment of additional resources

The Terraform configuration was then applied using `terraform apply`. The resulting nodes are shown in listing 1.

2.1.2 Kubernetes Cluster

The stack chosen for the Kubernetes cluster is CRI-O as the CRI, Flannel as the CNI and Rook with Ceph as the CSI.

CRIO-O Installation

First some environment variables are defined to install the correct version of CRI-O.

```
export OS=xUbuntu_22.04
export CRIO_VERSION=1.24
```

Listing 9: Setting CRI-O environment variables

Then the needed repositories are added to the sources list and the CRI-O package is installed. CRI-O could then be started.

```
echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/
    /"| sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list

echo "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/
    $CRIO_VERSION/$OS/ /"|sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-
    o:$CRIO_VERSION.list

curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:
    $CRIO_VERSION/$OS/Release.key | sudo apt-key add -
curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/
    Release.key | sudo apt-key add -

apt update
apt install cri-o cri-o-runc -y
```

Listing 10: Installing CRI-O

```
systemctl start crio
systemctl enable crio
```

Listing 11: Starting and enabling CRI-O

Cluster Installation

On all nodes the packages kubelet, kubeadm and kubectl were installed.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/
    apt/keyrings/kubernetes-apt-keyring.gpg

# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/
    stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Listing 12: Installation of kubeadm

On the master node the cluster could then be initialized. The pod network was set to 10.244.0.0/16, because Flannel uses this subnet by default. The `--cri-socket` parameter was used to specify, that CRI-O is used as the CRI.

```
$ kubeadm init --pod-network-cidr=10.244.0.0/16 --cri-socket=unix:///var/run/crio/crio.sock
...
Your Kubernetes control-plane has initialized successfully!
...
```

Listing 13: Initialization of cluster

To allow the regular ubuntu user to run the `kubectl` command, the configs were copied into the users home directory. Listing 15 displays the command to remove the `NoSchedule` taint from the master node. This makes it possible to deploy pods on it.

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Listing 14: Configuring kubectl

```
kubectl taint nodes final-master node-role.kubernetes.io/control-plane:NoSchedule-
```

Listing 15: Removing NoSchedule taint from master

On the worker nodes the command `kubeadm join` was then executed to join the cluster.

```
kubeadm join 192.210.8.88:6443 --token 0guwc2.ft0qexjn2ltt0c2d \
  --discovery-token-ca-cert-hash sha256:4
    e891dfbf6af157a3ba8b932f2894bd4ce4ed2ddfafed28244703d9c81e5b981  \
  --cri-socket=unix:///var/run/crio/crio.sock
```

Listing 16: Joining worker nodes to cluster

To confirm the nodes joining the cluster, `kubectl get nodes` was run on the master.

```
kubectl get nodes
NAME            STATUS   ROLES           AGE     VERSION
final-master    Ready    control-plane   9m24s   v1.28.4
final-worker1   Ready    <none>          2m27s   v1.28.4
final-worker2   Ready    <none>          30s     v1.28.4
```

Listing 17: Validating cluster nodes

Flannel Installation

The CNI Flannel was installed using the manifest found on the official GitHub page [1]. After installing the manifest, a flannel pod was started for each node.

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

Listing 18: Installing Flannel manifest

```
$ kubectl get pods -n kube-flannel
NAME                    READY   STATUS    RESTARTS   AGE
kube-flannel-ds-6rlxd   1/1     Running   0          41s
kube-flannel-ds-jj6wx   1/1     Running   0          41s
kube-flannel-ds-nhhgl   1/1     Running   0          41s
```

Listing 19: Validating Flannel installation

Rook Installation

Rook is used to set up a Ceph cluster and provide a block storage class. The installation manifests were pulled from the official GitHub repository of Rook [2]. The manifests crds.yaml and common.yaml contain common Rook resources.

```
kubectl apply -f crds.yaml -f common.yaml
```

Listing 20: Installing common Rook resources

In the operator.yaml file the parameter `ROOK_ENABLE_DISCOVERY_DAEMON` was enabled, which allows Rook to automatically find new volumes, as they are connected to the host system.

```
ROOK_ENABLE_DISCOVERY_DAEMON: "true"
```

Listing 21: Enabling discovery daemon in Rook operator

The Rook operator is then installed. In the rook-ceph namespace the added pods are displayed.

```
$ kubectl create -f operator.yaml
$ kubectl get pods -n rook-ceph
NAME                                  READY   STATUS    RESTARTS   AGE
rook-ceph-operator-77d685b47f-jtgbd   1/1     Running   0          108s
rook-discover-2qbjz                   1/1     Running   0          103s
rook-discover-2sfdz                   1/1     Running   0          104s
rook-discover-v4s2h                   1/1     Running   0          103s
```

Listing 22: Installing and validating Rook operator

To create the storage cluster the manifest cluster-test.yaml is used. This manifest creates an OSD for each node in the cluster.

```
$ kubectl create -f cluster-test.yaml
$  kubectl get pods -n rook-ceph
NAME                                                READY   STATUS      RESTARTS        AGE
csi-cephfsplugin-568n6                              2/2     Running     0               11m
csi-cephfsplugin-ldgst                              2/2     Running     0               11m
csi-cephfsplugin-llv6n                              2/2     Running     0               5m51s
csi-cephfsplugin-provisioner-55588874-6l47b         5/5     Running     0               11m
csi-cephfsplugin-provisioner-55588874-rzpk8         5/5     Running     0               11m
csi-rbdplugin-7fp64                                 2/2     Running     0               11m
csi-rbdplugin-8s878                                 2/2     Running     0               5m52s
csi-rbdplugin-provisioner-577dff4756-7tpf8          5/5     Running     0               11m
csi-rbdplugin-provisioner-577dff4756-sfjrm          5/5     Running     0               11m
csi-rbdplugin-swp6s                                 2/2     Running     0               11m
rook-ceph-exporter-final-master-86f5696ff4-rzrld    1/1     Running     0               88s
rook-ceph-exporter-final-worker1-578b59b6d5-wdfjk   1/1     Running     0               9m27s
rook-ceph-exporter-final-worker2-66488c4446-szt4b   1/1     Running     0               8m48s
rook-ceph-mgr-a-85db99cb8-5lmtl                     1/1     Running     0               10m
rook-ceph-mon-a-7758474dfb-2drsp                    1/1     Running     0               12m
rook-ceph-operator-77d685b47f-wr5jx                 1/1     Running     0               18m
rook-ceph-osd-0-696c59645-2gsjc                     1/1     Running     0               9m27s
rook-ceph-osd-1-7b95c6cd95-wtnkn                    1/1     Running     0               8m49s
rook-ceph-osd-2-bdfdcd47c-zwfxk                     1/1     Running     0               89s
rook-ceph-osd-prepare-final-master-t57dj            0/1     Completed   0               47s
rook-ceph-osd-prepare-final-worker1-sv7k8           0/1     Completed   0               39s
rook-ceph-osd-prepare-final-worker2-tcp4c           0/1     Completed   0               32s
rook-discover-6nhnv                                 1/1     Running     0               18m
rook-discover-pw8j9                                 1/1     Running     1 (4m31s ago)   5m52s
rook-discover-xpvlf                                 1/1     Running     0               18m
```

Listing 23: Installing and validating Rook cluster

To validate the functionality of the new storage cluster, the Rook toolbox is installed.

```
kubectl create -f toolbox.yaml
```

Listing 24: Setting up Rook toolbox

Using the command `ceph status` the current state of the storage cluster can be seen. Here it is seen, that the three 1TB disks added to the nodes were added to a storage pool.

```
$ kubectl -n rook-ceph exec deploy/rook-ceph-tools -- ceph status
  cluster:
    id:     cca3270b-ccf5-4820-917a-e3c9a97b583f
    health: HEALTH_OK

  services:
    mon: 1 daemons, quorum a (age 13m)
    mgr: a(active, since 10m)
    osd: 3 osds: 3 up (since 2m), 3 in (since 4m)

  data:
    pools:   1 pools, 32 pgs
    objects: 2 objects, 463 KiB
    usage:   79 MiB used, 3.0 TiB / 3 TiB avail
    pgs:     32 active+clean
```

Listing 25: Validating Ceph cluster state

To use the storage in Kubernetes, a block storage class is added using the storageclass.yaml manifest provided by Rook.

```
$ kubectl apply -f storageclass.yaml
cephblockpool.ceph.rook.io/replicapool created
storageclass.storage.k8s.io/rook-ceph-block created
```

Listing 26: Adding Rook storage class

2.2 CI

2.2.1 ArgoCD

To install ArgoCD, a new namespace is created and then the install.yaml is applied. The install.yaml file was copied over from exercise four.

```
kubectl create namespace argocd
kubectl apply -n argocd -f install.yaml
```

Listing 27: Installing ArgoCD

This started multiple pods and services, with the `argocd-server` LoadBalancer being the access to the ArgoCD web interface.

```
NAME                                      TYPE           PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP      7000/TCP,8080/TCP            2m5s
argocd-dex-server                         ClusterIP      5556/TCP,5557/TCP,5558/TCP   2m5s
argocd-metrics                            ClusterIP      8082/TCP                     2m4s
argocd-notifications-controller-metrics   ClusterIP      9001/TCP                     2m4s
argocd-redis                              ClusterIP      6379/TCP                     2m3s
argocd-repo-server                        ClusterIP      8081/TCP,8084/TCP            2m3s
argocd-server                             LoadBalancer   80:31846/TCP,443:30797/TCP   2m2s
argocd-server-metrics                     ClusterIP      8083/TCP                     2m1s
```

Listing 28: ArgoCD Services

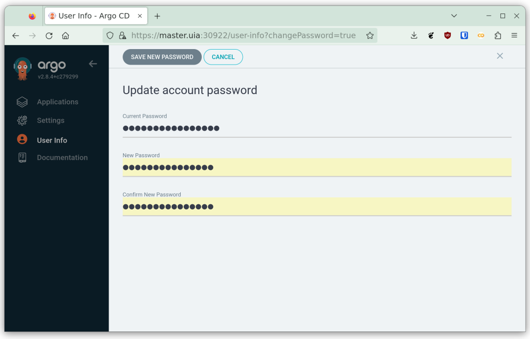
The CLI tool is installed and the default admin credentials are extracted. Using them, one could login to the web interface of ArgoCD. On there the default password was changed, as seen in figure 1.

```
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/latest/download/
    argocd-linux-amd64
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
rm argocd-linux-amd64
```

Listing 29: Installing ArgoCD CLI

```
argocd admin initial-password -n argocd
BRdKlhg2In3TIDzj

 This password must be only used for first time login. We strongly recommend you update the
    password using `argocd account update-password`.
```

Listing 30: Getting ArgoCD Default Password



Update account password

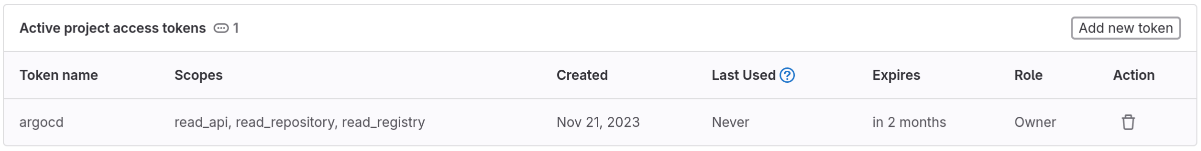
Current Password
●●●●●●●●●●●●●●●●●

New Password
●●●●●●●●●●●●●●●●

Confirm New Password
●●●●●●●●●●●●●●●●

Figure 1: Changing ArgoCD Password

To add the GitLab repository to ArgoCD, a new read only access token is generated in GitLab. Using this access token, the final-project repository is added (listing 31). As seen in figure 3, the repository is now shown in the web interface.

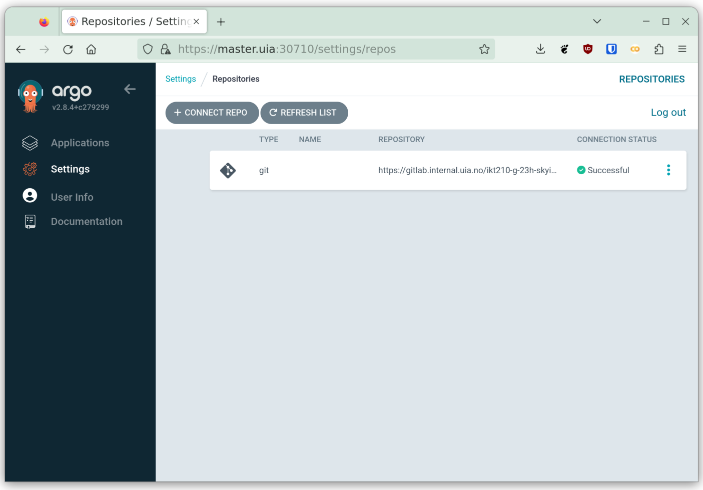


Token name	Scopes	Created	Last Used	Expires	Role	Action
argocd	read_api, read_repository, read_registry	Nov 21, 2023	Never	in 2 months	Owner	

Figure 2: Creating GitLab Access Token

```
argocd repo add https://gitlab.internal.uia.no/ikt210-g-23h-skyinfrastruktur/LabGroup8/final-
    project.git --username robinme --password glpat-ACCESSTOKEN
```

Listing 31: Adding GitLab Repository to ArgoCD



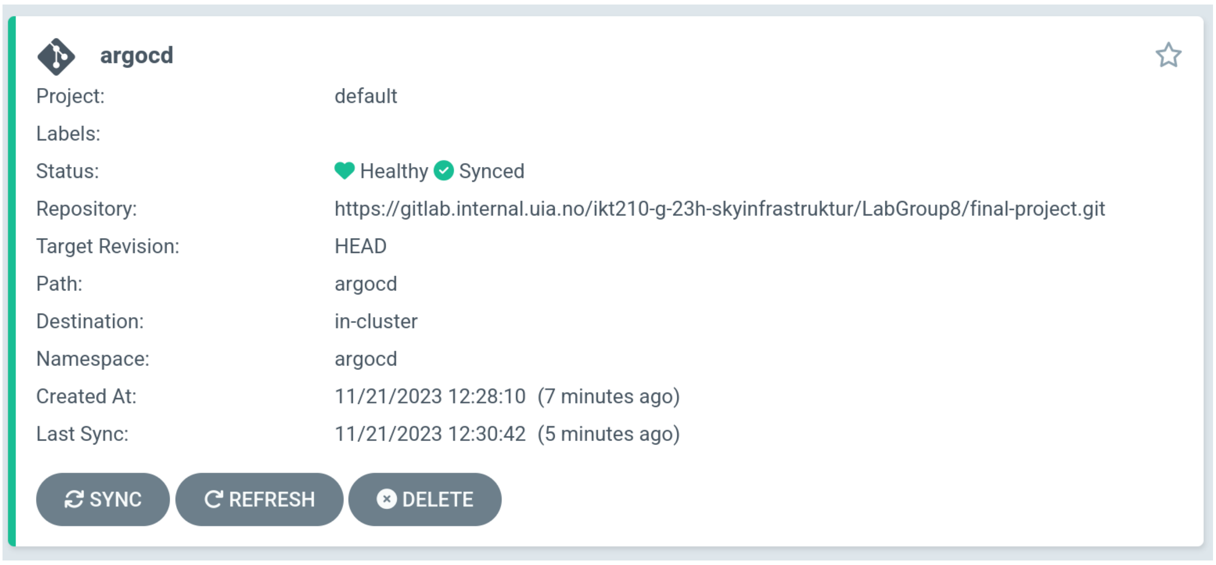
TYPE	NAME	REPOSITORY	CONNECTION STATUS
git		https://gitlab.internal.uia.no/ikt210-g-23h-skyi...	Successful

Figure 3: Validating Connection of Repository

ArgoCD was then added as an app inside ArgoCD, which is shown in figure 4.

```
argocd app create argocd \
--repo https://gitlab.internal.uia.no/ikt210-g-23h-skyinfrastruktur/LabGroup8/final-project.git
    \
--path ci/argocd \
--dest-server https://kubernetes.default.svc \
--dest-namespace argocd
--self-heal \
--sync-policy automated \
--sync-retry-limit 5

argocd app set argocd --auto-prune
```

Listing 32: Creating ArgoCD app inside ArgoCD


argocd

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://gitlab.internal.uia.no/ikt210-g-23h-skyinfrastruktur/LabGroup8/final-project.git
Target Revision:	HEAD
Path:	argocd
Destination:	in-cluster
Namespace:	argocd
Created At:	11/21/2023 12:28:10 (7 minutes ago)
Last Sync:	11/21/2023 12:30:42 (5 minutes ago)

Figure 4: ArgoCD app on ArgoCD management site

2.2.2 ArgoCD Image Update

ArgoCD Image Update was installed using the guide on their official website [3]. Applying the install.yaml started an additional pod in the argocd namespace (listing 34).

```
kubectl apply -n argocd -f install.yaml
```

Listing 33: Installing ArgoCD Image Updater

```
$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          140m
argocd-applicationset-controller-568754c579-j87s6   1/1     Running   0          140m
argocd-dex-server-7658dcdf77-sfrjc                  1/1     Running   0          140m
argocd-image-updater-88454679d-vhq7r                1/1     Running   0          34s
argocd-notifications-controller-5548b96954-x6fbp    1/1     Running   0          140m
argocd-redis-6976fc7dfc-7wpvb                       1/1     Running   0          140m
argocd-repo-server-7594f8849c-kkhx9                 1/1     Running   0          140m
argocd-server-58cc545d87-x4rcf                      1/1     Running   0          140m
```

Listing 34: Checking ArgoCD pods

To validate the functionality of Image Updater the Nginx image was tested.

```
$ kubectl exec --stdin --tty -n argocd argocd-image-updater-88454679d-vhq7r -- argocd-image-
    updater test nginx

DEBU[0000] Creating in-cluster Kubernetes client
INFO[0000] retrieving information about image image_alias= image_name=nginx registry_url=
INFO[0000] Fetching available tags and metadata from registry  application=test image_alias=
    image_name=nginx registry_url=
DEBU[0000] Using canonical image name 'library/nginx' for image 'nginx'  application=test
    image_alias= image_name=nginx registry_url=
INFO[0001] Found 560 tags in registry application=test image_alias= image_name=nginx
    registry_url=
INFO[0001] latest image according to constraint is nginx:1.25.3  application=test image_alias=
    image_name=nginx registry_url=
```

Listing 35: Testing Image Update

2.2.3 Sealed Secrets

Sealed Secrets was installed using the guide from the official Github repository [4]. Installing it added a new pod in the kube-system namespace (listing 37).

```
kubectl apply -f controller.yaml
```

Listing 36: Installing Sealed Secrets

```
$ kubectl get pods -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
coredns-5dd5756b68-4v5x4                     1/1     Running   0          17s
coredns-5dd5756b68-6gdbp                     1/1     Running   0          61m
etcd-final-master                            1/1     Running   0          142m
kube-apiserver-final-master                  1/1     Running   0          142m
kube-controller-manager-final-master         1/1     Running   0          142m
kube-proxy-29xfj                             1/1     Running   0          142m
kube-proxy-5m4bb                             1/1     Running   0          136m
kube-proxy-wkbhq                             1/1     Running   0          136m
kube-scheduler-final-master                  1/1     Running   0          142m
sealed-secrets-controller-7f5c556578-h8k69   1/1     Running   0          116s
```

Listing 37: Checking Sealed Secrets pod

The kubeseal CLI was installed from their repository and to test the functionality a new test secret was created in the manifest test-secret.yaml. This secret was then sealed and the sealed secret was created. Listing 39 shows the process.

```
KUBESEAL_VERSION='0.24.4' # Set this to, for example, KUBESEAL_VERSION='0.23.0'
wget "https://github.com/bitnami-labs/sealed-secrets/releases/download/v${KUBESEAL_VERSION:?}/
    kubeseal-${KUBESEAL_VERSION:?}-linux-amd64.tar.gz"
tar -xvzf kubeseal-${KUBESEAL_VERSION:?}-linux-amd64.tar.gz kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

Listing 38: Installing kubeseal CLI

```
$ echo -n "OhThisIsSoSecretYouCouldNotBelieveYourEyes" | base64
T2hUaGlzSXNTb1NlY3JldFlvdUNvdWxkTm90QmVsaWV2ZVlvdXJFeWVz

kubeseal -f test-secret.yaml -w test-sealed-secret.yaml
kubectl create -f test-sealed-secret.yaml

$ kubectl get secret mysecret
NAME       TYPE     DATA   AGE
mysecret   Opaque   1      19s

$ kubectl get secret mysecret -o jsonpath='{.data}'
{"password":"T2hUaGlzSXNTb1NlY3JldFlvdUNvdWxkTm90QmVsaWV2ZVlvdXJFeWVz"}

$ echo "T2hUaGlzSXNTb1NlY3JldFlvdUNvdWxkTm90QmVsaWV2ZVlvdXJFeWVz" | base64 --decode
OhThisIsSoSecretYouCouldNotBelieveYourEyes
```

Listing 39: Testing Sealed Secrets

2.3 Monitoring

The monitoring was set up using Prometheus Operator and the getting started guide available on the Prometheus Operator website [5]. The operator manifest was downloaded from Github and installed.

```
LATEST=$(curl -s https://api.github.com/repos/prometheus-operator/prometheus-operator/releases/
    latest | jq -cr .tag_name)

wget https://github.com/prometheus-operator/prometheus-operator/releases/download/${LATEST}/
    bundle.yaml

kubectl create -f .
```

Listing 40: Installation of Prometheus Operator

This started the prometheus-operator pod as shown below.

```
$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-669dd4ddbf-hh479   1/1     Running   0          25h
```

Listing 41: Prometheus Operator pod running

The getting started guide also sets up an example application to validate the functionality of Prometheus. This application has been deployed in the mon-example namespace using Kustomize. The Deployment manifest example-app.yaml is shown in listing 43. It starts three replicas of a pod, which expose metrics on port 8080. The metrics are collected using a PodMonitor resource.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
spec:
  replicas: 3
  ...
  template:
    metadata:
      labels:
        app: example-app
    spec:
      containers:
      - name: example-app
        ...
        image: fabxc/instrumented_app
        ports:
        - name: web
          containerPort: 8080
---
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
spec:
  selector:
    matchLabels:
      app: example-app
  podMetricsEndpoints:
  - port: web
```

Listing 42: Setup of example application to test PodMonitor

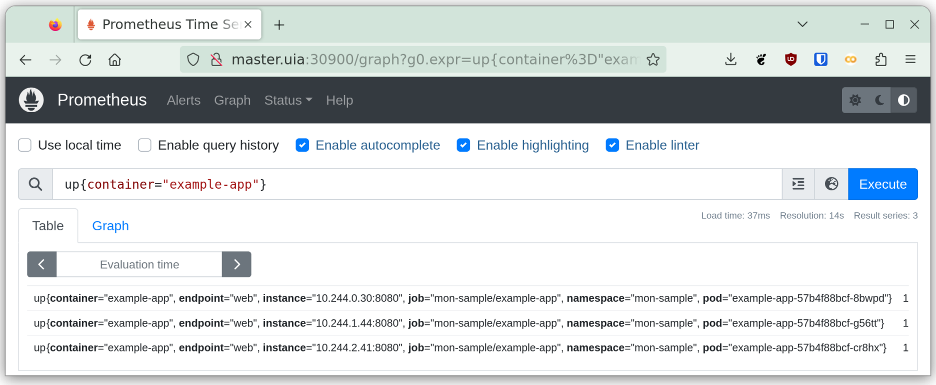
2.3.1 Prometheus

For Prometheus to be able to access the metrics, a ClusterRole first has to be created, which allows Prometheus to get different resources in the cluster. The ClusterRole and ClusterRoleBinding are configured in the serviceaccount.yaml file and were applied beforehand. Then a Prometheus instance was started using the Prometheus resource type. This is done in the manifest prometheus.yaml. Prometheus was configured to gather metrics from all namespaces with the `serviceMonitorNamespaceSelector`, `podMonitorNamespaceSelector` and `probeNamespaceSelector` keys. A NodePort was added, to be able to access the web interface of Prometheus.

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  serviceAccountName: prometheus
  serviceMonitorNamespaceSelector: {}
  serviceMonitorSelector: {}
  podMonitorSelector: {}
  podMonitorNamespaceSelector: {}
  probeNamespaceSelector: {}
  probeSelector: {}
  ...
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus-external
spec:
  type: NodePort
  ports:
  - name: web
    nodePort: 30900
    port: 9090
    protocol: TCP
    targetPort: web
  selector:
    prometheus: prometheus
```

Listing 43: Setup of example application to test PodMonitor

After deploying the manifest, it is possible to access the Prometheus web interface and see the collected metrics from the example application.



Use local time: ☐ Enable query history: ☐ Enable autocomplete: ✓ Enable highlighting: ✓ Enable linter: ✓

```
up{container="example-app"}
```

Load time: 37ms Resolution: 14s Result series: 3

```
up{container="example-app", endpoint="web", instance="10.244.0.30:8080", job="mon-sample/example-app", namespace="mon-sample", pod="example-app-57b4f88bcf-8bwpd"}  1
up{container="example-app", endpoint="web", instance="10.244.1.44:8080", job="mon-sample/example-app", namespace="mon-sample", pod="example-app-57b4f88bcf-g56tt"}  1
up{container="example-app", endpoint="web", instance="10.244.2.41:8080", job="mon-sample/example-app", namespace="mon-sample", pod="example-app-57b4f88bcf-cr8hx"}  1
```

Figure 5: Accessing Prometheus and querying example-app "up" metrics

2.3.2 Node Exporter

To monitor the cluster nodes, Node Exporter is used, which was set up by following a guide from Devopscube [6]. The container image `prom/node-eporter` is used and arguments are set, for what to monitor. A DaemonSet instead of a Deployment is used to make sure, that each node has exactly one Node Exporter instance running. The host paths `/` and `/sys` are mounted into the pods as read only. The ServiceMonitor connects to a ClusterIP Service, which then gets picked up by Prometheus. The whole deployment is contained in the node-exporter.yaml file.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
    ...
    spec:
      containers:
      - args:
        - --path.sysfs=/host/sys
        - --path.rootfs=/host/root
        - --no-collector.wifi
        - --no-collector.hwmon
        - --collector.filesystem.ignored-mount-points=^/(dev|proc|sys|var/lib/docker/.+|var/lib/
    kubelet/pods/.+)($|/)
        - --collector.netclass.ignored-devices=^(veth.*)$
        name: node-exporter
        image: prom/node-exporter
        ports:
          - containerPort: 9100
            protocol: TCP
        resources:
          limits:
            ...
        volumeMounts:
        - mountPath: /host/sys
          mountPropagation: HostToContainer
          name: sys
          readOnly: true
        - mountPath: /host/root
          mountPropagation: HostToContainer
          name: root
          readOnly: true
      volumes:
      - hostPath:
          path: /sys
        name: sys
      - hostPath:
          path: /
        name: root
```

Listing 44: Node Exporter DaemonSet configuration

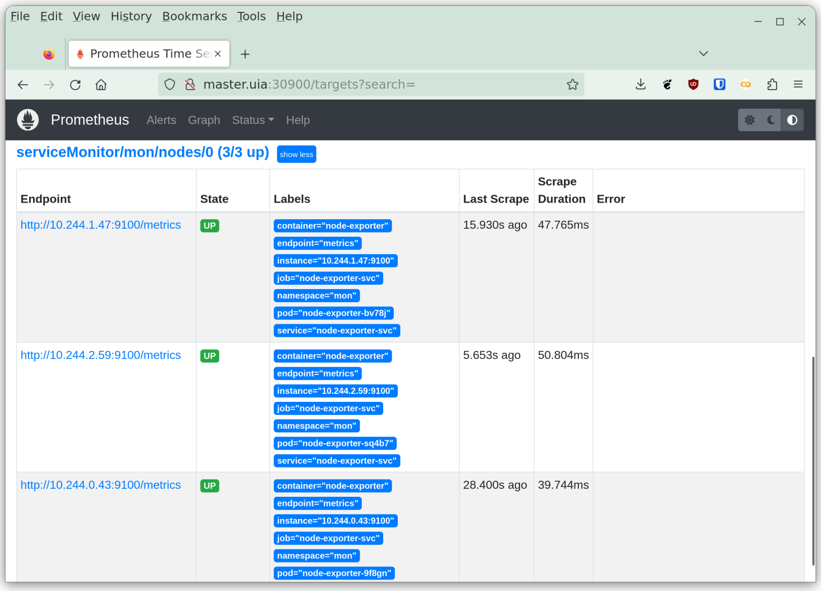
```
apiVersion: v1
kind: Service
metadata:
  name: node-exporter-svc
  labels:
    app: node-exporter
spec:
  type: ClusterIP
  selector:
      app: node-exporter
  ports:
  - name: metrics
    port: 9100
    targetPort: 9100
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: nodes
spec:
  selector:
    matchLabels:
      app: node-exporter
  endpoints:
  - port: metrics
```

Listing 45: Node Exporter ServiceMonitor configuration

To validate, that a pod is started on each node of the cluster, the command in listing 46 was executed. On the Prometheus web interface the three nodes are now visible under "Status" → "Targets" as seen in figure 6.

```
$ kubectl get pods -n mon -o wide | grep node-exporter
node-exporter-9f8gn     1/1     Running     10.244.0.43    final-master
node-exporter-bv78j     1/1     Running     10.244.1.47    final-worker1
node-exporter-sq4b7     1/1     Running     10.244.2.59    final-worker2
```

Listing 46: Node Exporter Pods


File Edit View History Bookmarks Tools Help
Prometheus Time Se
master.uia:30900/targets?search=
Prometheus
Alerts
Graph
Status
Help
show less

serviceMonitor/mon/nodes/0 (3/3 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.244.1.47:9100/metrics	UP	container="node-exporter" endpoint="metrics" instance="10.244.1.47:9100" job="node-exporter-svc" namespace="mon" pod="node-exporter-bv78j" service="node-exporter-svc"	15.930s ago	47.765ms	
http://10.244.2.59:9100/metrics	UP	container="node-exporter" endpoint="metrics" instance="10.244.2.59:9100" job="node-exporter-svc" namespace="mon" pod="node-exporter-sq4b7" service="node-exporter-svc"	5.653s ago	50.804ms	
http://10.244.0.43:9100/metrics	UP	container="node-exporter" endpoint="metrics" instance="10.244.0.43:9100" job="node-exporter-svc" namespace="mon" pod="node-exporter-9f8gn"	28.400s ago	39.744ms	

Figure 6: Node Exporter targets in Prometheus

2.3.3 Grafana

Grafana was set up using the `grafana/grafana` image. A PVC for all persistent data has been created and mounted on `/var/lib/grafana` path and a ConfigMap describing the datasources has been added as well (listing 48). To access the web interface a NodePort to port 30000 has been added. For Grafana to be able to connect to Prometheus a new ClusterIP Service has been added (listing 49).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
spec:
  selector:
    matchLabels:
      app: grafana
    spec:
      ...
      containers:
      - name: grafana
        image: grafana/grafana
        resources:
          limits:
            memory: "1Gi"
            cpu: "500m"
        ports:
        - containerPort: 3000
        volumeMounts:
        - name: grafana-storage
          mountPath: /var/lib/grafana
        - name: grafana-config
          mountPath: /etc/grafana/provisioning/datasources/datasource.yaml
          subPath: datasource.yaml
      volumes:
        - name: grafana-storage
          persistentVolumeClaim:
            claimName: grafana-storage
        - name: grafana-config
          configMap:
            name: grafana-config
---
apiVersion: v1
kind: Service
metadata:
  name: grafana-http
spec:
  type: NodePort
  selector:
    app: grafana
  ports:
  - port: 3000
    nodePort: 30000
....
```

Listing 47: Grafana Deployment

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: grafana-config
data:
  datasource.yaml: |
    apiVersion: 1
    datasources:
      - name: Prometheus
        type: prometheus
        access: server
        url: http://prometheus-internal:9090
        version: 1
        editable: true
        ...
```

Listing 48: Grafana datasource.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-internal
spec:
  type: ClusterIP
  ports:
  - name: web
    targetPort: web
    port: 9090
    protocol: TCP
  selector:
    prometheus: prometheus
```

Listing 49: Prometheus internal Service

After deployment, Grafana could be accessed in a browser using the default credentials `admin:admin`, which were immediately changed. Under "Dashboards" → "New" → "New Dashboard" → "Import Dashboard" a Node Exporter dashboard, found on the Grafana website [7] was added. Prometheus was selected as the datasource and metrics were immediatly displayed, as seen in figure 7.

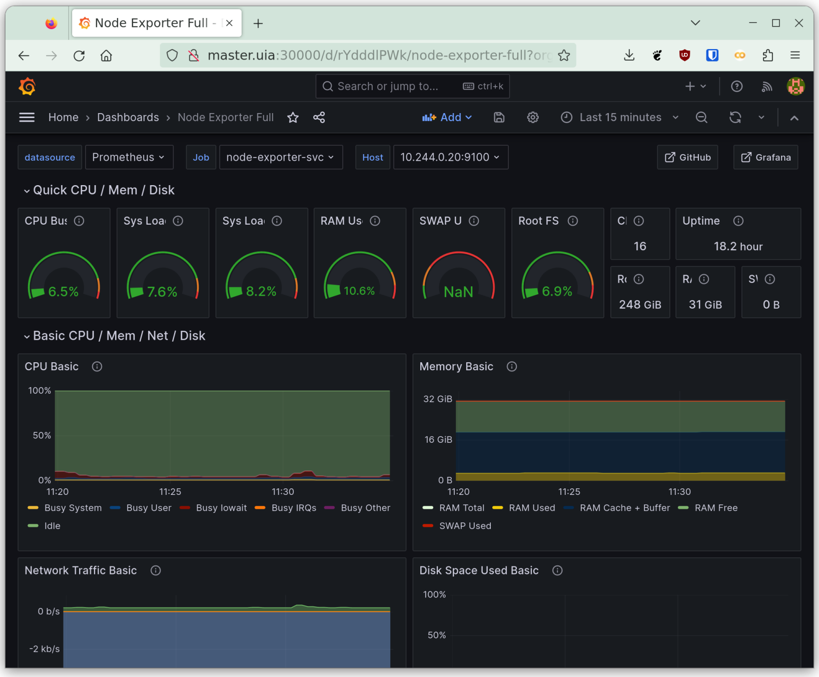


Figure 7: Node Exporter metrics in Grafana

2.3.4 Blackbox Exporter

The deployment of Blackbox Exporter is based on the example manifests found on the kube-prometheus repository [8]. Since the manifests found there are rather large, it was shortened to only the necessary parts: the Deployment, Service and ConfigMap, with the ConfigMap being copied directly, since it provides a good baseline for different blackbox probes. Blackbox Exporter itself is a pod with a ClusterIP Service running on port 9115.

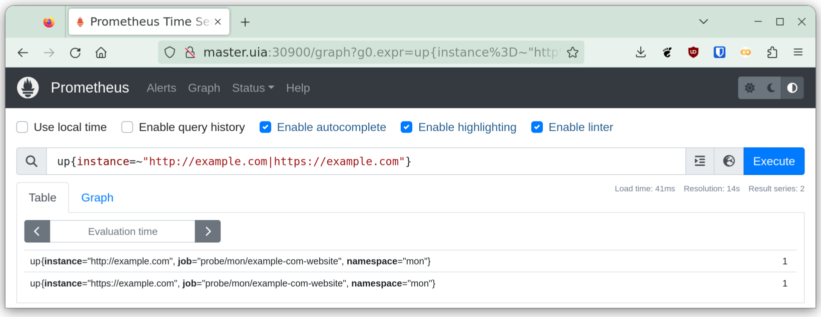
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blackbox-exporter
spec:
  selector:
    ...
    spec:
      containers:
      - name: blackbox-exporter
        image: quay.io/prometheus/blackbox-exporter:latest
        ...
        ports:
        - containerPort: 9115
          name: http
---
apiVersion: v1
kind: Service
metadata:
  name: blackbox-exporter
spec:
  selector:
    app: blackbox-exporter
  ports:
  - name: probe
    port: 9115
    targetPort: http
```

Listing 50: Blackbox Exporter Deployment

To test the deployment, the Probe resource type from Prometheus Operator could be used. An example from the Prometheus Operators documentation [9] was deployed, which uses the Blackbox Exporter to probe the sites `http://example.com` and `https://example.com` every 60 seconds (listing 51). The result of this could be seen on the Prometheus web interface (figure 8). Additionally, a new dashboard was added [10] to Grafana, which displays the Probes metrics (figure 9).

```
kind: Probe
apiVersion: monitoring.coreos.com/v1
metadata:
  name: example-com-website
spec:
  interval: 60s
  module: http_2xx
  prober:
    url: blackbox-exporter:9115
  targets:
    staticConfig:
      static:
      - http://example.com
      - https://example.com
```

Listing 51: Blackbox Exporter Probe



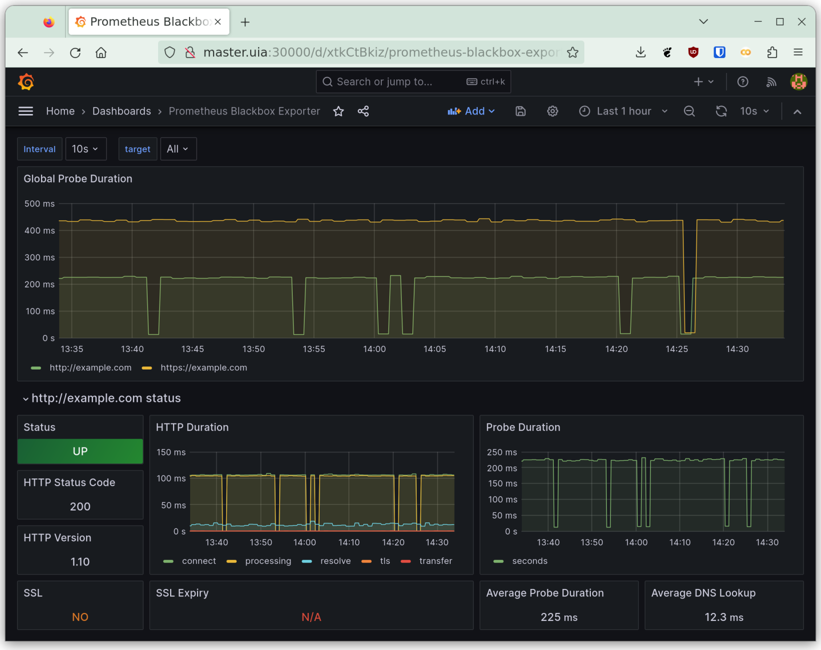
☐ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

```
up{instance=~"http://example.com|https://example.com"}
```

Load time: 41ms Resolution: 14s Result series: 2

```
up{instance="http://example.com", job="probe/mon/example-com-website", namespace="mon"}    1
up{instance="https://example.com", job="probe/mon/example-com-website", namespace="mon"}   1
```

Figure 8: Blackbox Exporter metrics in Prometheus



Interval: 10s target: All

http://example.com status

Status	UP
HTTP Status Code	200
HTTP Version	1.10
SSL	NO
SSL Expiry	N/A
Average Probe Duration	225 ms
Average DNS Lookup	12.3 ms

Figure 9: Blackbox Exporter metrics in Grafana

2.3.5 Alert Manager

Prometheus Operator provides Alertmanager, AltermanagerConfig and PrometheusRule types, which allow the configuration of alerting. A new Alertmanager with three replicas was created, which uses a webhook as a configuration (listing 52). A NodePort Service was added to see the AlertManager web interface (listing 52). To test the functionality of the AlertManager a new PrometheusRule was added, which triggers an example alert (listing 53). The Alertmanager and PrometheusRule were appended to the Prometheus resource (listing 54).

```
apiVersion: monitoring.coreos.com/v1
kind: Alertmanager
metadata:
  name: alertmanager
spec:
  replicas: 3
  alertmanagerConfigSelector:
    matchLabels:
      alertmanagerConfig: webhook
---
apiVersion: monitoring.coreos.com/v1alpha1
```

```yaml
kind: AlertmanagerConfig
metadata:
  name: alertmanager-config
  labels:
    alertmanagerConfig: webhook
spec:
  route:
    receiver: 'webhook'
  receivers:
  - name: 'webhook'
    webhookConfigs:
    - url: 'https://webhook.site/1db406d3-7d33-467d-9be1-40da46c8402b'
---
apiVersion: v1
kind: Service
metadata:
  name: alertmanager
spec:
  type: NodePort
  ports:
  - name: web
    nodePort: 30903
    port: 9093
    protocol: TCP
    targetPort: web
  selector:
    alertmanager: alertmanager
```

Listing 52: Alertmanager and AlertmanagerConfig setup

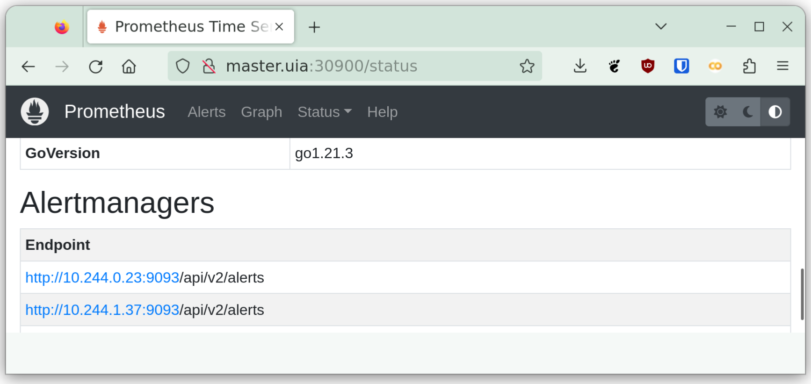
```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  creationTimestamp: null
  labels:
    role: alert-rules
  name: prometheus-example-rules
spec:
  groups:
  - name: ./example.rules
    rules:
    - alert: ExampleAlert
      expr: vector(1)
```

Listing 53: Example alert PrometheusRule

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  ...
  alerting:
    alertmanagers:
    - namespace: mon
      name: alertmanager
      port: web
  ruleSelector:
    matchLabels:
      role: alert-rules
  ...
```

Listing 54: Adding alerting to Prometheus resource

The Alert Manager instances are now shown on the Prometheus web interface under "Status" → "Runtime & Build Information". The triggered example alert is shown in Prometheus as well as on the Alert Manager web interface.

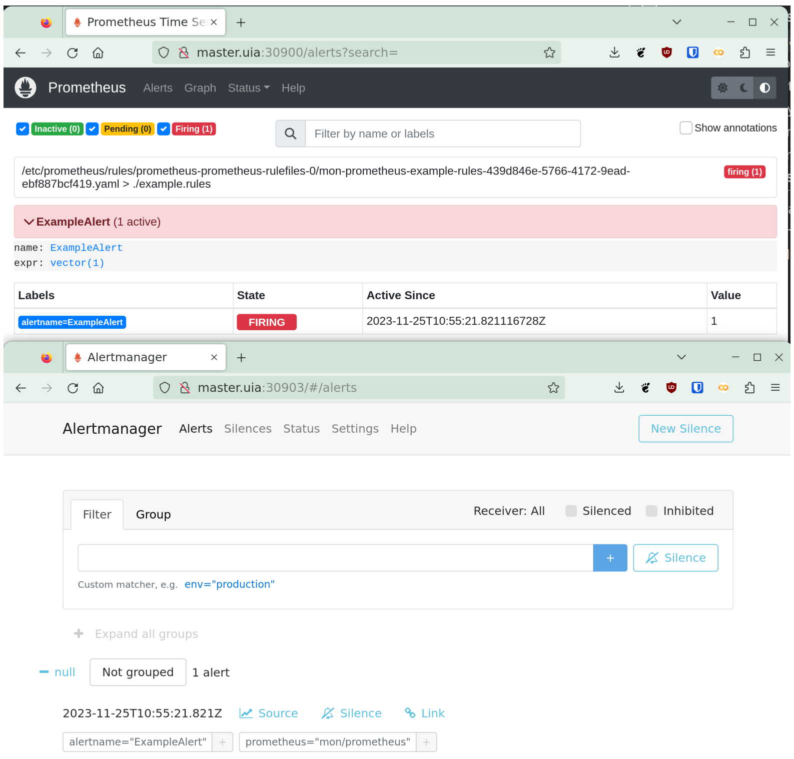


GoVersion	go1.21.3

Alertmanagers

Endpoint
http://10.244.0.23:9093/api/v2/alerts
http://10.244.1.37:9093/api/v2/alerts

Figure 10: Alert Managers in Prometheus



/etc/prometheus/rules/prometheus-prometheus-rulefiles-0/mon-prometheus-example-rules-439d846e-5766-4172-9ead-ebf887bcf419.yaml > ./example.rules firing (1)

ExampleAlert (1 active)

```
name: ExampleAlert
expr: vector(1)
```

Labels	State	Active Since	Value
alertname=ExampleAlert	FIRING	2023-11-25T10:55:21.821116728Z	1

2023-11-25T10:55:21.821Z Source Silence Link

alertname="ExampleAlert" prometheus="mon/prometheus"

Figure 11: Example alert shown in both Prometheus and Alert Manager

2.3.6 Thanos

Since Thanos contains multiple components and the task did not specify what to install, the Query system was configured. For Thanos Query to make sense, the Prometheus resource was extended to be higly available with two replicas. On the resource the Thanos sidecar system was appended (listing 55).

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  ...
  replicas: 2
  thanos:
    image: quay.io/thanos/thanos:v0.32.5
    objectStorageConfig:
      key: thanos.yaml
      name: thanos-objstore-config
```

Listing 55: Prometheus Thanos sidecar and replicas

To store metrics via Thanos a new object bucket was created using Rook (listing 56). The connection details of the bucket were exported and added to a configuration file called thanos-config.yaml (listing 57). These configuration were then added to a secret and sealed using `kubeseal` (listing 58). On the Prometheus resource, the `objectStorageConfig` key, configures the sidecar to store data in the newly created bucket.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: thanos-store
spec:
  generateBucketName: thanos-store
  storageClassName: rook-ceph-bucket
```

Listing 56: Ceph bucket for Thanos object storage

```
type: s3
config:
  bucket: thanos-store-31d15f83-877b-4d6c-af37-bd2473ae68d7
  endpoint: rook-ceph-rgw-ceph-object.rook-ceph.svc
  access_key: DTUXW9D6CM7C7QPK5MWU
  secret_key: H8Gc5yCzc7I33mfAZoLCr5ixkkCtAkfSzUaWAVZ6
```

Listing 57: Connection details in thanos-config.yaml

```
kubectl -n mon create secret generic thanos-objstore-config --from-file=thanos.yaml=thanos-
    config.yaml

kubectl get secret -n mon thanos-objstore-config --output=yaml > thanos-objstore-config.yaml

kubeseal -f thanos-objstore-config.yaml -w sealed-thanos-objstore-config.yaml
```

Listing 58: Sealing bucket connection details

The configuration of the querier is based on the quick start guide from the Thanos website [11]. For the Deployment, the `quay.io/thanos/thanos:v0.32.5` image with the `query` argument is used (listing 59). Thanos Query has two Services, one for internal network acccess to connect to the Prometheus instances and a NodePort for accessing the Thanos web interface (listing 60).

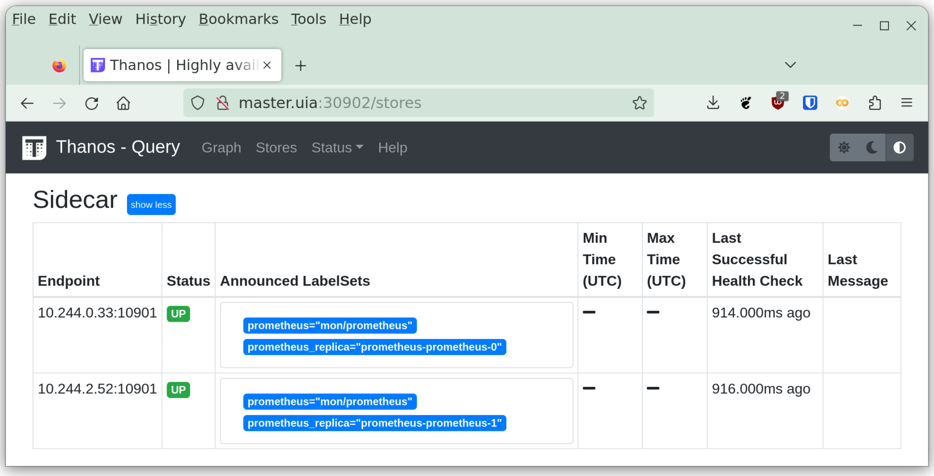
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thanos-querier
spec:
  selector:
    matchLabels:
      app: thanos-querier
    ...
    spec:
      containers:
      - name: thanos-querier
        image: quay.io/thanos/thanos:v0.32.5
        args:
        - query
        - --log.level=debug
        - --query.replica-label=prometheus_replica
        - --store=dnssrv+_grpc._tcp.thanos-sidecar.mon.svc.cluster.local
        ...
        ports:
        - containerPort: 10902
          name: http
```

Listing 59: Thanos Query Deployment

```
apiVersion: v1
kind: Service
metadata:
  name: thanos-querier
spec:
  ports:
  - name: http
    port: 10902
    targetPort: http
  selector:
    app: thanos-querier
---
apiVersion: v1
kind: Service
metadata:
  name: thanos-querier-external
spec:
  type: NodePort
  ports:
  - name: http
    port: 10902
    nodePort: 30902
    targetPort: http
  selector:
    app: thanos-querier
```

Listing 60: Thanos Query Services

After deploying Thanos, the redundant Prometheus instances were visible on the Thanos web interface and metrics could be viewed.



Sidecar

Endpoint	Status	Announced LabelSets	Min Time (UTC)	Max Time (UTC)	Last Successful Health Check	Last Message
10.244.0.33:10901	UP	prometheus="mon/prometheus" prometheus_replica="prometheus-prometheus-0"	–	–	914.000ms ago	
10.244.2.52:10901	UP	prometheus="mon/prometheus" prometheus_replica="prometheus-prometheus-1"	–	–	916.000ms ago	

Figure 12: Thanos Query web interface displaying redundant Prometheus instances

2.3.7 Overview of deployed monitoring stack

```
$ kubectl get pods -n mon
NAME                                 READY   STATUS
alertmanager-alertmanager-0          2/2     Running
alertmanager-alertmanager-1          2/2     Running
alertmanager-alertmanager-2          2/2     Running
blackbox-exporter-75b466cb55-kdx2w   1/1     Running
grafana-894b6898d-skrqq              1/1     Running
node-exporter-9f8gn                  1/1     Running
node-exporter-bv78j                  1/1     Running
node-exporter-sq4b7                  1/1     Running
prometheus-prometheus-0              3/3     Running
prometheus-prometheus-1              3/3     Running
thanos-querier-67b9bc94db-5gbdl      1/1     Running

$ kubectl get svc -n mon
NAME                     TYPE        PORT(S)
alertmanager             NodePort    9093:30903/TCP
alertmanager-operated    ClusterIP   9093/TCP,9094/TCP,9094/UDP
blackbox-exporter        ClusterIP   9115/TCP
grafana-http             NodePort    3000:30000/TCP
node-exporter-svc        ClusterIP   9100/TCP
prometheus-external      NodePort    9090:30900/TCP
prometheus-internal      ClusterIP   9090/TCP
prometheus-operated      ClusterIP   9090/TCP,10901/TCP
thanos-querier           ClusterIP   10902/TCP
thanos-querier-external  NodePort    10902:30902/TCP
thanos-sidecar           ClusterIP   10901/TCP
thanos-storer            ClusterIP   10901/TCP,10902/TCP
```

Listing 61: Deployed Pods and Services in mon namespace

2.4 Apps

2.4.1 Bitwarden

Since Bitwarden does not have documentation for a direct Kubernetes deployment without Helm, the manifests had to be built from ground up. As a reference, the guide for installation on Linux was used [12]. Bitwarden uses a script to generate needed configuration files and certificates. The Installation process was run on a local virtual machine to generate the needed files. The installation ID and key were generated beforehand on the Bitwarden website.

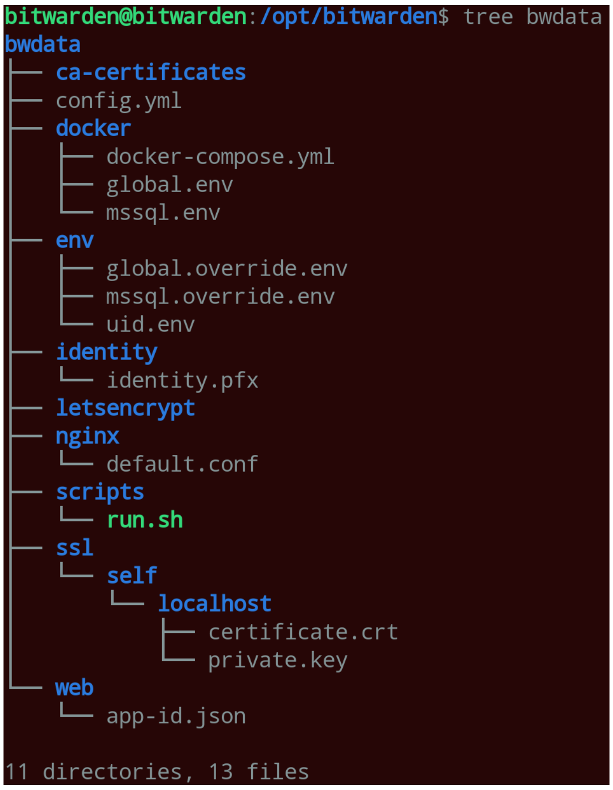
```
$ ./bitwarden.sh install
 _     _ _                         _
| |__ (_) |___      ____ _ _ __ __| | ___ _ __
| '_ \| | __\ \ /\ / / _` | '__/ _` |/ _ \ '_ \
| |_) | | |_ \ V  V / (_| | | | (_| |  __/ | | |
|_.__/|_|\__| \_/\_/ \__,_|_|  \__,_|\___|_| |_|

Open source password management solutions
Copyright 2015-2023, 8bit Solutions LLC
https://bitwarden.com, https://github.com/bitwarden

===================================================

bitwarden.sh version 2023.10.2
Docker version 24.0.7, build afdd53b
Docker Compose version v2.21.0

(!) Enter the domain name for your Bitwarden instance (ex. bitwarden.example.com): localhost
(!) Enter the database name for your Bitwarden instance (ex. vault): vault
(!) Enter your installation id (get at https://bitwarden.com/host): 55f73d24-0c2b-41b5-8dd0-
    b0c2010be6e3
(!) Enter your installation key: sCImn5KzOnPmqOeDtyvp
(!) Enter your region (US/EU) [US]: EU
(!) Do you have a SSL certificate to use? (y/N): n
(!) Do you want to generate a self-signed SSL certificate? (y/N): y
```

Listing 62: Generating Bitwarden Config

Figure 14 show the generated files, which were then used in the configuration of the manifests. The docker-compose.yaml used by Bitwarden contains eleven different containers, each with mounted volumens and environment variables set. To simplify the process of creating the deployment manifests, all PersistentStorageClaims were created in one separate manifest called storage.yaml. Each PVC is set to access mode `ReadWriteOnce`, has 5Gi of storage and uses the `rook-ceph-block` storage class.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: bitwarden-core
spec:
  resources:
    requests:
      storage: 5Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: rook-ceph-block
```

Listing 63: Example of PVC in storage.yaml



```
bitwarden@bitwarden:/opt/bitwarden$ tree bwdata
bwdata
├── ca-certificates
├── config.yml
├── docker
│   ├── docker-compose.yml
│   ├── global.env
│   └── mssql.env
├── env
│   ├── global.override.env
│   ├── mssql.override.env
│   └── uid.env
├── identity
│   └── identity.pfx
├── letsencrypt
├── nginx
│   └── default.conf
├── scripts
│   └── run.sh
├── ssl
│   └── self
│       └── localhost
│           ├── certificate.crt
│           └── private.key
└── web
    └── app-id.json

11 directories, 13 files
```

Figure 13: Generated files after running Bitwarden installation script

All environment variables found in the bwdata directory were added to one manifest called env.yaml and then sealed using kubeseal. The env.yaml file contains secrets, so this file would normally not be pushed to the repository, but is in this case for educational purposes.

```
kubeseal -f env.yaml -w sealed-env.yaml
```

Listing 64: Sealing of env.yaml

Deployments were configured based on the docker-compose.yaml, with the identity and nginx deployments having additional Init-Containers, which copy data from the bwdata directory into a persistent volume. Listing 65 shows the Dockerfile of the Init-Container used. Environment variables are added using the `envFrom: secretRef:` notation. Listing 66 shows the identity deployment in the original docker-compose.yaml file and listing 67 shows the translated deployment in a Kubernetes manifest. Some information has been removed for a better overview. Since multiple deployments mount the same volumes and all PVCs have been created with `ReadWriteOnce`, the Bitwarden deployment are fixed to one k8s node by using `nodeName: final-worker1`.

```
FROM busybox

RUN mkdir /bwdata

COPY . /bwdata
```

Listing 65: Dockerfile of Init-Container to copy bwdata into persistent volume

```
  identity:
    image: bitwarden/identity:2023.10.2
    container_name: bitwarden-identity
    restart: always
    volumes:
      - ../identity:/etc/bitwarden/identity
      - ../core:/etc/bitwarden/core
      - ../ca-certificates:/etc/bitwarden/ca-certificates
      - ../logs/identity:/etc/bitwarden/logs
    env_file:
      - global.env
      - ../env/uid.env
      - ../env/global.override.env
    networks:
      - default
      - public
```

Listing 66: Identity deployment in docker-compose.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bitwarden-identity
spec:
  ...
    spec:
      nodeName: final-worker1
      initContainers:
      - name: bitwarden-identity-init
        image: registry.internal.uia.no/ikt210-g-23h-skyinfrastruktur/labgroup8/final-project:
    init
        volumeMounts:
        - name: bitwarden-identity
          mountPath: /data
        command: ["/bin/sh","-c"]
        args: ["mv /bwdata/identity/* /data/"]
      containers:
      - name: bitwarden-identity
        image: bitwarden/identity:2023.10.2
        ...
        envFrom:
        - secretRef:
            name: global-env
        - secretRef:
            name: uid-env
        - secretRef:
            name: global-override-env
        ports:
        - containerPort: 5000
          name: web
        volumeMounts:
        - name: bitwarden-core
          mountPath: /etc/bitwarden/core
          ...
      volumes:
        - name: bitwarden-core
          persistentVolumeClaim:
            claimName: bitwarden-core
          ...
      imagePullSecrets:
      - name: registry-pull
```

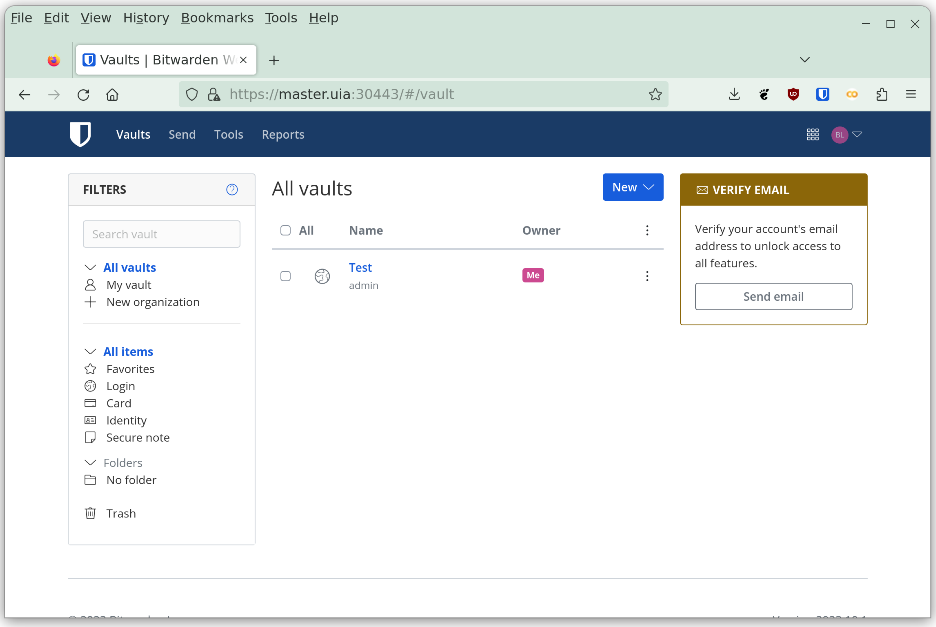
Listing 67: Translated Kubernetes Identity deployment Init-Container

All manifests are added to a kustomization.yaml which sets the bitwarden namespace. The following pods and services were deployed:

```
$ kubectl get pods -n bitwarden
NAME                                       READY   STATUS    RESTARTS   AGE
bitwarden-admin-5f97bd4558-bs8f6           1/1     Running   0          7m21s
bitwarden-api-7cfbdd8ddc-wm9bd             1/1     Running   0          7m22s
bitwarden-attachments-79bbd9c564-fgj4j     1/1     Running   0          7m22s
bitwarden-events-56f87fd469-sjtz4          1/1     Running   0          7m21s
bitwarden-icons-5f8f95546-g4wv4            1/1     Running   0          7m21s
bitwarden-identity-84bf5b546f-6bss5        1/1     Running   0          7m21s
bitwarden-mssql-69d6b685b6-858qw           1/1     Running   0          7m21s
bitwarden-nginx-79f8cdb899-s8znw           1/1     Running   0          7m21s
bitwarden-notifications-7c88dfb9cb-24zns   1/1     Running   0          7m21s
bitwarden-sso-7466ff8944-fxslv             1/1     Running   0          7m21s
bitwarden-web-749f4c9d57-4fdsm             1/1     Running   0          7m21s

$ kubectl get svc -n bitwarden
NAME            TYPE        PORT(S)          AGE
admin           ClusterIP   5000/TCP         7m22s
api             ClusterIP   5000/TCP         7m21s
attachments     ClusterIP   5000/TCP         7m21s
events          ClusterIP   5000/TCP         7m20s
icons           ClusterIP   5000/TCP         7m20s
identity        ClusterIP   5000/TCP         7m19s
mssql           ClusterIP   1433/TCP         7m19s
nginx-http      NodePort    8080:30080/TCP   7m18s
nginx-https     NodePort    8443:30443/TCP   7m18s
notifications   ClusterIP   5000/TCP         7m18s
sso             ClusterIP   5000/TCP         7m17s
web             ClusterIP   5000/TCP         7m17s
```

Listing 68: Deployed Bitwarden pods and services

Bitwarden could then be accessed over the Nginx reverse proxy over the NodePort 30443.


https://master.uia:30443/#/vault

All vaults

Verify your account's email address to unlock access to all features.

Figure 14: Accessing the Bitwarden web interface

2.4.2 Cryptpad

The Cryptpad Deployment was based on the docker-compose.yaml file available in the Cryptpad GitHub repository [13]. The compose file sets two environment variables specifying the main domain and a sandbox domain and one specifying the path to the config file. Additionally, some volumes are mounted for persistence and some ports are exposed. Since the cluster does not have a clean way to be reached from the internet and no access to an UiA internal DNS server is given, the domain names were specified on the accessing machine locally in the `/etc/hosts` file as shown in listing 69.

```
10.225.150.174 g8-cryptpad.uia
10.225.150.174 g8sb-cryptpad.uia
```

Listing 69: Setting domain names for Cryptpad access in local hosts file

For the domain names, a self-signed certificate was created using OpenSSL. It is important, that both domain names have the same certificate, which is why g8sb-cryptpad.uia was specified in the `subjectAltName`.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout key -out cert \
-subj "/C=CH/ST=Some/L=Place/O=UIA/OU=IT/CN=g8-cryptpad.uia" \
-addext "subjectAltName=DNS:g8sb-cryptpad.uia"
```

Listing 70: Creating certificates for the Cryptpad domains

Additionally Diffie-Hellman parameters are generated for the session key negotiation.

```
openssl dhparam -out dhparam.pem 4096
```

Listing 71: Generating Diffie-Hellman parametes

In the manifest file cryptpad.yaml, four PersistentVolumeClaims using the `rook-ceph-block` storage class (see example in listing 72) and a ClusterIP Service exposing the three ports used by the Cryptpad container image are created (see listing 73).

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cryptpad-datastore
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: rook-ceph-block
```

Listing 72: Cryptpad PVC example

```
apiVersion: v1
kind: Service
metadata:
  name: cryptpad-http
spec:
  type: ClusterIP
  selector:
    app: cryptpad
  ports:
  - port: 3000
    targetPort: 3000
    name: httpunsafe
  - port: 3001
    targetPort: 3001
    name: httpsafe
  - port: 3003
    targetPort: 3003
    name: websocket
```

Listing 73: Cryptpad Service configuration

The Cryptpad Deployment sets the three necessary environment variables, defines the three container ports 3000, 3001 and 3003 and mounts the four volumes. A ConfigMap mount for the config.js file was created using the `configMapGenerator` in kustomization.yaml (listing 76). This volume was however, not mounted in the end, since the default configurations are sufficient. It would allow to modify the configurations in the future though. The `fsGroup: 4001` key had to be set for Cryptpad to be able to create files in the volumes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cryptpad
spec:
  ...
    spec:
      securityContext:
        fsGroup: 4001
      containers:
      - name: cryptpad
        image: cryptpad/cryptpad:version-5.5.0
        imagePullPolicy: Always
       ...
        env:
        - name: CPAD_CONF
          value: /cryptpad/config/config.js
        - name: CPAD_MAIN_DOMAIN
          value: "https://g8-cryptpad.uia:30180"
        - name: CPAD_SANDBOX_DOMAIN
          value: "https://g8sb-cryptpad.uia:30180"
        ports:
        - containerPort: 3000
        - containerPort: 3001
        - containerPort: 3003
          protocol: TCP
        - name: cryptpad-blob
          mountPath: /cryptpad/blob
        ...
      volumes:
      - name: cryptpad-config
        configMap:
          name: cryptpad-config
      - name: cryptpad-blob
        persistentVolumeClaim:
          claimName: cryptpad-blob
      ...
```
Listing 74: Cryptpad Deployment

Cryptpad is run behind a Nginx reverse proxy, which uses the example configuration found on the Cryptpad repository [14], with some slight modifications. The `ssl_trusted_certificate` section has been removed, since we don't verify the chain of trust, the paths of the certificate, key and parameters are changed to `/ssl/key|cert|dhparam`, the default location was modified to point to the Cryptpad ClusterIP Service and a new location `/cryptpad_websocket` was added, which is necessary for Cryptpad to display all content. Listing 75 shows the configured locations.

```
    location / {
        proxy_pass            http://cryptpad-http:3000;
        proxy_set_header      X-Real-IP $remote_addr;
        proxy_set_header      Host $host;
        proxy_set_header      X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size  150m;

        proxy_http_version    1.1;
        proxy_set_header      Upgrade $http_upgrade;
        proxy_set_header      Connection "Upgrade";
    }

    location /cryptpad_websocket {
        proxy_pass            http://cryptpad-http:3003;
        proxy_set_header      X-Real-IP $remote_addr;
        proxy_set_header      Host $host;
        proxy_set_header      X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size  150m;

        proxy_http_version    1.1;
        proxy_set_header      Upgrade $http_upgrade;
        proxy_set_header      Connection "Upgrade";
    }
}
```
Listing 75: Nginx configuration file locations

To mount the certificate, key, parameters and configuration file in the Nginx pod ConfigMaps are used, which are generated in kustomization.yaml using a `configMapGenerator` (listing 76). The Nginx deployment manifest nginx.yaml mounts these ConfigMaps and exposes port 443 using a NodePort to port 30180.

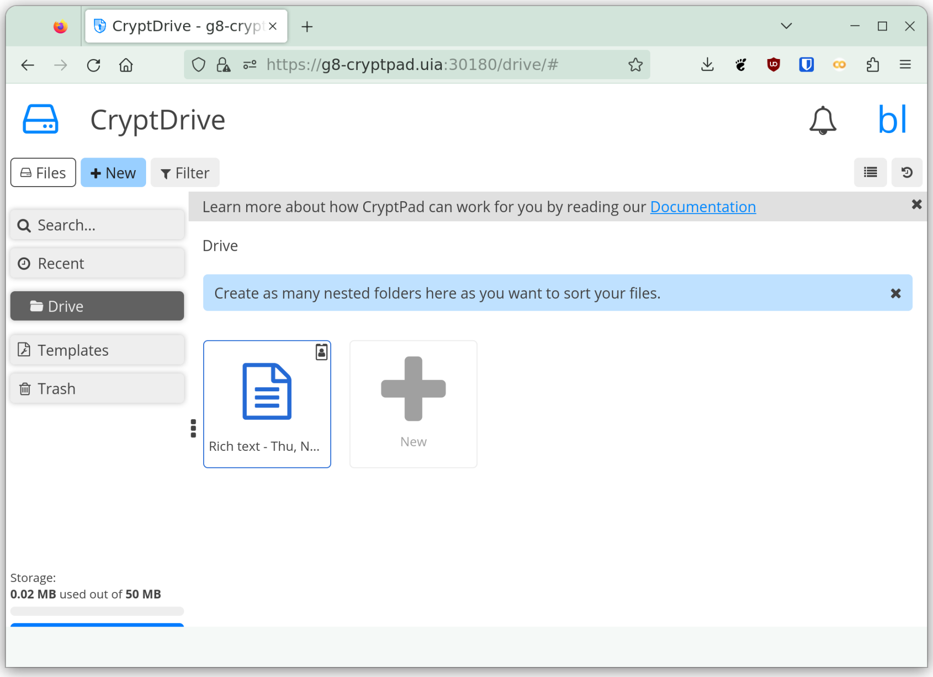
```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - "namespace.yaml"
  - "nginx.yaml"
  - "cryptpad.yaml"

namespace: cryptpad

configMapGenerator:
  - name: cryptpad-config
    behavior: create
    files:
    - config.js
  - name: nginx-config
    behavior: create
    files:
    - default.conf
  - name: nginx-cert
    behavior: create
    files:
    - ssl/cert
  - name: nginx-key
    behavior: create
    files:
    - ssl/key
  - name: nginx-dhparam
    behavior: create
    files:
    - ssl/dhparam.pem
```

Listing 76: Cryptpad Kustomization manifest

After deploying the system using `kubectl apply -k .` the following services and pods are created. As seen in figure 15 the Cryptpad web interface is now accessible in the browser under the URL `https://g8-cryptpad.uia:30180`. Important to note is, that this only works, when the two hostnames are added to `/etc/hosts`, as shown in listing 69 and the Firefox Browser is used. More to this in section 3.2.

```
$ kubectl get pods -n cryptpad
NAME                          READY   STATUS              RESTARTS   AGE
cryptpad-5467959547-8tnsl     0/1     ContainerCreating   0          43s
cryptpad-6d5495ffc4-d2z7f     1/1     Running             0          115s
nginx-6f976486f9-vn4k8        1/1     Running             0          114s

$ kubectl get svc -n cryptpad
NAME            TYPE        PORT(S)                      AGE
cryptpad-http   ClusterIP   3000/TCP,3001/TCP,3003/TCP   2m21s
nginx-http      NodePort    443:30180/TCP                2m20s
```

Listing 77: Deployed cryptpad pods and services



Learn more about how CryptPad can work for you by reading our Documentation

Drive

Create as many nested folders here as you want to sort your files.

Figure 15: Accessing the Cryptpad web interface

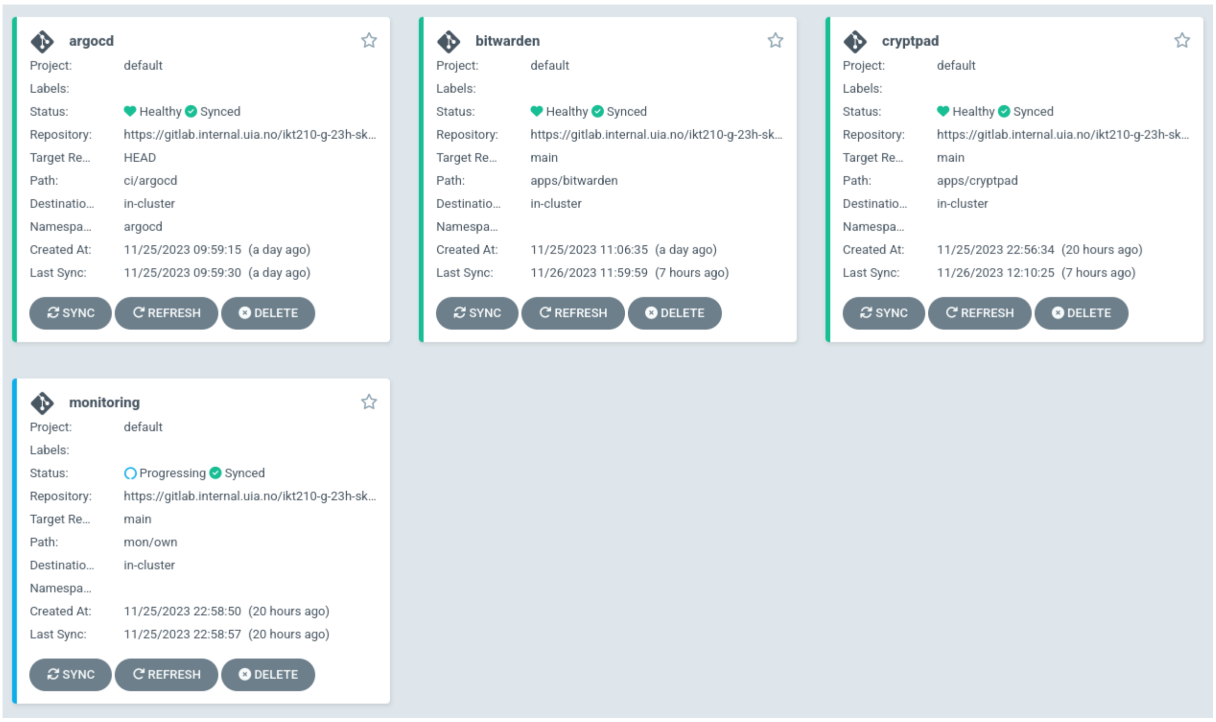
2.5 Additional configuration

2.5.1 Adding Deployments to ArgoCD

After finishing the deployments for Bitwarden, Cryptpad and the monitoring stack, they were all added to ArgoCD using the following commands:

```
argocd app create monitoring \
--repo https://gitlab.internal.uia.no/ikt210-g-23h-skyinfrastruktur/LabGroup8/final-project.git
    \
--path mon/own \
--dest-server https://kubernetes.default.svc \
--self-heal \
--sync-policy automated \
--sync-retry-limit 5 \
--revision main

argocd app set monitoring --auto-prune
```

Listing 78: Adding monitoring Deployment to ArgoCD



argocd

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://gitlab.internal.uia.no/ikt210-g-23h-sk...
Target Re...	HEAD
Path:	ci/argocd
Destinatio...	in-cluster
Namespa...	argocd
Created At:	11/25/2023 09:59:15 (a day ago)
Last Sync:	11/25/2023 09:59:30 (a day ago)

bitwarden

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://gitlab.internal.uia.no/ikt210-g-23h-sk...
Target Re...	main
Path:	apps/bitwarden
Destinatio...	in-cluster
Namespa...	
Created At:	11/25/2023 11:06:35 (a day ago)
Last Sync:	11/26/2023 11:59:59 (7 hours ago)

cryptpad

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://gitlab.internal.uia.no/ikt210-g-23h-sk...
Target Re...	main
Path:	apps/cryptpad
Destinatio...	in-cluster
Namespa...	
Created At:	11/25/2023 22:56:34 (20 hours ago)
Last Sync:	11/26/2023 12:10:25 (7 hours ago)

monitoring

Project:	default
Labels:	
Status:	Progressing Synced
Repository:	https://gitlab.internal.uia.no/ikt210-g-23h-sk...
Target Re...	main
Path:	mon/own
Destinatio...	in-cluster
Namespa...	
Created At:	11/25/2023 22:58:50 (20 hours ago)
Last Sync:	11/25/2023 22:58:57 (20 hours ago)

Figure 16: All Deployments added as ArgoCD apps

2.5.2 Security Configurations

It was attempted to adhere to the Kubernetes hardening guidelines by the NSA in the following ways. Where possible, containers were run as a different user than root if the software running inside the container allowed it. Secrets were encrypted using the Sealed Secrets package (unencrypted configs and secrets are still present in the repository for educational purposes). Additionally, each deployment was made in a separate namespace.

3 What problems the group encountered

3.1 Cilium CNI

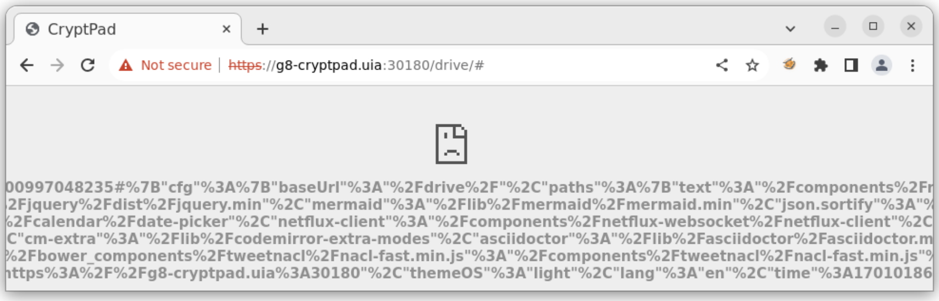
When deciding on the Kubernetes techstack, at first it was decided, that Cilium would be used for the CNI. However after installing it, the Cilium pods were never started. Looking at the logs, the error in listing 79 was shown. After some research a script was found, which checks the Linux kernel if all necessary flags are set for Cilium to function. This script showed the flags `CONFIG_BPF_JIT`, `CONFIG_FTRACE_SYSCALLS` and `CONFIG_KPROBE_EVENTS` missing. Since these flags need to be set, while compiling the kernel and could not be done at runtime, Cilium was dropped in favour of Flannel.

```
level=fatal msg="Load overlay network failed" error="program cil_from_overlay: replacing clsact
    qdisc for interface cilium_vxlan: operation not supported" interface=cilium_vxlan subsys=
    datapath-loader
```

Listing 79: Cilium Error

3.2 Cryptpad Browser Compatibility

After installing Cryptpad, everything worked fine in the Firefox browser. When trying to access the website from Chrome however, an error was displayed, as shown in figure 17. This seems to be a general problem multiple people have had [15]. Since it worked in Firefox and project time was nearing an end, this problem was not fixed.



```
00997048235#%7B"cfg"%3A%7B"baseUrl"%3A"%2Fdrive%2F"%2C"paths"%3A%7B"text"%3A"%2Fcomponents%2Fr
2Fjquery%2Fdist%2Fjquery.min"%2C"mermaid"%3A"%2Flib%2Fmermaid%2Fmermaid.min"%2C"json.sortify"%3A"%
%2Fcalendar%2Fdate-picker"%2C"netflux-client"%3A"%2Fcomponents%2Fnetflux-websocket%2Fnetflux-client"%2C
C"cm-extra"%3A"%2Flib%2Fcodemirror-extra-modes"%2C"asciidoctor"%3A"%2Flib%2Fasciidoctor%2Fasciidoctor.m
%2Fbower_components%2Ftweetnacl%2Fnacl-fast.min.js"%3A"%2Fcomponents%2Ftweetnacl%2Fnacl-fast.min.js"%
nttps%3A%2F%2Fg8-cryptpad.uia%3A30180"%2C"themeOS"%3A"light"%2C"lang"%3A"en"%2C"time"%3A17010186
```

Figure 17: Cryptpad error in Chorme browser

3.3 Thanos Store Setup

When configuring Thanos, the Store system was also intended to be set up. when deploying it, an error was shown, that no files could be created, as shown in listing 80. This could have something to do with the connection credentials of the bucket not being correctly mounted to the Store or a process not having the correct UID to access the files. Since time was running out, this error could not be fixed. The corresponding manifests are however still in the repository for education purposes.

```
ts=2023-11-26T17:18:36.150291383Z caller=main.go:135 level=error err="mkdir /var/thanos/store/
    meta-syncer: permission denied\nmeta fetcher\nmain.runStore\n\t/app/cmd/thanos/store.go:347\
    nmain.registerStore.func1\n\t/app/cmd/thanos/store.go:226\nmain.main\n\t/app/cmd/thanos/main
    .go:133\nruntime.main\n\t/usr/local/go/src/runtime/proc.go:267\nruntime.goexit\n\t/usr/local
    /go/src/runtime/asm_amd64.s:1650\npreparing store command failed...."
```

Listing 80: Thanos Store error

4 Reflection

This task was done with an additional time constraint, since I am going on holidays on the 28. November and did not want to do the project during that time. Overall the project worked out pretty smoothly. I sank a little too much time into Cilium in the beginning, but after I switched to Flannel, I was able to make quick progress.

Setting up the monitoring was quite the chore and since the requirements in the project task description are very vague. I don't know if I have done too little or too much. This is a general complaint I have: Please make this projects task description more specific. It wasn't clear, if I should add the two applications (Cryptpad and Bitwarden) to the monitoring, or if it was enough, to just configure the monitoring itself and proof that it works. The same thing with the Sealed Secrets and the ArgoCD Image Updates. I decided to use the Sealed Secrets, because it made sense in the context, but I saw no real use case for Image Updater, which is why I only installed it.

Installing Bitwarden worked out better than I expected, since I could pretty much just translate the docker-compose.yaml found in the Bitwarden repository into Kubernetes manifests. I struggled a lot with Cryptpad, since I wasn't sure how to set it up, in developement or production mode. I ended up with functional deployment by using entries in the local hosts file.

Since the requirement was given to use Kustomize, but I didn't really find that much of a use case for it, I only used it to set the namespaces and sometimes to generate some ConfigMaps. Maybe in a more elaborate monitoring setup it could also be useful to set a label, which defines that a resource should be monitored.

As a general reflection of the course, I think I learned the most during the exercises and the project. The lectures were sadly not very helpful. An even bigger focus on examples during lectures may help. Giving soft deadlines for the exercises was a nice guideline, to not fall to far behind. It would have been very helpful to receive feedback on the exercises earlier. Since I got my first feedback only after I already handed in another two assignments, I wasn't able to implement that feedback.

References

[1] “Flannel.” (Nov. 1, 2023), [Online]. Available: `https://github.com/flannel-io/flannel` (visited on 11/20/2023).

[2] “Rook.” (Nov. 24, 2023), [Online]. Available: `https://github.com/rook/rook.git` (visited on 11/24/2023).

[3] “Installation - argo CD image updater.” (2023), [Online]. Available: `https://argocd-image-updater.readthedocs.io/en/stable/install/installation/` (visited on 11/21/2023).

[4] Bitnami. “Sealed-secrets.” (2023), [Online]. Available: `https://github.com/bitnami-labs/sealed-secrets` (visited on 11/21/2023).

[5] “Prometheus operator getting started,” Prometheus Operator. (2023), [Online]. Available: `https://prometheus-operator.dev/docs/user-guides/getting-started/` (visited on 11/24/2023).

[6] “How to setup prometheus node exporter on kubernetes.” (Apr. 6, 2021), [Online]. Available: `https://devopscube.com/node-exporter-kubernetes/` (visited on 11/24/2023).

[7] “Node exporter full,” Grafana Labs. (2023), [Online]. Available: `https://grafana.com/grafana/dashboards/1860-node-exporter-full/` (visited on 11/26/2023).

[8] kube-prometheus. “Kube-prometheus manifests.” (2023), [Online]. Available: `https://github.com/prometheus-operator/kube-prometheus/tree/main/manifests` (visited on 11/26/2023).

[9] Prometheus Operator. “Blackbox exporter,” Prometheus Operator. (Mar. 8, 2021), [Online]. Available: `https://prometheus-operator.dev/docs/kube/blackbox-exporter/` (visited on 11/26/2023).

[10] “Prometheus blackbox exporter,” Grafana Labs. (2023), [Online]. Available: `https://grafana.com/grafana/dashboards/7587-prometheus-blackbox-exporter/` (visited on 11/26/2023).

[11] “Thanos quick start.” (2023), [Online]. Available: `https://thanos.io/v0.12/thanos/quick-tutorial.md/` (visited on 11/25/2023).

[12] “Install and deploy - linux — bitwarden help center,” Bitwarden. (2023), [Online]. Available: `https://bitwarden.com/help/install-on-premise-linux/` (visited on 11/22/2023).

[13] cryptpad. “Cryptpad repository.” (Nov. 24, 2023), [Online]. Available: `https://github.com/cryptpad/cryptpad` (visited on 11/26/2023).

[14] cryptpad. “Cryptpad nginx example default.conf.” (Nov. 24, 2023), [Online]. Available: `https://raw.githubusercontent.com/cryptpad/cryptpad/main/docs/example.nginx.conf` (visited on 11/26/2023).

[15] jbhanks. “Cryptpad different results in different browsers.” (Jan. 13, 2021), [Online]. Available: `https://github.com/cryptpad/cryptpad/issues/673` (visited on 11/26/2023).

Listings

List of Figures