- acend gmbh

Labs

- acend gmbh

1. Installation and configuration

Installing the predecessor, OpenShift 3, meant that you had to configure everything in advance using an Ansible inventory. Nearly every aspect of the cluster had to be configured with variables, most could be changed afterwards, but not all. And some did not work well with others, leading to a hard-to-do installation when you had no experience or a complex setup.

This changed in an essential way with OpenShift 4. Instead of configuring everything in advance, you start with the installation of a basic cluster. The cluster can then be configured using Kubernetes resources, mostly custom resources, sometimes configmaps or secrets.

In this first lab section of the OpenShift 4 Operations training, you are going to install your own OpenShift 4 cluster, configure it and add some additional, important components.

1.1 Installation

In this lab, you are going to install an OpenShift 4 cluster on AWS.

Task 1.1.1: Preparing the environment

Using the information provided by your trainer, ssh into your bastion host and change into the `ocp4-ops` directory.

It is best practice to create a timestamped directory for each cluster installation, for example:

```
```

Task 1.1.2: Customizing the installation

First, we need an SSH keypair. The public key will be used to grant us access to the OpenShift machines we are going to create. Either use an existing keypair or create a new one by executing:

```
```

Note the SSH public key.

Also note the pull secret available in `~/ocp4-ops/pull-secret` on your bastion host. The pull secret is used by the installer to pull the necessary images from the Red Hat registry.

Change into the previously created directory:

```
```

Now, create a file called `install-config.yaml` . Add the content from the following box and also add the noted SSH public key and pull secret.

Then, change the values of `metadata.name` and `platform.aws.userTags.user` to reflect your username +username+.

```yaml
apiVersion: v1
baseDomain: openshift.ch
compute:
- name: worker
  platform:
    aws:
      type: m5.2xlarge
      zones:
      - us-east-1b
  replicas: 3
controlPlane:
  name: master
  platform:
    aws:
      type: m6i.xlarge
  replicas: 3
metadata:
  name: <username>-ops-training #FIXME
networking:
  networkType: OVNKubernetes
platform:
  aws:
    region: us-east-1
    userTags:
      Application: ocp4-ops
      Customer: acend
      Owner: <username> #FIXME
capabilities:
  baselineCapabilitySet: None
  additionalEnabledCapabilities:
  - MachineAPI
  - marketplace
  - openshift-samples
  - Console
  - Insights
  - Storage
  - CSISnapshot
  - NodeTuning
  - ImageRegistry
publish: External
pullSecret: '{"auths": ...}' #FIXME
sshKey: ssh-ed25519 AAAA... #FIXME
```

Note

This file is also available at https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/install-config.yaml .

Backup the file `install-config.yaml` , since it will be consumed during the installation process:

```
mkdir backup
cp install-config.yaml backup/install-config.yaml
```

Task 1.1.3: Deploying the cluster

Now you are ready to create your own cluster:

```
openshift-install create cluster --log-level debug
```

Note

The installation usually takes about 30 minutes to complete. In the meantime, have a look at the OpenShift installation documentation for a list of available parameters.

Example output:

```
...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/home/myuser/install_dir/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.mycluster.example.com
INFO Login to the console with user: "kubeadmin", and password: "4vYBz-Ee6gm-ymBZj-Wt5AL"
INFO Time elapsed: 36m22s
```

Task 1.1.4: Verifying the installation

By setting the environment variable `KUBECONFIG` you can provide credentials to the OpenShift CLI (`oc`):

```
export KUBECONFIG=$HOME/ocp4-ops/$(date +"%Y-%m-%d")/auth/kubeconfig
```

You will now check whether you can log in to the cluster with the `kubeadmin` credentials:

```
oc whoami --show-server
```

The output should look like this:

```
https://api.+username+-ops-training.openshift.ch:6443
```

The cluster operators are a good indicator whether the cluster is healthy or not:

```
oc get clusteroperators
```

Example output:

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
authentication                             4.7.6     True        False         False      105m
baremetal                                  4.7.6     True        False         False      3d2h
cloud-credential                           4.7.6     True        False         False      3d3h
cluster-autoscaler                         4.7.6     True        False         False      3d2h
config-operator                            4.7.6     True        False         False      3d2h
console                                    4.7.6     True        False         False      113m
csi-snapshot-controller                    4.7.6     True        False         False      107m
dns                                        4.7.6     True        False         False      111m
etcd                                       4.7.6     True        False         False      3d2h
image-registry                             4.7.6     True        False         False      98m
ingress                                    4.7.6     True        False         False      3d2h
insights                                   4.7.6     True        False         False      3d2h
kube-apiserver                             4.7.6     True        False         False      3d2h
kube-controller-manager                    4.7.6     True        False         False      3d2h
kube-scheduler                             4.7.6     True        False         False      3d2h
kube-storage-version-migrator              4.7.6     True        False         False      106m
machine-api                                4.7.6     True        False         False      3d2h
machine-approver                           4.7.6     True        False         False      3d2h
machine-config                             4.7.6     True        False         False      103m
marketplace                                4.7.6     True        False         False      107m
monitoring                                 4.7.6     True        False         False      3h47m
network                                    4.7.6     True        False         False      3d2h
node-tuning                                4.7.6     True        False         False      146m
openshift-apiserver                        4.7.6     True        False         False      105m
openshift-controller-manager               4.7.6     True        False         False      3d2h
openshift-samples                          4.7.6     True        False         False      146m
operator-lifecycle-manager                 4.7.6     True        False         False      3d2h
operator-lifecycle-manager-catalog         4.7.6     True        False         False      3d2h
operator-lifecycle-manager-packageserver   4.7.6     True        False         False      108m
service-ca                                 4.7.6     True        False         False      3d2h
storage                                    4.7.6     True        False         False      107m
```

1.2 Cluster inspection

Before diving headfirst into configuring the freshly set-up cluster, let's first take a look at certain features. As we installed the OpenShift cluster on AWS, some components have already been configured for us to use right away, amongst them:

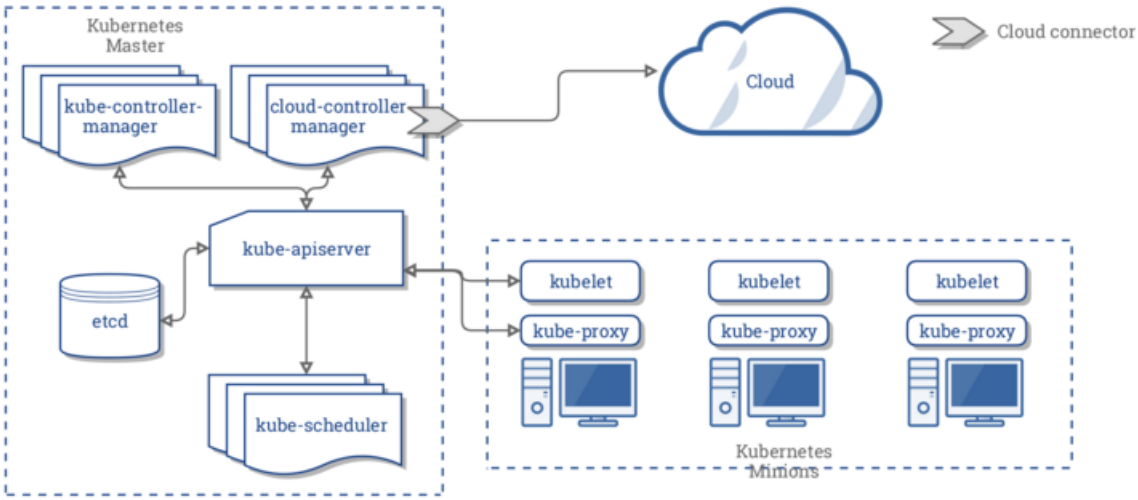
- An AWS cloud provider, making it possible for controllers to, e.g., create and update AWS load balancers and do node lifecycle management
- A storage class to provision AWS Elastic Block Store (EBS) volumes

Cloud provider

In order to understand what a cloud provider is or does, it's easiest to look at the description of the AWS cloud provider :

> The AWS cloud provider provides the interface between a Kubernetes cluster and AWS service APIs. This project allows a Kubernetes cluster to provision, monitor and remove AWS resources necessary for operation of the cluster.

The cloud provider consists of different components which then make use of the mentioned interface (picture source: kubernetes.io):



OpenShift supports a wide range of platforms in which it can integrate this way. However, depending on the installation method and platform used, the level of integration can vary greatly. AWS was the first platform to be supported by OpenShift 4 and hence offers one of the most complete integrations.

Refer to the following resources if you're interested in more details:

- Cloud Controller Manager
- Cloud Controller Manager Administration
- cloud-provider on github.com
- cloud-provider-aws on github.com
- The Future of Cloud Providers in Kubernetes (blogpost from April 17,2019)

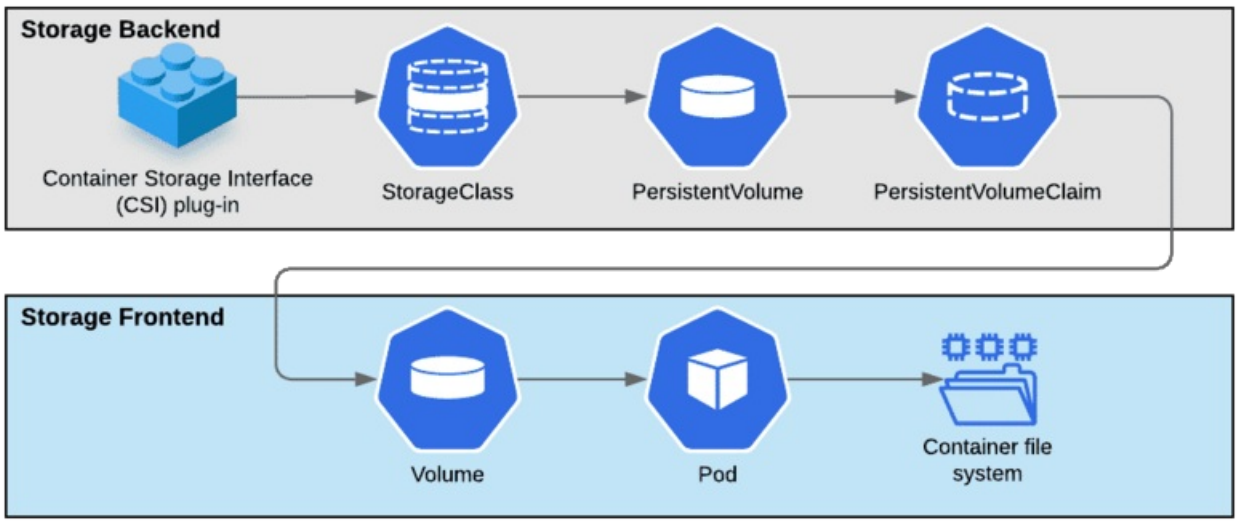
Storage

Storage in Kubernetes represents a significant part of the ability to save application state within the cluster.

In the early days of OpenShift 3, administrators had to manually create PersistentVolume resources. These pointed to an actual volume from a storage provider (such as an NFS server) and could then be claimed by users via PersistentVolumeClaims. If the claim could be fulfilled by one of the existing PersistentVolumes, it was bound to it and could then be used inside the container.

Kubernetes and OpenShift matured and not long after were able to provide the means to automatically or, as it is also called, dynamically create PersistentVolume resources. This includes provisioning a volume (or similar) on the storage provider itself and is usually triggered by the creation of PersistentVolumeClaims.

In order to visualize the whole workflow, consider the following diagram:



We already know most parts of this diagram:

- A PersistentVolume is bound to a PersistentVolumeClaim
- A successfully bound PersistentVolumeClaim allows the usage of a volume in a Pod
- The Pod declaration defines which volume is used inside what container and at what path

What's new is the StorageClass and CSI plug-in. A StorageClass represents the "classes" or "profiles" of storage a cluster offers. It especially contains the information that is used when a PersistentVolume belonging to a class needs to be dynamically provisioned. Kubernetes then makes the appropriate calls via its Container-Storage Interface (CSI) to a storage provisioner. The storage provisioner in turn creates the requested volume.

Task 1.2.1: Storage class inspection

Have a closer look at the `gp3-csi` storage class.

As you can see, the `provisioner` field is set to `ebs.csi.aws.com`. This means the AWS EBS provisioner will automatically create and delete EBS volumes when a user creates a PersistentVolumeClaim referencing the `gp3-csi` storage class.

Each provisioner offers different functionality, exposed as parameters inside the storage class. The EBS provisioner allows for the encryption of the data inside the volumes, controlled via the `parameters.encrypted` field, which is already set. Also set is the field `parameters.type`. By exposing this field the provisioner makes it possible for us to create additional storage classes, which would for example allow the use of slower, cheaper disks for workload that does not need high performance storage (e.g. backup). The provisioner supports even more parameters.

The remaining standard fields tell the provisioner to delete volumes when a PersistentVolumeClaim is deleted on OpenShift (because the `reclaimPolicy` is set to `Delete`) and to wait with binding the PersistentVolumeClaim to an actual PersistentVolume until a Pod starts using it (via the `volumeBindingMode` field).

Network and architecture

If you are interested in the underlying network and architecture of an AWS OpenShift cluster, have a look at AWS Architecture Blog's article "Architecture Patterns for Red Hat OpenShift on AWS".

1.3 Configuration

After the initial installation, we can start configuring the cluster. Because of the extended use of operators, nearly everything in OpenShift 4 is configured via custom resources. In some cases configmaps or templates are used, but this is clearly the minority.

Most of the configuration is done post-installation and can be used to better adapt the cluster to its environment or change its default behaviour. We are going to configure our own kubelet arguments and change the default project template in order to automatically create some NetworkPolicy resources.

Task 1.3.1: Configure kubelet arguments

OpenShift lets us change the kubelet configuration via the custom resource `KubeletConfig`. But why change it at all?

The default kubelet configuration doesn't reserve any memory or cpu for the kubelet or the operating system itself. It also defines an eviction threshold which, in some cases, is too low for the kubelet to react fast enough.

Configure a `KubeletConfig` resource each for master and worker nodes defining the following parameters:

- Set a hard eviction threshold to 500Mi available memory
- Set the reserved kubelet resources to 250m cpu and 1Gi memory
- Set the reserved system resources to 250m cpu and 1Gi memory

You can find relevant information on the OpenShift documentation pages here and here .

Change the masters' configuration to this:

And the worker nodes' configuration to this:

> **Note**
> Instead of defining specific `systemReserved` values, you could also simply supply a line defining `.spec.autoSizingReserved: true`. That way OpenShift calculates recommended values for you.

> **Note**
> These resource files are also available at https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/kubeletconfig_master.yaml and https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/kubeletconfig_worker.yaml , respectively.

There are multiple possible ways to apply these configuration resources to the cluster. We are going to use

- acend gmbh

one of the quickest methods and apply via URL:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/kubeletconfig_master.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/kubeletconfig_worker.yaml
```

To learn more about kubelet configuration, check the Kubernetes and OpenShift documentation .

Task 1.3.2: Generate the default project template

When creating a new project using `oc new-project` , OpenShift instantiates a template using the information provided. We can modify this template if we want to alter it or add more resources, such as NetworkPolicies or LimitRanges.

And this is exactly what you are going to do first of all: Generate the default template according to the documentation and change its name to `project-request-extended` .

> **Note**
> You might want to change the filename the template is written to to something more meaningful than what is used in the documentation (`template.yaml`). A good practice is to use a naming convention such as `<resource type>_<resource name>.yaml`. This allows you to quickly find the resource definition you're looking for based on the filename in case it is part of a larger collection.

Simply execute the command from the documentation with a slight change to the filename (according to the tip above):

```
oc adm create-bootstrap-project-template -o yaml > template_project-request-extended.yaml
```

Open the file using your favorite editor and change the name:

```
kind: Template
metadata:
  creationTimestamp: null
  name: project-request-extended
objects:
- 0 ~
```

Task 1.3.3: Network policies

Now, what you usually want to ensure first on a new cluster is that the different namespaces are isolated from each other in terms of network traffic. User A's pods in project A should not be able to directly talk to user B's pods in project B. Except of course for certain use cases, but thanks to the flexibility of network policies, we can easily define a sane default and change it on a namespace-basis if necessary.

Add the necessary network policies to the default project template. A sane default is already provided by Red Hat in its documentation .

Your file should now look like this:

> **Note**
> For your convenience, we also provide a file containing all network policies at https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/networkpolicies.yaml .

- acend gmbh

> **Note**
>
> Pay extra attention to the additional `metadata.namespace` fields added to the NetworkPolicy resources. If you don't add them, the network policies might end up being created in another namespace.

> **Note**
>
> Also note the additional namespace definition (`openshift-config`) at the top of the resource definition. This is to make sure the template is going to be created in the correct namespace where OpenShift will be looking for it.

Task 1.3.4: Limit range

Limit ranges are very important in keeping the resource usage on the cluster in check. In this training, we assume you already know about limit ranges. If not, the documentation provides a good foundation.

A good starting point could look as follows:

Add the LimitRange resource to your default project template.

Your project template should now look like this:

- acend gmbh

Task 1.3.5: Configure the template

The only thing missing now is to configure the cluster to use the new template.

Create the template and configure all necessary resources so new projects use our custom template.

> **Note**
> You might want to have another look at the different documentation pages so you don't forget anything.

When you think everything is set up, create a new project and check that the NetworkPolicy and LimitRange resources were created automatically.

Create the template on the cluster by either using your own template:

```
oc apply -f template_project-request-extended.yaml
```

Or use our provided solution:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/master/content/en/docs/1.3/resources/template_project-request-extended.yaml
```

And finally, we need to configure the Project custom resource:

```
oc patch project.config.openshift.io cluster --type=merge --patch '{"spec": {"projectRequestTemplate": {"name": "project-request-extended"}}}'
```

Task 1.3.6: Console URL

You probably already noticed the quite unwieldy console URL https://console-openshift-console.apps.+username+-ops-training.openshift.ch . It's among a number of other route hostnames that are created by OpenShift this way by default.

Because the console URL is the one we probably will use the most, we are going to simplify it. Find the appropriate instructions in OpenShift's documentation and adapt the route name.

- acend gmbh

The proper way is to edit the ingress config resource:

```
oc edit ingress.config.openshift.io cluster
```

Append the `componentRoutes` part so your resource looks similar to the example below:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  componentRoutes:
    - name: oauth-ui
      namespace: openshift-console
      hostname: apps.training.openshift.ch
      servingCertKeyPairSecret:
        name: training-openshift-ch
  [...]
```

1.4 Additional components

In this lab, you will install additional components that are commonly used in production.

Task 1.4.1: Install cert-manager

cert-manager massively simplifies certificate management for Kubernetes. It provides easy to use tools to issue, manage and automatically renew certificates and supports the ACME protocol. This allows us to easily and automatically get certificates signed by Let's Encrypt for our cluster.

Install cert-manager as an Operator using the OperatorHub:

- In the **Administrator** view, navigate to **Operators** -> **OperatorHub**
- Enter **cert-manager** into the filter box
- Select the **cert-manager Operator for Red Hat OpenShift** and click **Install**
- Leave everything as it is, except that you set the **Update approval** to **Manual**
- Click **Install**
- As soon as it appears, approve the install plan presented to you by clicking **Approve**
- Check for the existence of the `cert-manager-operator` pod

```
oc --namespace cert-manager-operator get pods
```

- Also, the Operator should already have created at least the webhook and cainjector pods in the `cert-manager` namespace

```
oc --namespace cert-manager get pods
```

The Operator will also automatically create a `CertManager` custom resource named `cluster`. In order to use DNS01 challenges on OpenShift clusters installed on AWS, we need to add some configuration to cert-manager's configuration :

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
spec:
  controlPlaneConfig:
    overrideArgs:
      - '--dns01-recursive-nameservers-only'
      - '--dns01-recursive-nameservers=1.1.1.1:53'
```

This effectivley adds these two parameters to the cert-manager deployment. They are needed because of AWS' different DNS zones for private and public resolution. This way cert-manager knows that, in order to check for the DNS01 challenge, it needs to call a public DNS server.

The last piece of configuration missing are the credentials. On supported infrastructure providers, you can simply get these credentials via the Cloud Credential Operator. As AWS is supported, we're going to use this mechanic :

- Create a CredentialsRequest resource YAML file named `credentialsrequest_cert-manager.yaml` with:

- acend gmbh

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
        - "route53:ChangeTagsForResource"
        - "route53:GetChange"
        - "route53:ListHostedZones"
        - "route53:ListTagsForResource"
      effect: Allow
      resource: "arn:aws:route53:::change/*"
    - action:
        - "route53:ChangeResourceRecordSets"
        - "route53:ListResourceRecordSets"
      effect: Allow
      resource: "arn:aws:route53:::hostedzone/*"
    - action:
        - "route53:ListHostedZonesByName"
      effect: Allow
      resource: "*"
  secretRef:
    name: aws-creds
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

- Create it:

```
oc create -f credentialsrequest_cert-manager.yaml
```

- Update the subscription object for cert-manager Operator for Red Hat OpenShift by running the following command:

```
oc -n openshift-cert-manager-operator patch subscription openshift-cert-manager-operator --type=merge -p '{"spec":{"config":{"env":[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"aws-creds"}]}}}'
```

- Verify that the cert-manager controller pod shows a volume and a mountPath entry each for the secret called `aws-creds`

```
oc -n cert-manager get pod -l 'app.kubernetes.io/component=controller' -o yaml
```

- You are looking for the following entries:

```
...
    name: cloud-credentials
    secret:
      secretName: aws-creds
...
    volumeMounts:
    - mountPath: /.aws
      name: cloud-credentials
...
```

Now you can create the `ClusterIssuer` resource which is supplied as a file inside the directory `~/ocp4-ops/resources/cert-manager/` .

```
oc create -f ~/ocp4-ops/resources/cert-manager/clusterissuer_letsencrypt-production.yaml
```

Verify that the `ClusterIssuer` is ready to issue certificates:

```
oc get clusterissuer letsencrypt-production
```

Look for column `READY` to be true:

```
NAME                     READY   AGE
letsencrypt-production   True    17s
```

Task 1.4.2 Replace the ingress controller certificate

After your `ClusterIssuer` is ready, you can request a wildcard certificate to be used on the Ingress Controller

for the default subdomain `apps.+username+-ops-training.openshift.ch` .

Create a file named `certificate_ingress.yaml` and fill in the following content, replacing the placeholder `<username>` with your actual username `+username+` .

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: cert-ingress
  namespace: openshift-ingress
spec:
  commonName: *.apps.<username>-ops-training.openshift.ch
  dnsNames:
  - apps.<username>-ops-training.openshift.ch
  issuerRef:
    kind: ClusterIssuer
    name: letsencrypt-production
  secretName: cert-ingress
```

Now apply the file.

```
oc apply -f certificate_ingress.yaml
```

It will take around 90 seconds for the certificate to be ready. Check with:

```
oc -n openshift-ingress get certificate
```

You’re looking for the column `READY` to be `True` :

```
NAME           READY   SECRET       AGE
cert-ingress   True    cert-ingress   4d
```

Update the `IngressController` configuration according to the documentation in order to use the certificate.

```
oc patch ingresscontroller/default \
  --type=merge -p '{"spec":{"defaultCertificate": {"name": "cert-ingress"}}}' \
  -n openshift-ingress-operator
```

After the router pods have all been replaced, the console URL should present a valid certificate. Have a look inside the `openshift-console` namespace to see the state of the pods.

```
oc -n openshift-console get pods
```

Task 1.4.3 Replace the API certificate

The default API server certificate is issued by an internal OpenShift Container Platform cluster CA. Clients outside of the cluster will not be able to verify the API server’s certificate by default. This certificate can be replaced by one that is issued by a CA that clients trust.

Create a file named `certificate_api.yaml` and fill in the following content.
Adjust the paramaters `commonName` and `dnsNames` to match your cluster name.

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: cert-api
  namespace: openshift-config
spec:
  commonName: api.<username>-ops-training.openshift.ch
  dnsNames:
  - api.<username>-ops-training.openshift.ch
  issuerRef:
    kind: ClusterIssuer
    name: letsencrypt-production
  secretName: cert-api
```

- acend gmbh

```
oc apply -f certificate_api.yaml
```

Again, check if the certificate is ready, then patch the API server to use the certificate .

```
oc patch apiserver cluster \
    --type=merge -p \
    '{"spec":{"servingCerts": {"namedCertificates":
      [{"names": ["api.<cluster-name>.training.openshift.acend.ch"],
      "servingCertificate": {"name": "tls-cert"}}]}}}'
```

Wait for the `kube-apiserver` operator to detect the configuration change and apply roll it out.

Since the `kubeconfig` file you have been working with so far contains the CA of the self-signed certificate, the newly created certificate cannot be validated against this CA:

```
$ oc whoami
Unable to connect to the server: x509: certificate signed by unknown authority
```

Open the kubeconfig file with your favourite editor (e.g. `vim $KUBECONFIG`) and remove the `certificate-authority-data` line under `clusters` .

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0...                                    # delete this line
    server: https://api.<cluster-name>.training.openshift.acend.ch:6443
  name: api-<cluster-name>-training-openshift-acend-ch:6443
```

Check again:

```
$ oc whoami
system:admin
```

Task 1.4.4 Install OADP

OpenShift API for Data Protection , or short OADP, is a tool to back up and restore Kubernetes resources.

> **Note**
> S3 buckets were already created for you to use as backup locations.

OADP is provided as an Operator, you will therefore install it using the OperatorHub:

- Navigate to **Operators** -> **OperatorHub**
- Search for **OADP**
- Select the **OADP Operator** provided by the Red Hat catalog (indicated by the **Red Hat** tag in the upper right of the box)
- Click **Install**
- Leave everything as it is, except that you set the **Update approval** to **Manual**
- Click **Install**
- As soon as it appears, approve the install plan presented to you by clicking **Approve**

What's left is to configure OADP:

- Create a secret containing the access credentials for OADP to access the S3 bucket:

- acend gmbh

```
oc --namespace openshift-adp create secret generic cloud-credentials --from-file cloud=/root/ops/resources/aws/credentials
```

- Create a `DataProtectionApplication` manifest as follows, named `dataprotectionapplication_ops-training-aws.yaml`, replacing the placeholder `<username>` with your actual username `+username+`:

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: ops-training-aws
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - openshift
        - aws
      resourceTimeout: 10m
    restic:
      enable: false
  podAnnotations:
    sidecar.istio.io/inject: "false"
  backupLocations:
    - velero:
        provider: aws
        default: true
        objectStorage:
          bucket: cust-name-ops-training-backup-volume
          prefix: velero
        config:
          region: us-east-1
          profile: "default"
        credential:
          key: cloud
          name: cloud-credentials
```

- Lastly, apply the manifest

```
oc create -f dataprotectionapplication_ops-training-aws.yaml
```

In one of the next chapters, you will learn how to use OADP for scheduled backups of cluster resources.

1.5 Uptime app

In this lab you're going to deploy an application on your freshly-installed OpenShift cluster.

The application's availability will be monitored and recorded by an external monitoring system. Your main goal for the rest of this training is to keep this application up and running, no matter what you do.

> **Note**
> At any time you can view the availability of your cluster console and the uptime app on this dashboard. Credentials to login will be provided by your trainers.

Uptime app

The application you're going to install is a simple Python app. It's going to be scaled to 2 replicas in order to achieve high availability.

Task 1.5.1: Deploy the uptime app

First you need to create a new project named `uptime-app-prod` for the application.

```
oc new-project uptime-app-prod
```

Next, you're going to create the following resources:

- Deployment
- Service
- Route

To deploy these resources, you can simply execute the following command:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-training/basic/webbackend/docs/01/resources/uptime-app.yaml -n uptime-app-prod
```

The deployed application should now be available at https://uptime-app-uptime-app-prod.apps.+username+-ops-training.openshift.ch .

With 2 replicas and the Pod anti-affinity , which will make sure all 2 Pods are never deployed on the same underlying node, we have a good setup for the training to continue.

1.6 Backup

In this lab, you will create scheduled backups for the most important cluster components.

Task 1.6.1: Create user workload backups

First, check the `backupStorageLocation` 's health and name:

```
oc -n openshift-adp get backupstoragelocations.velero.io
```

The output should show you that you've got `backupStorageLocation` resource named `default` with `PHASE` `Available` . If the `PHASE` shows something else, there's most probably a permissions or typo problem in your `dataProtectionApplication` resource.

```
NAME      PHASE       LAST VALIDATED   AGE   DEFAULT
default   Available   4s               12m   true
```

Now, create a daily backup of the resources in namespace `uptime-app-prod` with a lifetime of 5 days by creating the following `Schedule` manifest:

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: daily-backup-uptime-app-prod
  namespace: openshift-adp
spec:
  schedule: "0 1 * * *"
  template:
    ttl: 120h
    includedNamespaces:
    - uptime-app-prod
    volumeConfig:
```



```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/01/resources/schedule_daily-backup-uptime-app-prod.yaml
```

In order to test it, let's trigger a manual backup. We are using the `velero` cli for this which is pre-installed on your bastion host:

```
velero -n openshift-adp backup create --from-schedule daily-backup-uptime-app-prod
```

When the backup is completed, have a look at its status part:

```
oc -n openshift-adp describe backup | grep -A 9 Status
```

The output should look similar to this:

```
Status:
  Completion Timestamp:  2021-04-12T14:04:11Z
  Expiration:            2021-04-12T15:04:03Z
  Format Version:        1.1.0
  Phase:                 Completed
  Progress:
    Items Backed Up:  51
    Total Items:      51
  Start Timestamp:    2021-04-12T14:04:03Z
  Version:            1
```

Task 1.6.2 Create etcd backup

To be able to restore the cluster in case of a disaster, you will create a scheduled `etcd` backup.

Create a new project named `training-infra-etcd-backup`.

```
oc new-project training-infra-etcd-backup
```

Since `etcd` is running on the control plane nodes, we need to make sure the cronjob's pods that are creating the `etcd` snapshots are running on one as well. We can do this by annotating the namespace with the corresponding node selector:

```
oc patch namespace training-infra-etcd-backup -p \
  '{"metadata": {"annotations": {"openshift.io/node-selector": "node-role.kubernetes.io/master=""}}}'
```

For the cronjob pod to be able to access the `etcd` files, we need to create a service account with additional permissions by attaching a custom `SecurityContextConstraints` (SCC) policy:

- Service account:

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: etcd-backup
```

- SCC:

```
kind: SecurityContextConstraints
metadata:
  name: privileged-etcd-backup
  annotations:
    kubernetes.io/description: "privileged allows access to all privileged and host
      features and the ability to run as any user, any group, any fsGroup, and with
      any SELinux context.  WARNING: this is the most relaxed SCC and should be used
      only for cluster administration. Grant with caution."
priority: null
allowPrivilegedContainer: true
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegeEscalation: true
allowPrivilegedContainer: true
allowedCapabilities:
- '*'
allowedUnsafeSysctls:
- '*'
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
- system:nodes
- system:masters
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:training-infra-etcd-backup:etcd-backup
volumes:
- '*'
```

```
oc -n training-infra-etcd-backup apply -f https://raw.githubusercontent.com/acend/openshift4-operations-training/master/backup/etcd-backup/scc_etcd-backup.yaml
oc -n training-infra-etcd-backup apply -f https://raw.githubusercontent.com/acend/openshift4-operations-training/master/backup/etcd-backup/scc_privileged-etcd-backup.yaml
```

Edit the file `~/ocp4-ops/resources/etcd-backup/secret_etcd-backup-s3-bucket.yaml` and add your S3 bucket name for etcd backups:

```
#
AWS_S3_BUCKET: "<username>-ocp-training-backup-etcd
type: Opaque
```

Now we can create the secret:

```
oc -n training-infra-etcd-backup apply -f ~/ocp4-ops/resources/etcd-backup/secret_etcd-backup-s3-bucket.yaml
```

Finally we can create the `ConfigMap` containing the backup script and the `CronJob` resources. Here's what they look like:

- acend gmbh

Create them with:

Note

In our example of the `CronJob` the backup job is started every six hours. If you do not want to wait that long, trigger the job manually with

```
oc create job --from=cronjob/etcd-backup -n training-infra-etcd-backup etcd-test-job
```

You can verify the backup job by looking at the result or by checking the logs of the pod:

- Job status:

- Check the logs:

The logs should not contain any errors and should show the successful upload of the backup to the S3 bucket:

2. Day 2-operations

Day 2-operations consist of typical operations tasks after a successful installation such as updating the cluster, making sure your backups work, resizing your cluster to the current capacity needs or replacing failed nodes or even masters.

Before you are going to do just that, we make sure that infrastructure components have their own, dedicated nodes where application workload cannot interfere.

2.1 Infrastructure nodes

Not exactly a day 2-operations task, but one that immensily simplifies operations. The concept of infrastructure nodes was more prominently advertised by Red Hat when OpenShift 3 was the latest and greatest. This evidently changed somehow for OpenShift 4, but with a bit of legwork we're able to create those, too.

Infra nodes are dedicated worker nodes intended for infrastructure components such as the image registry, routers or the whole monitoring stack. Providing dedicated nodes helps ensuring that there's always enough resources for infrastructure components.

Task 2.1.1: Create a machine set

The best way to add infra nodes to the cluster is via machine sets. Machine sets are usually only supported on clusters installed with the IPI installation method. They could be compared to deployments: If deployments are responsible for creating replica sets and ultimately pods, machine sets create machines to then add them as nodes to the cluster.

Probably the easiest way to create a machine set for infra nodes is to copy and adapt the existing `worker` MachineSet in the `openshift-machine-api` namespace.

First of all, get the yaml definition of the existing worker MachineSet resource inside the `openshift-machine-api` namespace.

Check the name of the existing worker MachineSet resource inside the `openshift-machine-api` namespace:

```
oc get machinesets -n openshift-machine-api
```

Using this information, write the existing worker MachineSet resource into a file:

```
oc get machinesets -n openshift-machine-api -o yaml machineset name> > machineset_infra.yaml
```

Now, adapt the file so its content describes an `infra` MachineSet.

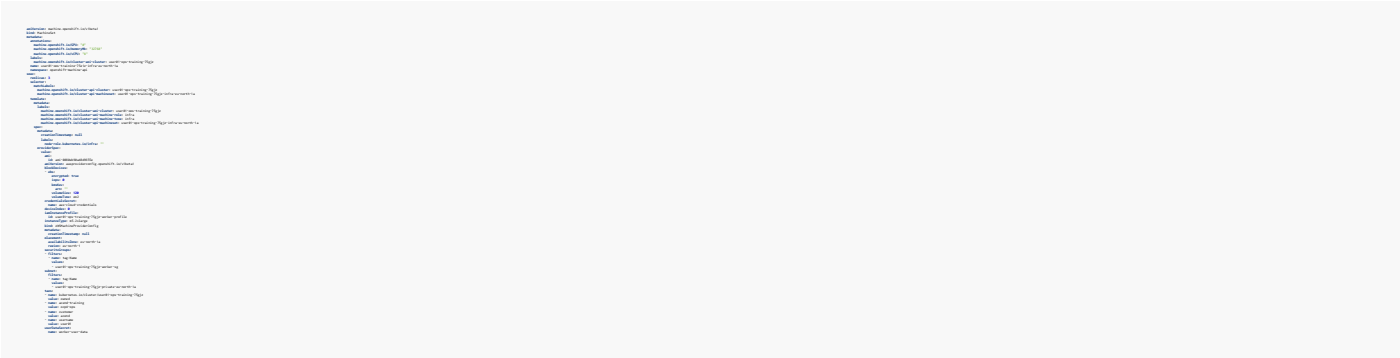
> **Warning**
> By far not all occurrences of `worker` must be changed into `infra` .
>
> Refer to the documentation if you are not sure what to replace.

Edit the file using your favorite editor (still `vim` of course):

- Remove all the unnecessary clutter such as the `.metadata.managedFields` part or `creationTimestamp` fields
- Replace all occurrences of `worker` with `infra` **except for everything under** `providerSpec`
- Ensure that `replicas` is set to 3
- Add the infra role label to the `.spec.template.spec.metadata.labels` field like so:

```
spec:
  metadata:
    creationTimestamp: null
    labels:
      node-role.kubernetes.io/infra: ""
  providerSpec:
```

When you're finished, your MachineSet resource file should look similar to this:



Warning

You cannot use the sample MachineSet resource definition above without further modification. It's recommended you use your own worker MachineSet definition and make the described changes.

Now, create the MachineSet.

```
oc create -f machineset_infra.yaml
```

Check if the machines were created and that they're in the `Provisioning` state:

```
oc get machines -n openshift-machine-api
```

It shouldn't take too long and you will see new infra nodes popping up.

2.2 Infrastructure components

Now that we have fresh infra nodes up and running, let's see to it that they have something to do. The usual suspects to run on infrastructure nodes are:

- routers
- registry
- most of the monitoring components
- most of the logging components

Red Hat provides a detailed list of what components don't incur OpenShift Container Platform worker subscriptions . So if you have any other components listed there that are not going to be looked at in this training, it could make perfect sense to move these components as well.

"Moving" components from one node to another is usually done with either using node selectors and labels or using taints and tolerations. Using node selectors and labels exists since OpenShift 3 and works the same way as the use of selector in services to define which pods they should cover. A possibility would therefore be to simply configure all infrastructure components with a node selector that points to the infra nodes' role label (`node-role.kubernetes.io/infra=''`).

There might however be use cases where node selectors reach their limit of flexibility. Imagine you had different types of worker nodes and labelled them accordingly and additionally to the worker role label (`node-role.kubernetes.io/infra=''`). What if you wanted to prevent all but a specific subset of pods to be deployed to a certain node? We can limit said pods to this one node, but we can't prevent other pods to be deployed on it as well.

This is where taints and tolerations come into play. When node selectors and labels represent an allowlist of nodes, taints and tolerations can be used to disallow pods from being deployed to specific nodes. This is the exact situation we are in right now. Our new infrastructure nodes have the infra and the worker label, but we want to make sure that only infra components can be deployed on them. Which means we are going to use a combination of the two variants:

- Use node selectors to move the infra components onto the infra nodes
- Use taints to prevent other pods from being deployed onto the infra nodes

Task 2.2.1: Taints

The first step we need to take is taint the new infra nodes. Nodes that have taints on them effectively tell the scheduler to only deploy pods onto them if they comply with the corresponding toleration.

A taint consists of a key, value, and effect and is expressed as `key=value:effect` ; the value is optional. As an effect we can choose from `NoSchedule` , `PreferNoSchedule` or `NoExecute` .

First of all, taint your infra nodes with key `node-role.kubernetes.io/infra` and effect `NoSchedule` .

This action's effect is that new pods are not scheduled onto infra nodes if they do not match the taint. Just what we wanted.

> **Note**
>
> Instead of manually tainting the infra nodes, the taints could be added to the infra `MachineSet` object. The OpenShift documentation explains how to do that. This would be the more elegant solution because it automatically taints infra nodes when new ones are added or existing ones replaced.

- acend gmbh

Before we continue with moving all the different infra components onto the infra nodes, if you want to know more about taints and tolerations, have a look at OpenShift's documentation .

Task 2.2.2: Router

The OpenShift documentation explains how to move the router and other infra pods. However, it uses node selectors and labels to do so.

In order to find out how to define a toleration for the Ingress Controller, a possible way is to have a look at its CustomResourceDefinition:

```
oc get crd ingresscontrollers.operator.openshift.io -o yaml
```

In there we find:

```
tolerations:
  description: "tolerations is a list of tolerations applied to ingress controller deployments. \n The default is an empty list. \n See https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/"
  type: array
  items:
    description: The pod this Toleration is attached to tolerates any taint that matches the triple key,value,effect using the matching operator operator.
    properties:
      effect:
        description: Effect indicates the taint effect to match. Empty means match all taint effects. When specified, allowed values are NoSchedule, PreferNoSchedule and NoExecute.
        type: string
      key:
        description: Key is the taint key that the toleration applies to. Empty means match all taint keys. If the key is empty, operator must be Exists; this combination means to match all values and all keys.
        type: string
      operator:
        description: Operator represents a key's relationship to the value. Valid operators are Exists and Equal. Defaults to Equal. Exists is equivalent to wildcard for value, so that a pod can tolerate all taints of a particular category.
        type: string
      tolerationSeconds:
        description: TolerationSeconds represents the period of time the toleration (which must be of effect NoExecute, otherwise this field is ignored) tolerates the taint. By default, it is not set, which means tolerate the taint forever (do not evict). Zero and negative values will be treated as 0 (evict immediately) by the system.
        type: integer
```

This means the toleration definition will have to look like this:

```
spec:
  nodePlacement:
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
```

Additionally, we need to define a node selector. Again looking at the CRD's definition, the node selector by itself looks like this:

```
spec:
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/infra: ""
```

So what we effectively want is the combination of the two.

```
spec:
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
```

Add the node selector and toleration to the `default` `ingresscontroller` resource.

You could either directly edit the resource:

```
oc edit ingresscontroller/default -n openshift-ingress-operator
```

Or use the following patch command:

```
oc patch ingresscontroller/default -n openshift-ingress-operator --type=merge -p '{"spec":{"nodePlacement":{"nodeSelector":{"matchLabels":{"node-role.kubernetes.io/infra":""}},"tolerations":[{"effect":"NoSchedule","key":"node-role.kubernetes.io/infra"}]}}}'
```

- acend gmbh

You can now observe the Ingress Controller Operator do its work and immediately start redeploying the router pods onto the infra nodes:

> **Note**
> You'll have to manually terminate the following command with ctrl+c.

```
oc get pods -o openshift-ingress -o wide -w
```

Task 2.2.3: Registry

Repeating the same procedure as we did with the router but with the `configs.imageregistry.operator.openshift.io` CRD, we end up with a similar-looking definition:

```
spec:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
```

Add it to the `configs.imageregistry.operator.openshift.io` custom resource `cluster` .

Again, either edit the resource directly or use the following command:

```
oc patch configs.imageregistry.operator.openshift.io/cluster --type=merge -p '{"spec":{"nodeSelector":{"node-role.kubernetes.io/infra":""},"tolerations":[{"effect":"NoSchedule","key":"node-role.kubernetes.io/infra"}]}}'
```

Watch the pods how they're being redeployed one by one:

```
oc get pods -o openshift-image-registry -o wide -w
```

Task 2.2.4: Monitoring

Moving the monitoring stack involves moving a number of different pods to the infra nodes. Placement of these pods is defined in a ConfigMap and is documented here .

Gather all the relevant information and create an appropriate ConfigMap.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    kubeStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    grafana:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    telemeterClient:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    openshiftStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    thanosQuerier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
```

Apply it to the cluster.

```
oc create -f https://raw.githubusercontent.com/openshift/openshift-ansible/master/roles/openshift_cluster_monitoring_operator/examples/config.yaml
```

As soon as the ConfigMap is created, the monitoring operator begins redeploying the pods:

```
oc get pods -n openshift-monitoring -w
```

This concludes moving all the current infrastructure components to the infra nodes.

2.3 Restore

Task 2.3.1: Restoring user workload backups

In the backup lab you created a scheduled backup of all resources in namespace `uptime-app-prod`. Now we are going to test the restore procedure.

> **Warning**
> To keep your restore time as short as possible, make sure you read through the whole procedure, understand it and maybe even prepare all resources for the restore before deleting the project!

Make sure you read above warning, then go ahead and delete the project `uptime-app-prod`.

```
oc delete project uptime-app-prod
```

Update your backup storage location to read-only mode (this prevents backup objects from being created or deleted in the backup storage location during the restore process):

```
oc patch backupstoragelocation default \
    --namespace openshift-adp \
    --type merge \
    --patch '{"spec":{"accessMode":"ReadOnly"}}'
```

In order to restore a backup, we need to create a `Restore` object. A `Restore` object looks like this:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-uptime-app-prod
  namespace: openshift-adp
spec:
  backupName: <backup-name>
  includedNamespaces:
    - uptime
  excludedResources:
    - nodes
    - events.events.k8s.io
    - events
    - route.apps.ocp.io
    - controllerrevisions.apps
  restorePVs: true
```

Copy above `Restore` resource into a file on your bastion host. Replace the placeholder `<backup-name>` with the name of the backup you want to restore. To list your available backups:

```
oc -n openshift-adp get backups
```

Start the restore by creating the `Restore` object:

```
oc apply -f restore-file
```

After the restore has completed, your uptime app should be up and running again:

```
oc -n uptime-app-prod get pods
```

Example output:

```
NAME                          READY   STATUS    RESTARTS   AGE
uptime-app-56df4df7d8-hnsps   1/1     Running   0          20m
uptime-app-56df4df7d8-jfkzx   1/1     Running   0          20m
```

When ready, revert your backup storage location to read-write mode:

```
oc patch backupstoragelocation default \
   --namespace openshift-adp \
   --type merge \
   --patch '{"spec":{"accessMode":"ReadWrite"}}'
```

2.4 Update

In this lab we will update our OpenShift 4 cluster to the latest stable errata release.

Updating an OpenShift 4 cluster is a fully automated process that is managed by several cluster operators. If the cluster has Internet access, updates are provided over-the-air, otherwise you need to synchronize the new images to an internal registry prior to starting the update or use a pull-through registry.

Check “Updating a restricted network cluster” for details on updating a cluster in a disconnected environment.

The update process continuously rolls out new releases of all cluster operators and verifies their readiness before continuing. If a cluster operator is stuck in a non-ready state, the update does not proceed and manual intervention might become necessary.

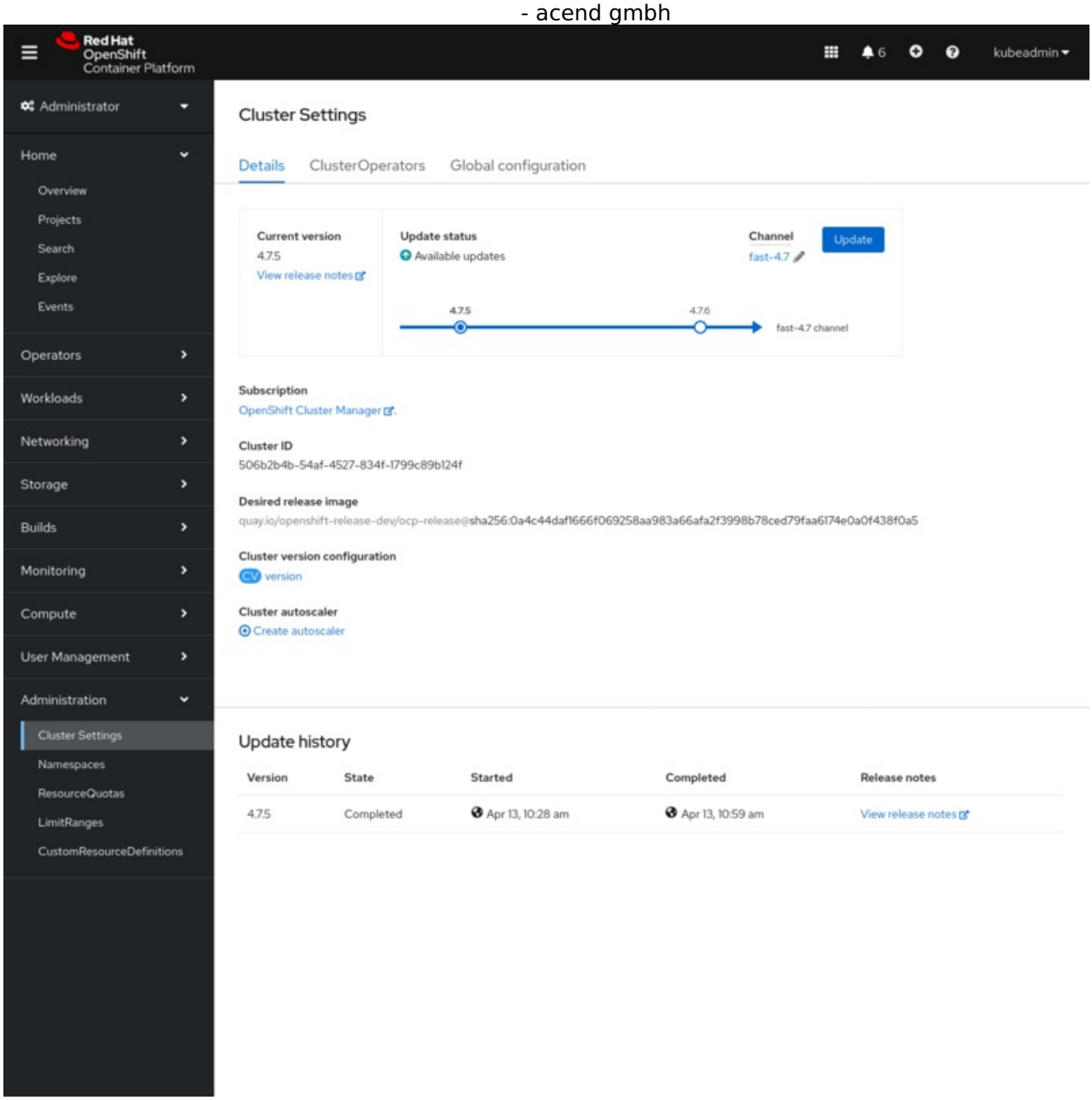
After all relevant cluster operators are updated successfully, the machine config operator updates the configuration of the master and worker nodes and - if a newer version is available - replaces the nodes’ operating system image (Red Hat Enterprise Linux CoreOS).

Task 2.4.1: Updating the cluster

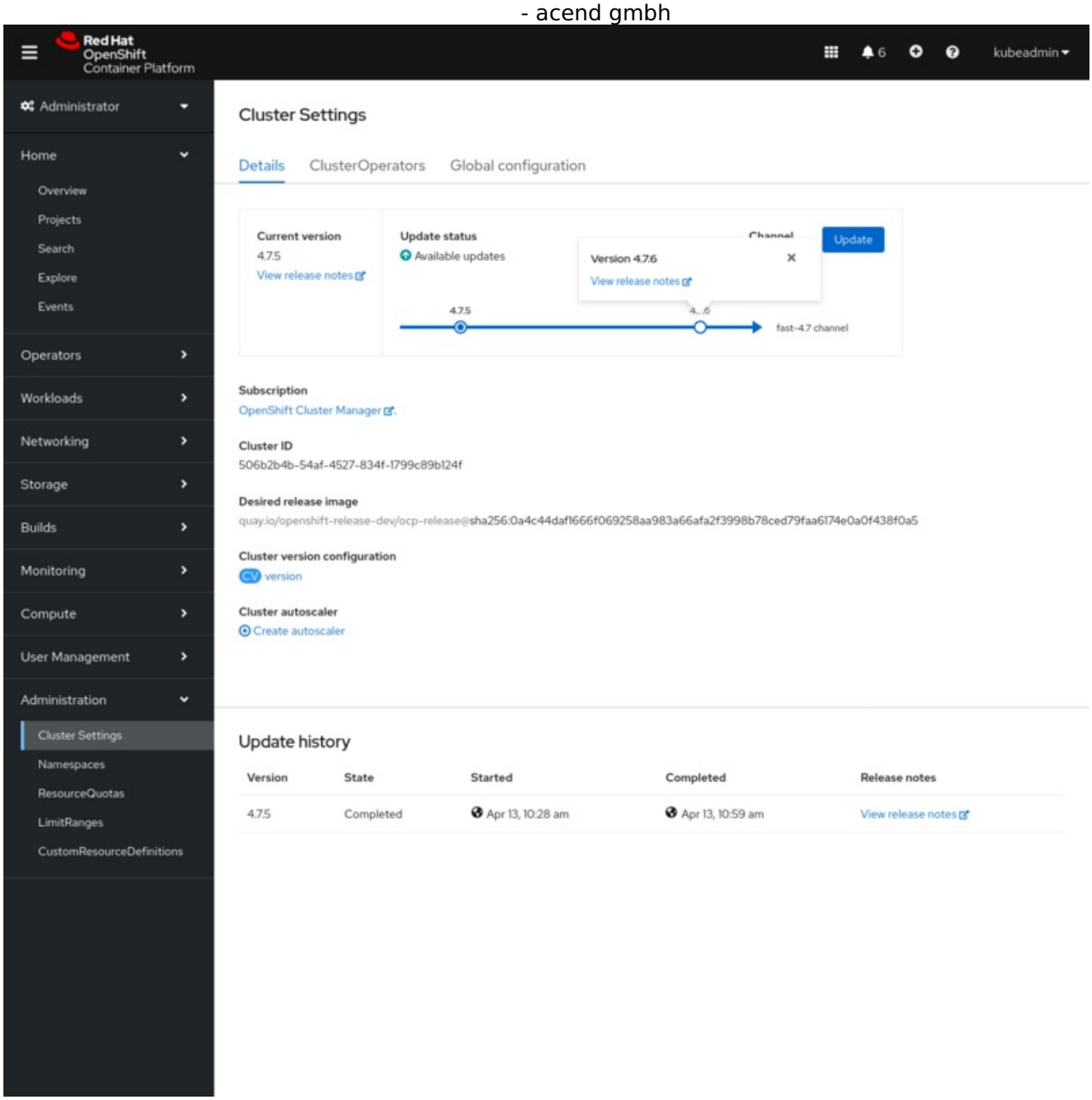
The cluster update can be performed from the web console or the CLI. Choose the one you prefer to do or will most likely perform in the future.

Updating from the web console

Navigate to **Administration**, then **Cluster Settings** to get started. You will see the available update paths in the graph:

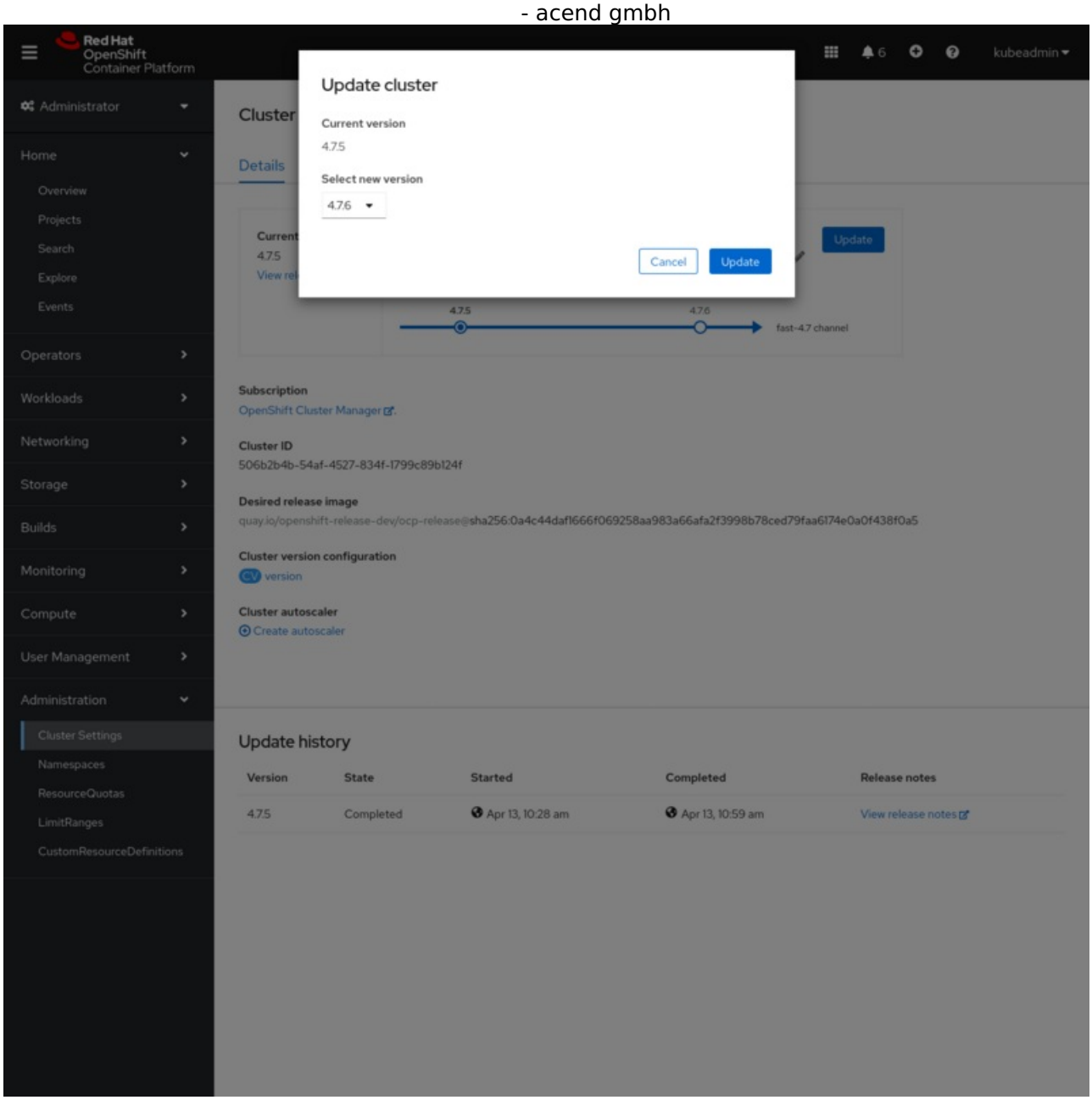


To read the release notes of the new version, click on the target version and follow the link:

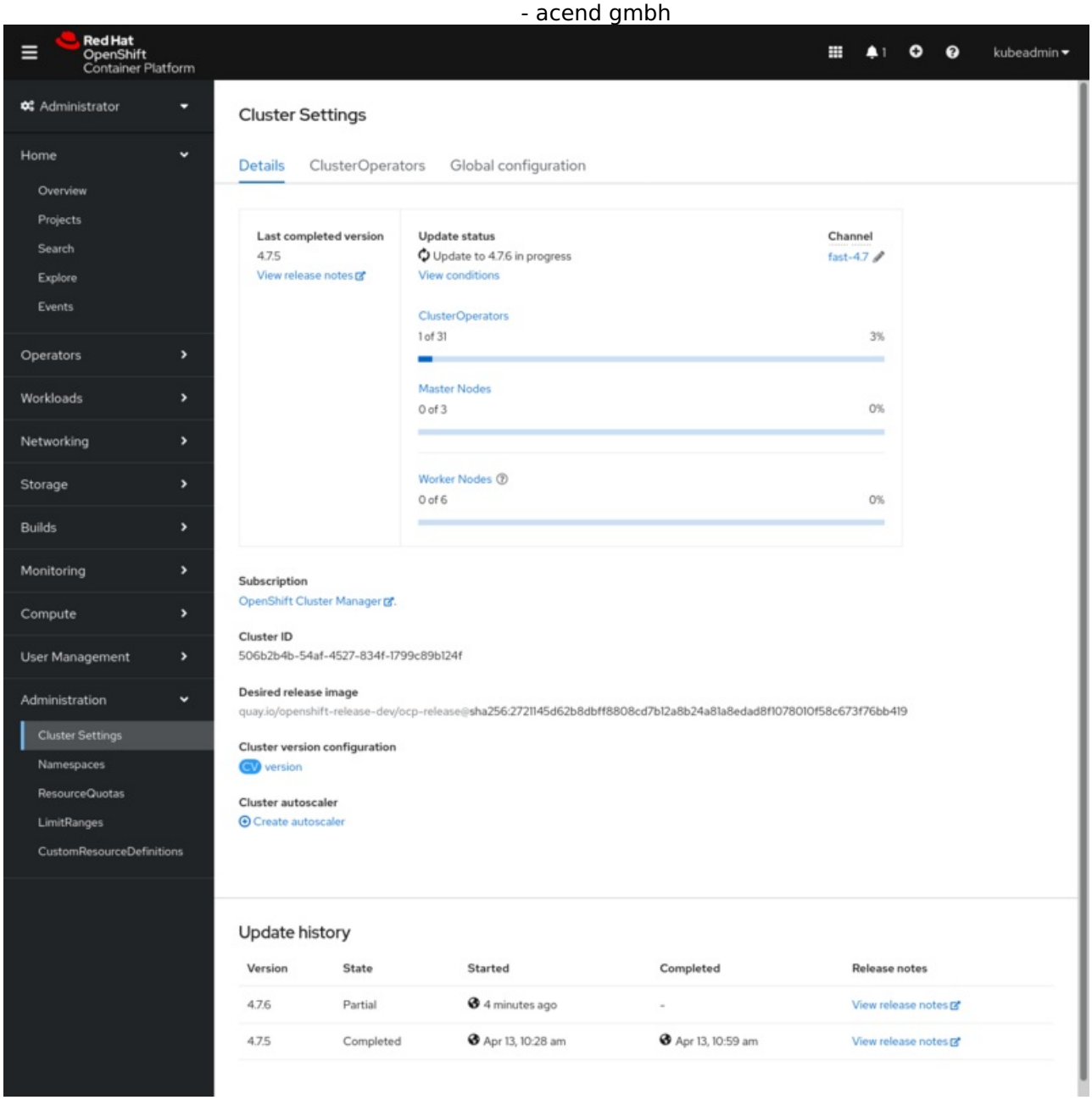


The update process from the web console is pretty straightforward. To start the update, all you have to do is press two buttons.

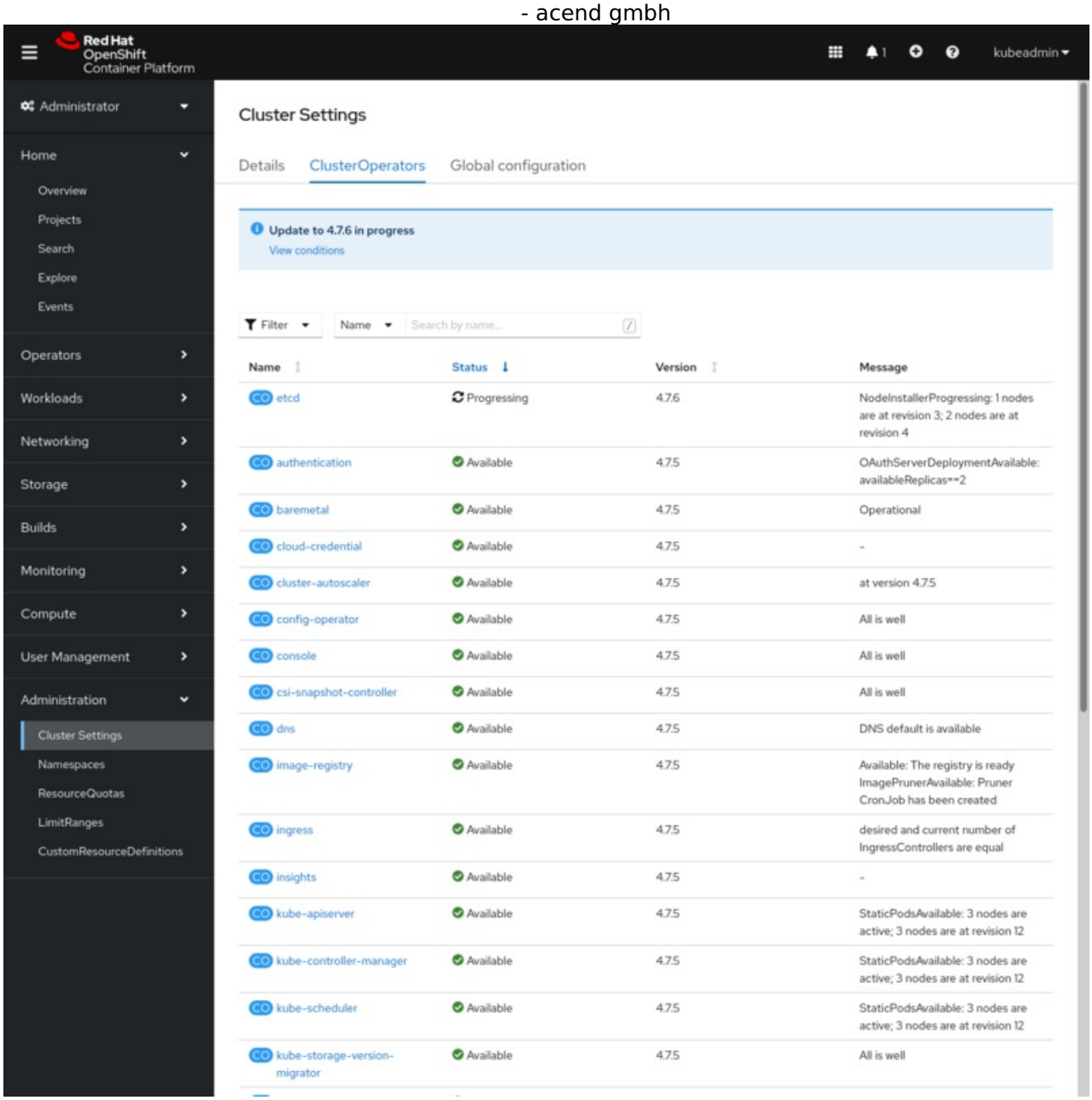
Click the update button and confirm the update to the target version by again clicking on the update button:



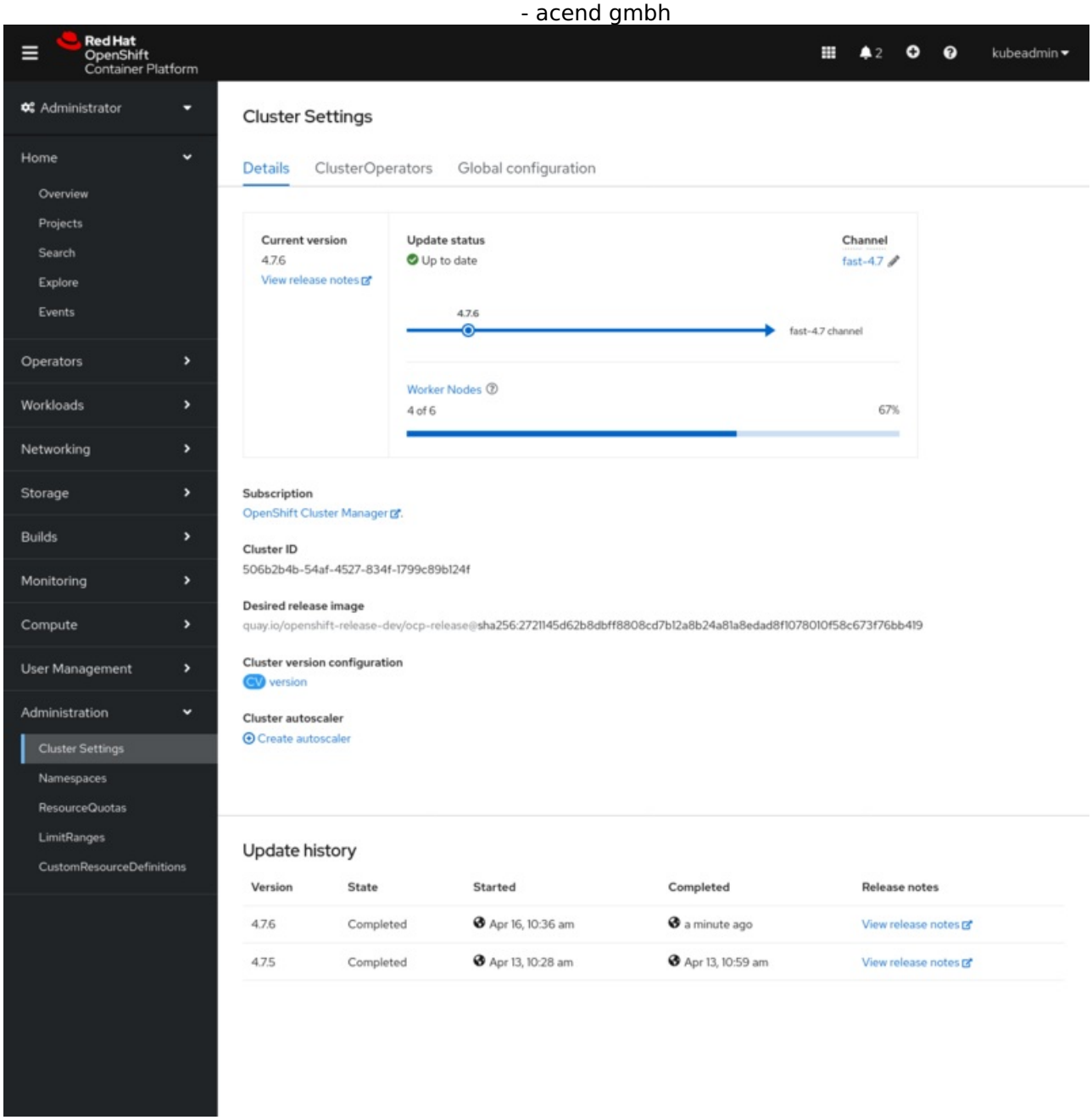
The update process will start updating the cluster operators:



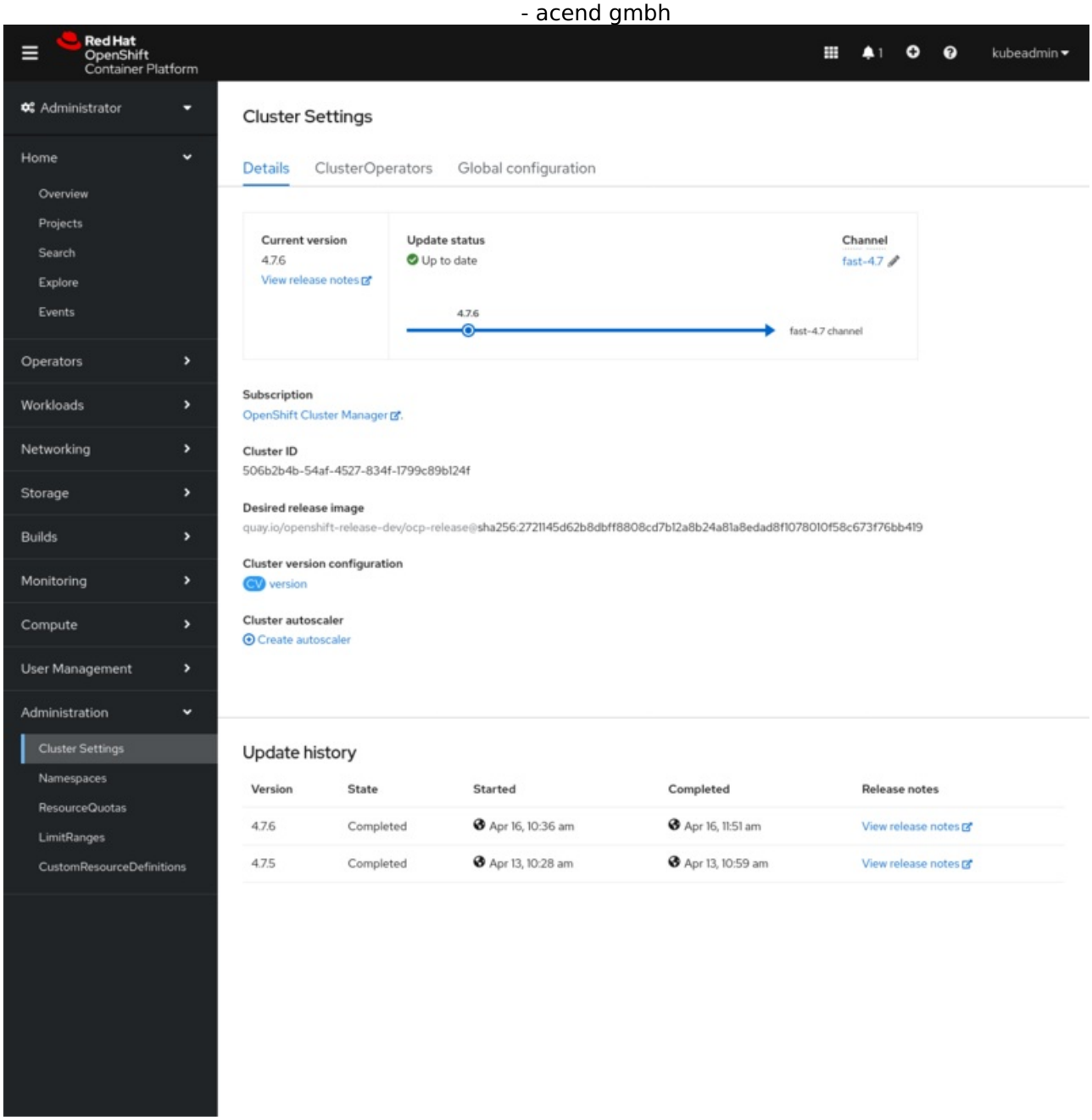
You can see more details in the second tab:



Worker nodes may continue to update after the update of the masters and cluster operators is complete:



Finally, the update is completed:



Updating the cluster from the CLI

Before starting the update, ensure that your cluster is available by looking at the `clusterversion` resource.

```
```

Example output:

```
```

You can check the available updates by running the `oc adm upgrade` command without any parameters:

```
oc adm upgrade
```

Example output:

```
Cluster version is 4.7.5

Updates:

VERSION  IMAGE
4.7.8    quay.io/openshift-r...
```

Warning

You can only update to one of the available target versions. Do not force the update to an unsupported version, since it could render your cluster useless.

Starting the update process depends on what we want to do. We have two options:

- To update to the latest version:

```
oc adm upgrade --to-latest=true
```

- To update to a specific version:

```
oc adm upgrade --to=<version>
```

You can track the progress of the update by checking the history of the output of the `clusterversion` :

```
oc get clusterversion -o jsonpath='{.items[0].status.history}'
```

Example output:

```
[
  {
    "completionTime": null,
    "image": "quay.io/openshift-r...",
    "startedTime": "2021-04-15T08:39:49Z",
    "state": "Partial",
    "verified": false,
    "version": "4.7.8"
  },
  {
    "completionTime": "2021-04-15T08:39:49Z",
    "image": "quay.io/openshift-r...",
    "startedTime": "2021-04-15T08:20:41Z",
    "state": "Completed",
    "verified": false,
    "version": "4.7.5"
  }
]
```

You can also check the progress of the update by tracking the status of the ClusterOperators.

```
oc get clusteroperators
```

Example output:

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
authentication                             4.7.5     True        False         False      148m
baremetal                                  4.7.5     True        False         False      3d
cloud-credential                           4.7.5     True        False         False      3d
cluster-autoscaler                         4.7.5     True        False         False      3d
config-operator                            4.7.6     True        False         False      3d
console                                    4.7.5     True        False         False      2d21h
csi-snapshot-controller                    4.7.5     True        False         False      3d
dns                                        4.7.5     True        False         False      3d
etcd                                       4.7.6     True        False         False      3d
image-registry                             4.7.5     True        False         False      3d
ingress                                    4.7.5     True        False         False      3d
insights                                   4.7.5     True        False         False      3d
kube-apiserver                             4.7.6     True        False         False      3d
kube-controller-manager                    4.7.6     True        False         False      3d
kube-scheduler                             4.7.6     True        False         False      3d
kube-storage-version-migrator              4.7.5     True        False         False      3d
machine-api                                4.7.6     True        False         False      3d
machine-approver                           4.7.5     True        False         False      3d
machine-config                             4.7.5     True        False         False      3d
marketplace                                4.7.5     True        False         False      3d
monitoring                                 4.7.5     True        False         False      74m
network                                    4.7.5     True        False         False      3d
node-tuning                                4.7.5     True        False         False      3d
openshift-apiserver                        4.7.6     True        False         True       3d
openshift-controller-manager               4.7.5     True        False         False      3d
openshift-samples                          4.7.5     True        False         False      3d
operator-lifecycle-manager                 4.7.5     True        False         False      3d
operator-lifecycle-manager-catalog         4.7.5     True        False         False      3d
operator-lifecycle-manager-packageserver   4.7.5     True        False         False      3d
service-ca                                 4.7.5     True        False         False      3d
storage                                    4.7.5     True        False         False      3d
```

Note

In the example output above you can see that the `openshift-apiserver` ClusterOperator is already reporting the new version, but is currently in a degraded state. This is expected behaviour during the update process, so no need to worry.

After all cluster operators are rolled out, the master and worker nodes are updated. The machine config operator sequentially cordons the nodes, applies the new configuration and restarts the Kubelet service.

Check the status of the nodes to see the process.

```
oc get nodes
```

Example output:

```
NAME                                          STATUS                     ROLES          AGE    VERSION
ip-10-0-130-69.eu-north-1.compute.internal    Ready,SchedulingDisabled   infra,worker   43h    v1.20.0+bafe72f
ip-10-0-139-81.eu-north-1.compute.internal    Ready                      worker         3d     v1.20.0+bafe72f
ip-10-0-154-185.eu-north-1.compute.internal   Ready                      infra,worker   43h    v1.20.0+bafe72f
ip-10-0-154-3.eu-north-1.compute.internal     Ready                      infra,worker   43h    v1.20.0+bafe72f
ip-10-0-155-28.eu-north-1.compute.internal    Ready,SchedulingDisabled   master         3d1h   v1.20.0+bafe72f
ip-10-0-173-170.eu-north-1.compute.internal   Ready                      worker         3d     v1.20.0+bafe72f
ip-10-0-184-171.eu-north-1.compute.internal   Ready                      master         3d1h   v1.20.0+bafe72f
ip-10-0-195-93.eu-north-1.compute.internal    Ready                      worker         3d     v1.20.0+bafe72f
ip-10-0-212-27.eu-north-1.compute.internal    Ready                      master         3d1h   v1.20.0+bafe72f
```

Note

By default, only one machine per pool is allowed to be unavailable during the update process. For a larger cluster, this can take a long time for the configuration change to be reflected. We can alter this behaviour by changing the `maxUnavailable` value in the respective nodes' `MachineConfigPool`.

2.5 Worker nodes

We initially installed this cluster using only worker nodes. We then successfully created infra nodes and moved all infrastructure components onto them. Now, essentially the only real workload that's running on the 3 worker nodes is our uptime application.

Without going into detail yet on how and where you can find dashboards and metrics for capacity monitoring, it's safe to say that one small Python app doesn't need 3 worker nodes with 8 CPU cores and 32 GB of memory. So we are going to decrease the number of worker nodes to 2 and also reduce their size.

We have to admit that in real life, this would probably be the other way round. Instead of reducing the number of nodes and their capacity, we would add more nodes and maybe even make them bigger. However, be it reducing or enlarging the cluster, the process is the same and the forecast for this training tells us that we are not going to see much more workload.

Task 2.5.1: Change the number of nodes

Changing the number of nodes can be either done via CLI or in the web console.

Change the number of nodes in the web console

- Make sure you're in the Administrator view (see the dropdown menu top left)
- In the menu on the left, choose Compute and then MachineSets
- Click the 3-dot button on the right next to the worker machineset:

- Choose to edit Machine count
- Reduce the number of machines to 2 and click Save

If you now switch to the Nodes or Machines overview in the menu on the left, you can see that OpenShift already started the deprovisioning process.

Change the number of nodes on the command line

Scaling OpenShift nodes via CLI works the exact same way as scaling pods.

We first want to check which MachineSet resource we want to scale:

```
```

Now pick the worker machineset and use it in the following command to scale it:

```
```

Yes, it's that easy.

Task 2.5.2: Change the node size

The way nodes are resized depends on the installation method. If the cluster was installed using UPI, the underlying infrastructure solution is responsible for sizing the machines. Thus, if you want to resize a node in a UPI installation, you need to do it yourself, OpenShift doesn't directly manage these machines.

In an IPI installation, the infrastructure is managed by OpenShift, too. This is done using machine sets which we just used to resize the number of worker nodes.

In order to find out what exactly we have to change, let's have a closer look at the MachineSet resource. Here's an excerpt of the more relevant fields:

We can see that the annotations contain the information of how many resources machines from this set have. But changing these values won't change anything. The field we are interested in is the `instanceType` field. It contains the AWS instance flavor .

Change this field to `m5.large` (2 vCPUs, 8 GiB Memory).

After the change, nothing happens. OpenShift only applies this change to new nodes. We'll have to delete every machine with the old instance type one by one so they get replaced by new ones.

Delete one of the worker machines.

Now periodically check for the nodes' status until the new node is up again.

Repeat the same step for the other worker machine.

> **Note**
> It might make perfect sense to first scale up the nodes before you begin the renewal process. This helps to ensure high availability on the cluster.

2.6 Control plane nodes

> **Note**
> While many occurrences of “master” have been replaced throughout OpenShift and Kubernetes, this is unfortunately not yet the case for `Machine` resources. This is why you are going to encounter a mixture of “control plane” and “master” terms in this lab.

During this lab, we will learn how to replace a failed control plane node. As long as we do not lose the majority of our control plane nodes, we can simply replace those that failed. If we lose the majority (e.g., 2 out of 3), we need to restore the state of `etcd` from a previously created snapshot, which will not be covered in this lab.

Since the control plane nodes are hosting `etcd`, the key-value store that holds all the cluster’s information and state, it is not as straightforward as replacing a worker node backed by a machine set.

> **Note**
> Above statement is not completely correct anymore. OpenShift 4.12 introduced the control plane machine sets for specific infrastructure providers . Because the control plane machine set on AWS is supported and active by default, we are going to delete it in the first step of this lab.
> This enables you to do this lab and experience the replacement of a control plane node anyway, in case you need to do it on infrastructure that is not supported by or has no active control plane machine set.

Task 2.6.1: Replacing the machine

As mentioned in above note, the first task is to delete the control plane machine set. The control plane machine set will immediately be recreated, however, it will not be active anymore.

```
oc -n openshift-machine-api delete controlplanemachineset.machine.openshift.io cluster
```

Now, save the resource definition of the master machine you want to replace.

```
oc -n openshift-machine-api get machine <failed-master-machine> -o yaml > master_<failed-master-machine>.yaml
```

Edit the file to reflect the following changes:

- Remove:
  - `metadata.annotations`
  - `metadata.creationTimestamp`
  - `metadata.generations`
  - `metadata.uid`
  - `metadata.resourceVersion`
  - `spec.providerID`
  - all of `status`

The `machine` resource should look similar to this:

```yaml
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  finalizers:
  - machine.machine.openshift.io
  labels:
    machine.openshift.io/cluster-api-cluster: user01-ops-training-75gjz
    machine.openshift.io/cluster-api-machine-role: master
    machine.openshift.io/cluster-api-machine-type: master
    machine.openshift.io/instance-type: m5.xlarge
    machine.openshift.io/region: eu-north-1
    machine.openshift.io/zone: eu-north-1c
  name: user01-ops-training-75gjz-master-2
  namespace: openshift-machine-api
spec:
  metadata: {}
  providerSpec:
    value:
      ami:
        id: ami-0080eb90a48d9655e
      apiVersion: awsproviderconfig.openshift.io/v1beta1
      blockDevices:
      - ebs:
          encrypted: true
          iops: 0
          kmsKey:
            arn: ""
          volumeSize: 120
          volumeType: gp2
      credentialsSecret:
        name: aws-cloud-credentials
      deviceIndex: 0
      iamInstanceProfile:
        id: user01-ops-training-75gjz-master-profile
      instanceType: m5.xlarge
      kind: AWSMachineProviderConfig
      loadBalancers:
      - name: user01-ops-training-75gjz-int
        type: network
      - name: user01-ops-training-75gjz-ext
        type: network
      metadata:
        creationTimestamp: null
      placement:
        availabilityZone: eu-north-1c
        region: eu-north-1
      securityGroups:
      - filters:
        - name: tag:Name
          values:
          - user01-ops-training-75gjz-master-sg
      subnet:
        filters:
        - name: tag:Name
          values:
          - user01-ops-training-75gjz-private-eu-north-1c
      tags:
      - name: kubernetes.io/cluster/user01-ops-training-75gjz
        value: owned
      - name: customer
        value: acend
      - name: username
        value: user01
      - name: acend-training
        value: ocp4-ops
      userDataSecret:
        name: master-user-data
```

Delete the master machine you want to replace.

Warning

Make sure you delete the master machine you just saved the resource definition from in the previous step.

```
oc -n openshift-machine-api delete machine <failed-master-machine>
```

This will queue the machine for deletion. You can verify this by checking the machines' state.

```
oc -n openshift-machine-api get machines -o wide
```

Example output:

```
NAME                                               PHASE      TYPE         REGION       ZONE          AGE    NODE                                          PROVIDERID                              STATE
user01-ops-training-75gjz-infra-eu-north-1a-9mzd9   Running    m5.2xlarge   eu-north-1   eu-north-1a   46h    ip-10-0-130-69.eu-north-1.compute.internal    aws:///eu-north-1a/i-0a7b134c9b0d07d4d   running
user01-ops-training-75gjz-infra-eu-north-1a-h88ln   Running    m5.2xlarge   eu-north-1   eu-north-1a   46h    ip-10-0-154-3.eu-north-1.compute.internal     aws:///eu-north-1a/i-0b98e80b4cd46d367   running
user01-ops-training-75gjz-infra-eu-north-1a-smgzs   Running    m5.2xlarge   eu-north-1   eu-north-1a   46h    ip-10-0-154-185.eu-north-1.compute.internal   aws:///eu-north-1a/i-0758413e1557c439f   running
user01-ops-training-75gjz-master-0                  Running    m5.xlarge    eu-north-1   eu-north-1a   3d3h   ip-10-0-155-28.eu-north-1.compute.internal    aws:///eu-north-1a/i-0736635506a9eb2cc   running
user01-ops-training-75gjz-master-1                  Running    m5.xlarge    eu-north-1   eu-north-1b   3d3h   ip-10-0-184-171.eu-north-1.compute.internal   aws:///eu-north-1b/i-067f7bac5768ab117   running
user01-ops-training-75gjz-master-2                  Deleting   m5.xlarge    eu-north-1   eu-north-1c   3d3h   ip-10-0-212-27.eu-north-1.compute.internal    aws:///eu-north-1c/i-02210ddbae879539e   running
user01-ops-training-75gjz-worker-eu-north-1a-vv59v  Running    m5.2xlarge   eu-north-1   eu-north-1a   3d3h   ip-10-0-139-81.eu-north-1.compute.internal    aws:///eu-north-1a/i-0d243001e9ba4cb9b   running
user01-ops-training-75gjz-worker-eu-north-1b-zddgj  Running    m5.2xlarge   eu-north-1   eu-north-1b   3d3h   ip-10-0-173-170.eu-north-1.compute.internal   aws:///eu-north-1b/i-028cec3fbf26c0f13   running
user01-ops-training-75gjz-worker-eu-north-1c-6mqls  Running    m5.2xlarge   eu-north-1   eu-north-1c   3d3h   ip-10-0-195-93.eu-north-1.compute.internal    aws:///eu-north-1c/i-0d7f8ed2c8b9c0932   running
```

This machine will not be deleted right away. You can see in the resource definition that its lifecycle contains a pre-drain-hook, executed before draining the machine. This step must run before the draining of the machine, before the deletion itself:

```yaml
spec:
  lifecycleHooks:
    preDrain:
    - name: EtcdQuorumOperator
      owner: clusteroperator/etcd
```

The hook ensures that the collection of etcd nodes (the "quorum") remains in a healthy state. This is no longer granted if we remove one node out of the three expected ones. If we manually decrease the number of required nodes in the healthy state, then the third node can be deleted.

Task 2.6.2: Deleting the `etcd` member

For the master machine to be fully removed, we need to also remove the `etcd` member, since it still has the configuration of the old control plane node.

Connect to an `etcd` pod that is not running on the control plane node you just removed:

```
oc -n openshift-etcd rsh <etcd-pod>
```

View the member list and take note of the ID and name of the `etcd` member you want to remove.

> **Note**
> To identify the member you need to remove, compare the list with the existing pod names. In the example, the list shows a member with the name `ip-10-0-212-27.eu-north-1.compute.internal` which does not correspond with any of the control plane node names.

All relevant information can be found in [OpenShift's documentation](#) .

```
etcdctl member list -w table
```

Example output:

```
+------------------+---------+--------------------------------------------+---------------------------+---------------------------+------------+
|        ID        | STATUS  |                    NAME                    |        PEER ADDRS         |       CLIENT ADDRS        | IS LEARNER |
+------------------+---------+--------------------------------------------+---------------------------+---------------------------+------------+
|  ce9057897l689b5 | started |  ip-10-0-155-28.eu-north-1.compute.internal |  https://10.0.155.28:2380 |  https://10.0.155.28:2379 |      false |
| 2c53f4526118a2eb | started | ip-10-0-184-171.eu-north-1.compute.internal | https://10.0.184.171:2380 | https://10.0.184.171:2379 |      false |
| 43997d8a75197361 | started |  ip-10-0-212-27.eu-north-1.compute.internal |  https://10.0.212.27:2380 |  https://10.0.212.27:2379 |      false |
+------------------+---------+--------------------------------------------+---------------------------+---------------------------+------------+
```

Remove the old peer entry.

```
etcdctl member remove <id>
```

Verify the member was removed.

```
etcdctl member list -w table
```

This is it! You can now disconnect from the etcd pod.

The machine we marked for deletion can now finally proceed. We can check the machines' state again.

> **Note**
> The machine deletion might take several minutes to complete.

```
oc -n openshift-machine-api get machines -o wide
```

Task 2.6.3: Recreate the control plane node

The cluster is again in a properly running state, of course however with only two instead of three control plane nodes and etcd members.

Recreate the master machine using the file definition you created before.

```
oc -n openshift-machine-api apply -f machine_<failed-master-machine>.yaml
```

- acend gmbh

Verify the machine was created successfully:

```
oc -n openshift-machine-api get machines -o wide
```

The `etcd` operator should automatically scale up a new member and add it to the cluster.

Verify that three `etcd` pods are up and running.

```
oc -n openshift-etcd get pods -l etcd
```

Note

If only two pods are running, you can force a redeployment:

```
oc patch etcd cluster \
  -p='{"spec": {"forceRedeploymentReason": "recovery-'"$(date --rfc-3339=ns)"'"}}' \
  --type=merge
```

All that is left to do is clean up the old secrets that are not used anymore.

List the secrets of the old peer that can be safely deleted:

```
oc -n openshift-etcd get secrets | grep "<removed>peer"
```

Example output:

```
etcd-peer-ip-10-0-142-25.ec2.internal   kubernetes.io/tls   2   56m
etcd-peer-ip-10-0-154-61.ec2.internal   kubernetes.io/tls   2   56m
etcd-serving-ip-10-0-142-25.ec2.internal   kubernetes.io/tls   2   56m
```

Delete those secrets.

```
oc -n openshift-etcd delete secrets \
  etcd-peer-<removed>peer-ip-10-0-142-25.ec2.internal \
  etcd-serving-<removed>peer-ip-10-0-142-25.ec2.internal
```

3. Monitoring and logging

OpenShift provides a cluster monitoring stack which can be used to monitor default platform components or even user-defined projects components .

The OpenShit-provided cluster logging uses ElasticSearch to store cluster and application logs, which can then be visualised using Kibana.

In this lab, you are going to configure alerting for the cluster monitoring stack, enable monitoring for user-defined projects and deploy the logging stack.

3.1 Rules and dashboards

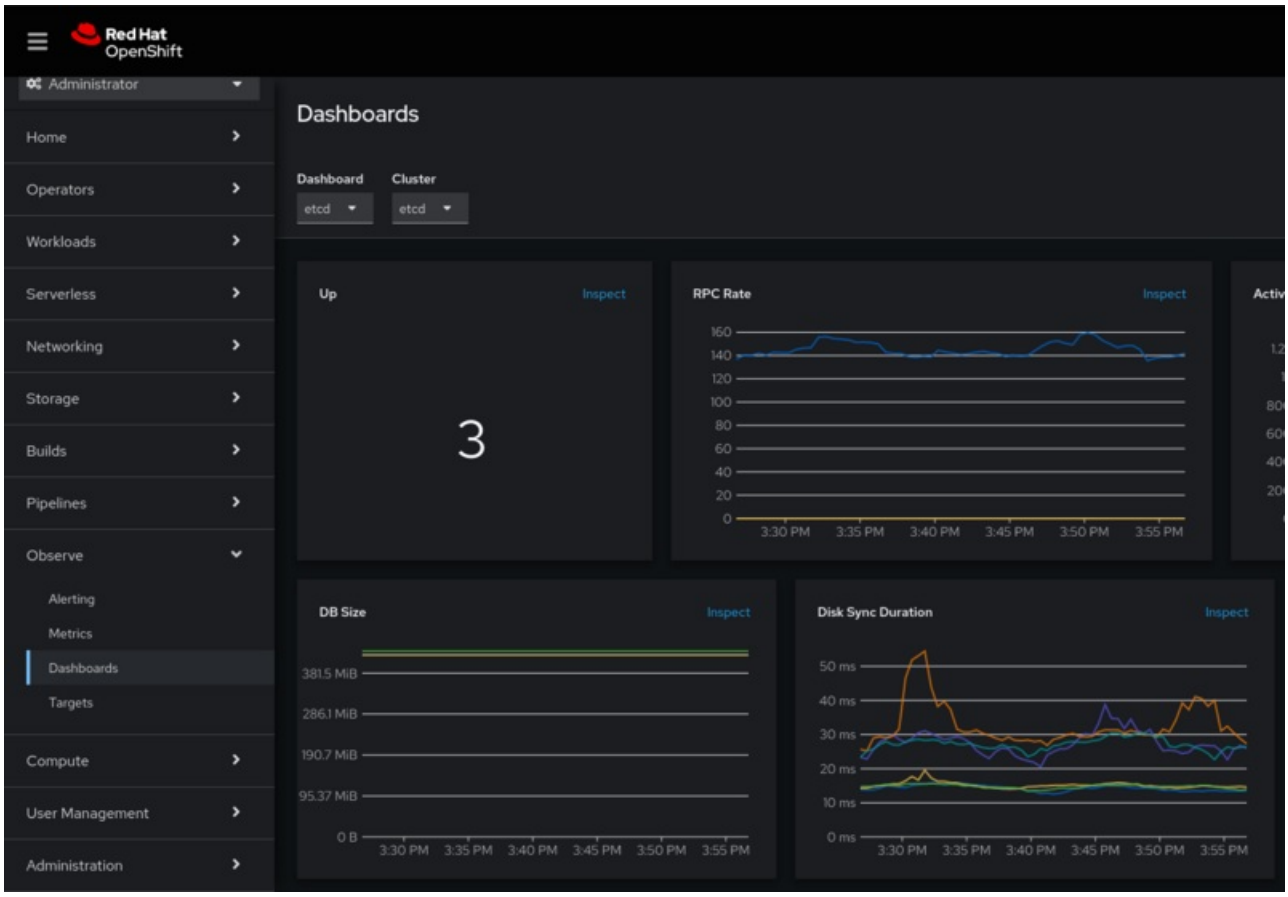
Task 3.1.1: Provided dashboards

OpenShift provides a set of default dashboards for cluster adminstrators and OpenShift users.

- **Cluster dashboards**

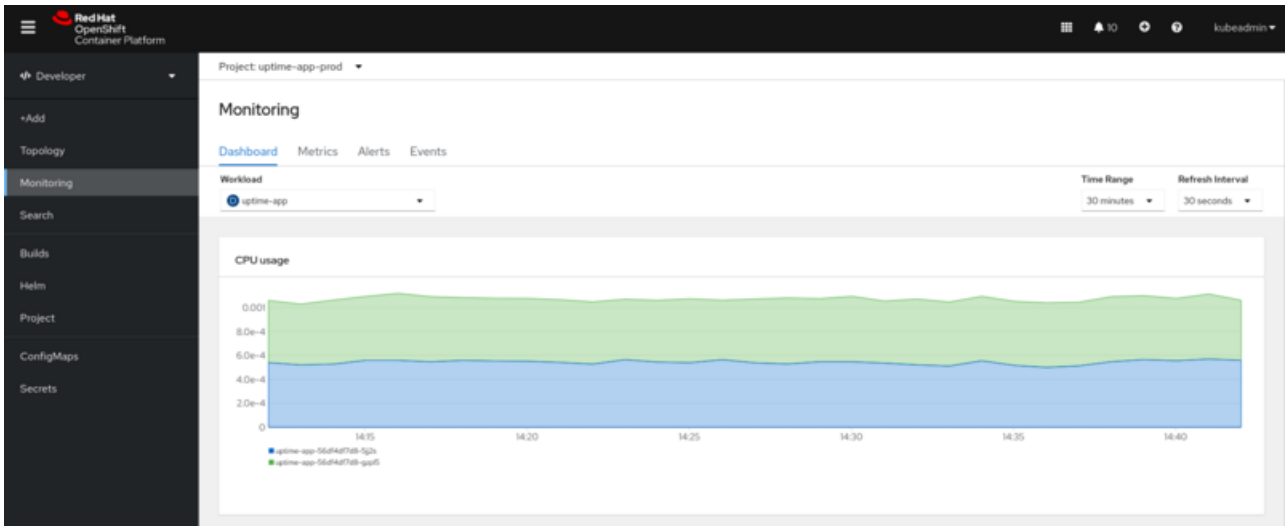
To get a sense of where these dashboards can be found, let’s take a look at the performance of the etcd cluster.

Navigate to **Observe** -> **Dashboards** on the web console and select **etcd** in the dashboard drop-down list.



- **User dashboards**

Switch to the **Developer** console and select the **uptime-app-prod** project. Then navigate to the **Observing** tab. Select the **uptime-app** in the **Workload** drop-down list. This will display a wide selection of different metrics relevant to your app like CPU usage, memory usage and networking performance.



Task 3.1.2: Provided alerting rules

OpenShift provides an extensive set of alerts, based on the kubernetes-mixin project. Check out the configured alerts by navigating to **Observe** -> **Alerting** -> **Alerting rules** and take a look at some of the predefined alerting rules.

Active alerts (state `Pending` or `Firing`) are displayed in **Observe** -> **Alerting** -> **Alerts**.

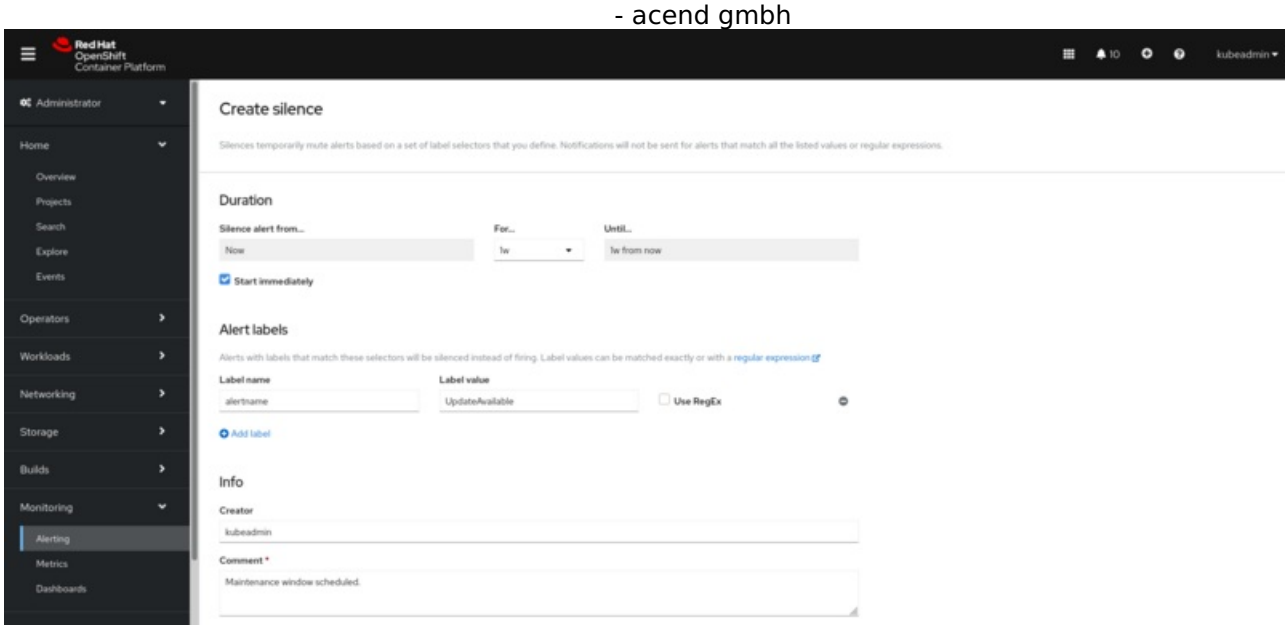
As you can see, these alerts are as generic as possible to fit most platforms. They cannot be altered and adding more rules in `openshift-monitoring` is not supported. In a later lab, you will learn how to add rules by enabling monitoring for user-defined projects.

Task 3.1.3: Create silences

Sometimes you have to silence an alert because someone is already working on it, or the issue is scheduled to be fixed in a future maintenance window, or it might be a false positive.

To do so, navigate to **Observe** -> **Alerting** -> **Silences** and hit the **Create Silence** button. A common task is to silence the `UpdateAvailable` alert, as we may have already scheduled the corresponding update.

Silence the alert for 1 week.

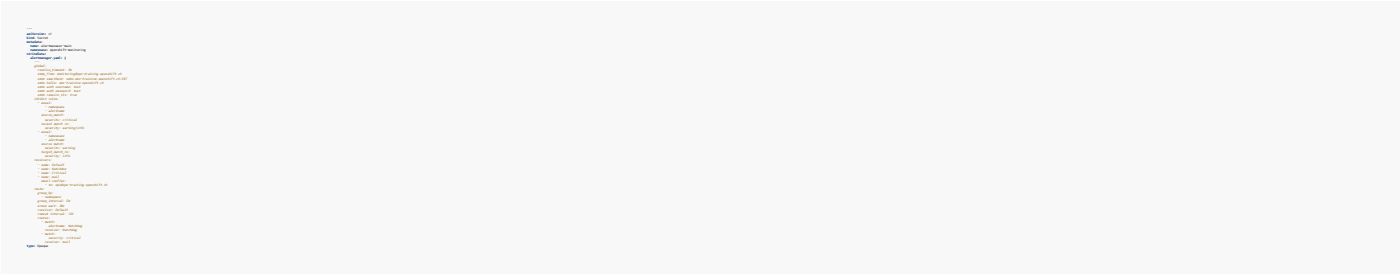


This will suppress all `UpdateAvailable` alerts for one week.

3.2 Monitoring stack configuration

Task 3.2.1: Enable alerting

Let’s take a look at an example of how you can send alerts to a receiver of your choice. Configuring a new receiver can either be done using the web console or by directly configuring the appropriate secret. Before going into detail on where this configuration lies, let’s have a look at an example:



The main components that can be configured when applying a custom Alertmanager configuration comprise:

- `receivers` : With a receiver, one or more notification types such as mails, webhooks or messaging platforms like Slack or PagerDuty can be defined.
- `routes` : With routing blocks, a tree of routes and child routes can be defined. Each routing block has a matcher which can match one or several labels of an alert. Per block, one receiver can be specified, or if empty, the default receiver is taken.
- `Watchdog` : There is a default Watchdog alert that is periodically firing to make sure that the complete alerting pipeline is functional. It is best practice to implement a Deadman’s switch on the receiving side to get notified when sending alerts should fail.
- `inhibit_rules` : You can group alerts so if one alert of this group was triggered, others will not. This is to prevent notification flooding. As an example, imagine an etcd member is unavailable (**severity: critical**). As a consequence, this would also trigger a high number of failed leader proposals (**severity: warning**). When fixing the unavailable etcd member, both alerts will be resolved, so you only care about one of them.

You are now going to configure Alertmanager so that alerts created by Prometheus are sent to your alerts channel on Slack. In order to do this, navigate to **Administration**, **Cluster Settings**, **Global Configuration**, **Alertmanager** on the web console. Here you can see the pre-configured **Receivers**. Click on **Configure** next to the **Default** receiver. Now fill in the **Slack API URL** and **Channel** information provided to you by your trainer and click **Save**.

> **Note**
>
> By clicking on **Show advanced configuration**, you have the option of customizing the messages’ appearance in Slack. Feel free to do so if you’d like.

Switch to Slack and check if notifications are showing up in your alerts channel. This might take some minutes so don’t hesitate to continue the lab and check back later.

Configuring Alertmanager receivers via web console is easier than writing the configuration manually. However, the time will come when you want to configure receivers automatically or with more advanced settings not exposed on the web console. In order to do so, you adapt the `alertmanager-main` secret in the `openshift-monitoring` namespace. Have a look at how Alertmanager is currently configured. To do that, extract the `alertmanager-main` secret’s information in the `openshift-monitoring` namespace.

```
oc -n openshift-monitoring extract secrets/alertmanager-main
```

Task 3.2.2: Persistence and metrics retention

By default, the monitoring stack loses all scraped metrics on restarts because it does not persist its data. Let's make our Prometheus time series database persistent and increase the metrics retention to 2 days so the volume won't fill up.

To do so, edit the `cluster-monitoring-config` configmap in the `openshift-monitoring` namespace.

```
oc -n openshift-monitoring edit cm cluster-monitoring-config
```

Add the following snippet:

```
...
prometheusK8s:
  ...
  retention: 2d
  volumeClaimTemplate:
   spec:
     resources:
       requests:
         storage: 5Gi
...
```

It is advisable to persist Alertmanager itself, too. This allows Alertmanager to retain active alerts on restarts and therefore prevents existing alerts from triggering again. Edit the same configmap as before (`cluster-monitoring-config` in `openshift-monitoring`).

```
oc -n openshift-monitoring edit cm cluster-monitoring-config
```

Add the following config:

```
...
alertmanagerMain:
  ...
  volumeClaimTemplate:
   spec:
     resources:
       requests:
         storage: 1Gi
  ...
```

After adding the above configuration to the ConfigMap, the Cluster Monitoring Operator will create the required persistent volume claims automatically. Check for these new persistent volume claims.

```
oc -n openshift-monitoring get pvc
```

The output should look similar to this:

```
NAME                                       STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
alertmanager-main-db-alertmanager-main-0   Bound    pvc-49453e5f-1e04-4d00-88cb-cd04528c9700   1Gi        RWO            gp2            5m43s
alertmanager-main-db-alertmanager-main-1   Bound    pvc-42b8083b-5dc2-4218-a8e2-15111e0a87e7   1Gi        RWO            gp2            5m43s
alertmanager-main-db-alertmanager-main-2   Bound    pvc-824824e3-ee0b-49c6-a786-3efd86055acb   1Gi        RWO            gp2            5m42s
prometheus-k8s-db-prometheus-k8s-0         Bound    pvc-19894227-c8ad-4ac3-a793-71a4a409fd23   5Gi        RWO            gp2            6m32s
prometheus-k8s-db-prometheus-k8s-1         Bound    pvc-11fcf37e-5803-41e7-95ae-f2779010b201   5Gi        RWO            gp2            6m32s
```

3.3 User-defined monitoring rules

Task 3.3.1: Enable the user-workload monitoring stack

As mentioned in a lab before, it is not supported to alter the OpenShift-provided monitoring stack. If you want to add additional monitoring targets or add custom Prometheus rules, you need to enable the user-workload monitoring stack.

And this is exactly what we are going to do. First, edit the `cluster-monitoring-config` configmap in the `openshift-monitoring` namespace.

```
oc -n openshift-monitoring edit cm cluster-monitoring-config
```

Add the line `enableUserWorkload: true` under `data/config.yaml` (leave the rest as it is).

```
...
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
...
```

Saving the configmap will automatically deploy your own Prometheus instance in a newly created namespace `openshift-user-workload-monitoring`. Verify that all pods are in state `Running`.

```
oc -n openshift-user-workload-monitoring get pods
```

```
NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-764495967-z5tln    2/2     Running   0          74s
prometheus-user-workload-0             5/5     Running   1          65s
prometheus-user-workload-1             5/5     Running   1          65s
thanos-ruler-user-workload-0           3/3     Running   0          65s
thanos-ruler-user-workload-1           3/3     Running   0          65s
```

Optional: You may also want to enable storage and persistence for your user workload monitoring stack, which is done in the same way as you learned in the previous lab.

Task 3.3.2: Add a custom Prometheus rule

With the user-workload monitoring stack now running, you can add your own custom Prometheus rules. To do so, create a PrometheusRule custom resource with the following content:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    prometheus: k8s
    role: alert-rules
  name: monitoring-stack-alerts
spec:
  groups:
  - name: general.rules
    rules:
    - alert: TargetDown
      annotations:
        message: '{{ $labels.job }} targets in {{ $labels.namespace }} namespace are down.'
      expr: |2
          (
            count(up == 0) BY (job, namespace, service)
          /
            count(up) BY (job, namespace, service)
          )
          * 100
          > 10
      for: 10m
      labels:
        severity: warning
```

Either create a file with above content or apply it directly:

- acend gmbh

```
<redacted>
```

Let’s take a look at the main components that can be configured applying a custom PrometheusRule:

- **alert:** The alert’s name should be as short and precise as possible because it is used by most receivers as the title of the alert message.
- **annotations:** Human-readable metadata like a brief summary or a link to a playbook explaining the alert and actions on the alert.
- **expr:** PromQL expression that defines the condition in which Prometheus should trigger the alert
- **for:** Duration of the fulfilled condition until the alarm is triggered. If not set, alerts are triggered at their first occurrence.
- **labels:** Metadata that can be used to dynamically route the alert to a corresponding receiver.

Task 3.3.3: Allowing users to add monitoring resources

By default, only cluster administrators have access to the user-workload monitoring stack and are able to create Prometheus custom resources like PrometheusRules. You can grant platform users access to the stack by giving them one of the cluster roles below:

- `monitoring-rules-view` : View `PrometheusRule` resources in their projects
- `monitoring-rules-edit` : Edit `PrometheusRule` resources in their projects
- `monitoring-edit` : Edit `PrometheusRule` , `ServiceMonitor` and `PodMonitor` resources in their projects
- `user-workload-monitoring-config-edit` : Let’s users alter the `user-workload-monitoring-config` configmap in the `openshift-user-workload-monitoring` namespace

This is particularly useful as it allows users to change their monitoring configuration in a self-service manner.

3.4 Logging

In this lab we will install and configure the logging stack.

Task 3.4.1 Install cluster logging

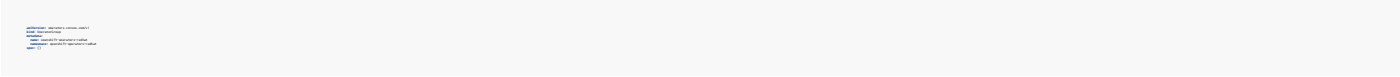
You can install OpenShift Logging to aggregate all the logs from your OpenShift cluster, such as node logs, application logs and infrastructure logs.

To deploy the cluster logging stack from the CLI, we need to create the following objects. Have a look at them first:

- OpenShift Elasticsearch Operator Namespace

- Red Hat OpenShift Logging Namespace

- OpenShift Elasticsearch Operator `OperatorGroup`



- Red Hat OpenShift Logging `OperatorGroup`

- OpenShift Elasticsearch Operator `Subscription`

- Red Hat OpenShift Logging `Subscription`

Now either copy and paste above resource definitions and then apply them to your cluster or directly use our provided files:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/ns_openshift-operators-redhat.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/ns_openshift-logging.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/og_openshift-operators-redhat.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/og_cluster-logging.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/sub_elasticsearch-operator.yaml
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/sub_cluster-logging.yaml
```

Verify if the Elasticsearch Operator installation succeeded. In order to find that out, check for the resource `clusterserviceversion` (or `csv` for short) in all namespaces.

```
oc get csv --all-namespaces
```

Example output:

```
NAMESPACE                                NAME                             DISPLAY                          VERSION   REPLACES   PHASE
default                                  elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
kube-node-lease                          elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
kube-public                              elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
kube-system                              elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
openshift-apiserver-operator             elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
openshift-apiserver                      elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
openshift-authentication-operator        elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
openshift-authentication                 elasticsearch-operator.5.1.2-7   OpenShift Elasticsearch Operator   5.1.2-7              Succeeded
...
```

There should be an OpenShift Elasticsearch Operator in each namespace. The version number might be different than shown.

Verify the Cluster Logging Operator installation, this time by checking for `clusterserviceversions` inside the `openshift-logging` namespace.

```
oc get csv -n openshift-logging
```

Example output:

```
NAME                      DISPLAY                     VERSION   REPLACES                   PHASE
...
cluster-logging.5.1.2-7   Red Hat OpenShift Logging   5.1.2-7   cluster-logging.5.1.1-36   Succeeded
...
```

There should be a Cluster Logging Operator in the openshift-logging namespace. The version number might be different than shown.

Now you can create an OpenShift Logging instance.

> **Note**
> Note the already baked-in tolerations.

The logging instance definition looks as follows:

```yaml
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage:
        size: 50G
      resources:
        requests:
          memory: "8Gi"
      proxy:
        resources:
          limits:
            memory: 256Mi
          requests:
             memory: 256Mi
      redundancyPolicy: "SingleRedundancy"
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
    retentionPolicy:
      application:
        maxAge: 1d
      infra:
        maxAge: 1d
      audit:
        maxAge: 1d
  visualization:
    type: "kibana"
    kibana:
      replicas: 1
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
  collection:
    logs:
      type: "fluentd"
      fluentd:
        tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/infra
```

Again, you can use the provided file or put above content in a file of your own:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/clusterlogging_instance.yaml
```

Note

The provided configuration is a basic setup that should support most use cases. For details on how to configure the logging stack, consult the chapter [About the Cluster Logging custom resource] in the official documentation.

Verify the installation by listing the pods in the openshift-logging project.

You should see several pods for OpenShift Logging, Elasticsearch, Fluentd, and Kibana.

```
oc get pods -n openshift-logging
```

Example output:

```
NAME                                            READY   STATUS    RESTARTS   AGE
cluster-logging-operator-664b7f76d8-v65hn       1/1     Running   0          12d
collector-74jkk                                 2/2     Running   0          12d
collector-g2djj                                 2/2     Running   0          12d
collector-gzrt5                                 2/2     Running   0          12d
collector-j2q86                                 2/2     Running   0          12d
collector-nkngv                                 2/2     Running   0          12d
collector-pbvhn                                 2/2     Running   0          12d
elasticsearch-cdm-0w2i8ldc-1-58b948bd84-cc7t2   2/2     Running   0          12d
elasticsearch-cdm-0w2i8ldc-2-7b6989ddb7-c78tg   2/2     Running   0          12d
elasticsearch-cdm-0w2i8ldc-3-64b5f55f58-9455z   2/2     Running   0          12d
kibana-687fdb6fb5-64jpw                         2/2     Running   0          12d
```

Task 3.4.2 Enable audit log forwarding

By default, the logging stack does not store the audit logs in Elasticsearch, since Elasticsearch does not provide encryption.
For the purpose of this lab you will configure the logging stack to store the audit logs in the central Elasticsearch cluster.

We can make use of the cluster log forwarding feature of the OpenShift Logging stack, which allows us to route logs to different log stores by defining forwarding pipelines.

Note

Because we do not have an external log store to forward our logs to, we will make use of the cluster-internal Elasticsearch instance. Check the documentation on [Forwarding logs to third party systems] to learn more about the supported third-party log stores.

To forward the audit logs to the internal Elasticsearch instance, we need to define a `ClusterLogForwarder` object:

```yaml
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  pipelines:
  - name: audit-default
    inputRefs:
    - audit
    outputRefs:
    - default
  - name: infrastructure-default
    inputRefs:
    - infrastructure
    outputRefs:
    - default
  - name: application-default
    inputRefs:
    - application
    outputRefs:
    - default
```

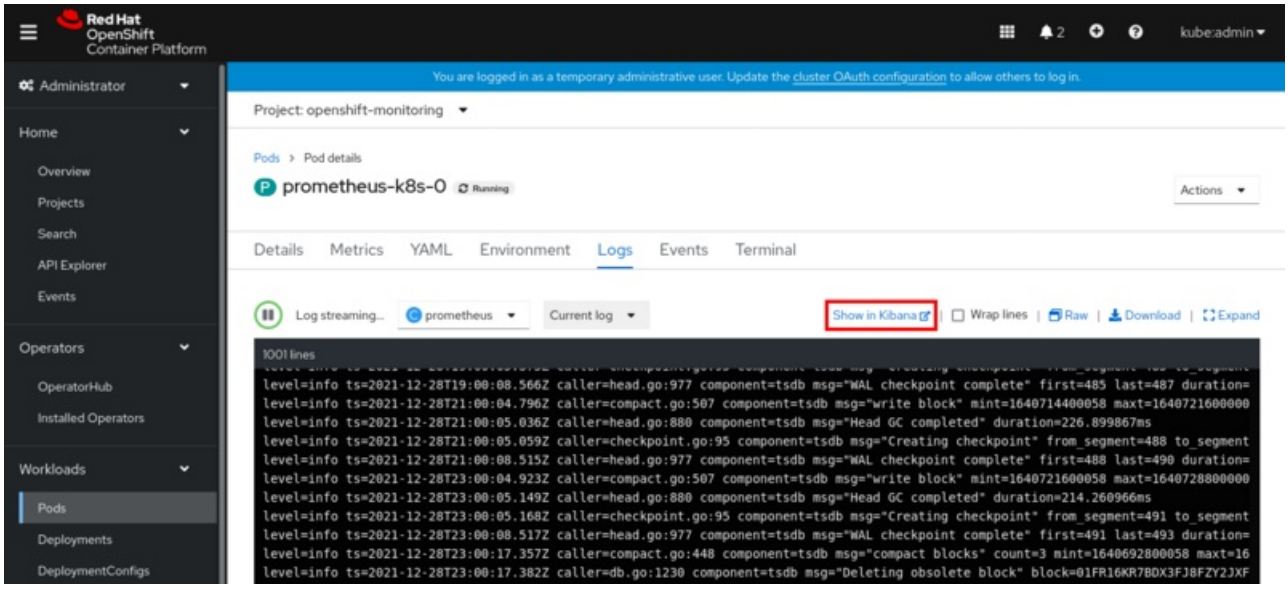
You can also use our provided file:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/03/resources/logging/clusterlogforwarder_instance.yaml
```

Task 3.4.3 Configure and use Kibana

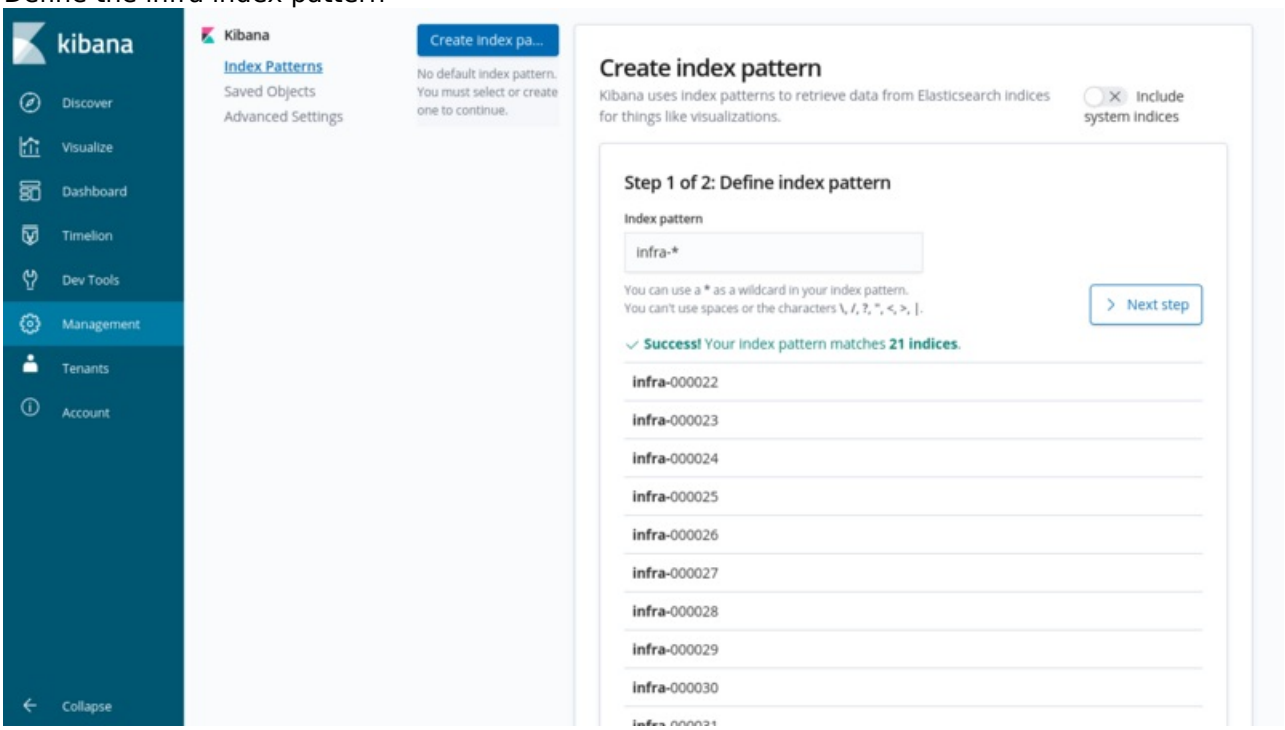
Now that you have installed and confugred the logging stack, it is time to check the log visualizer - Kibana.

To get to Kibana you can either click the Application Launcher and select **Logging**, or by clicking on **Show in Kibana** in the log browser of the console:



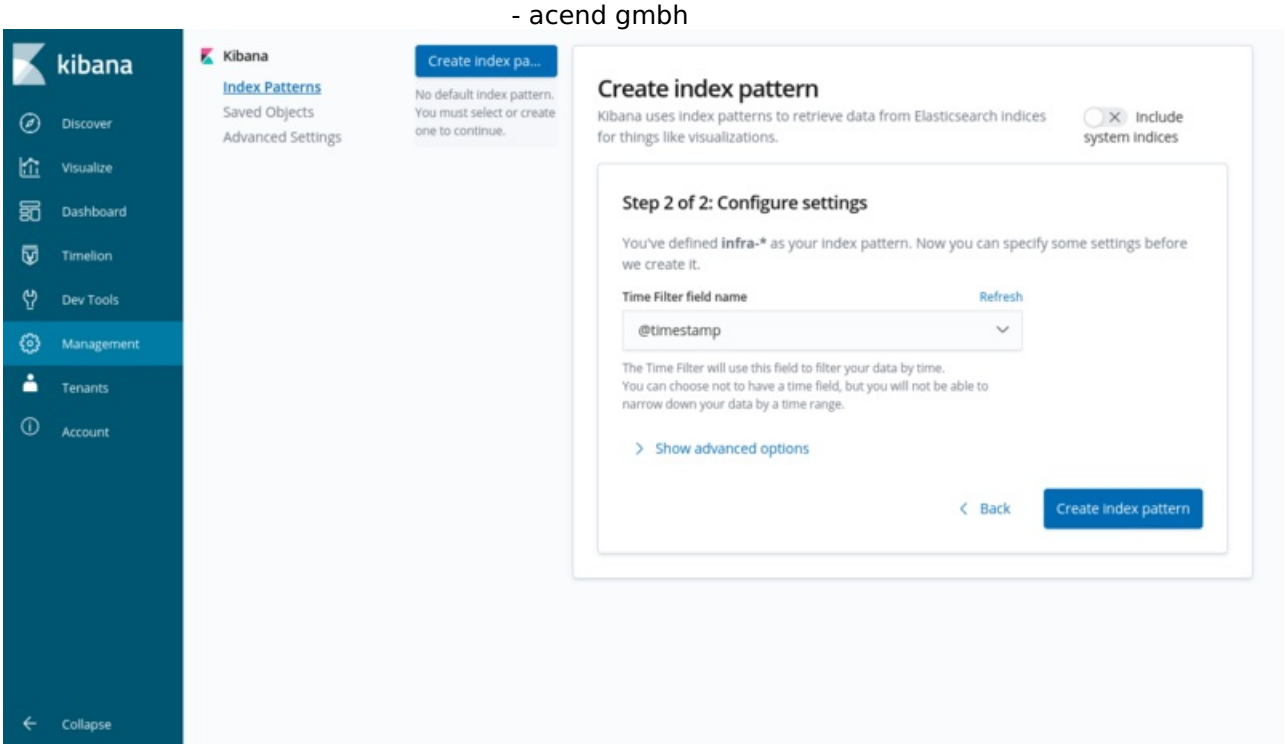
The log store of the logging stack (Elasticsearch) stores the logs in three index categories: Application, Infrastructure and, if enabled, Audit.
The first time you log in to Kibana, you need to create index patterns for your user:

- Create the infra index pattern:
  - Define the infra index pattern



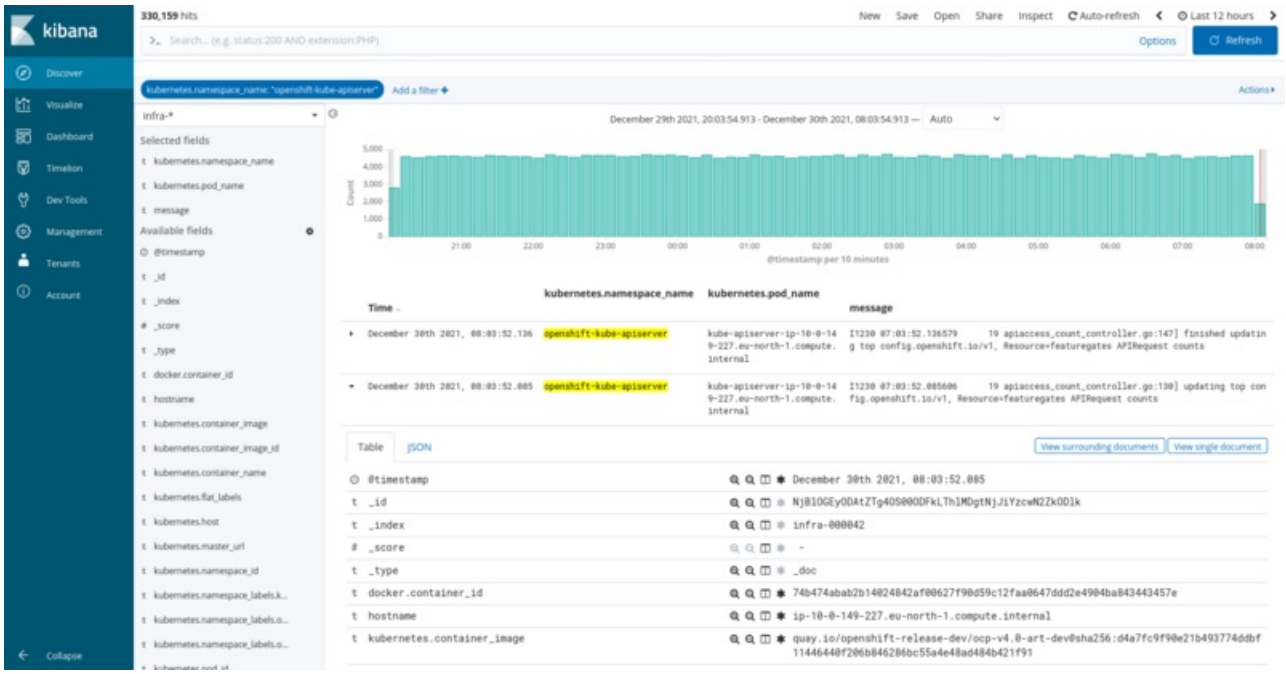
  - Select `@timestamp` as the filed name

- acend gmbh



- Repeat these steps for the remaining indices (audit, apps)
- Go to **Discover** to see the logs

The following screenshot shows the logs of the infra index for the pods in the namespace `openshift-kube-apiserver` for the last 12 hours:



Try to see the same logs on your cluster by exploring the features of Kibana.

- acend gmbh

4. Troubleshooting

This chapter is all about “what to do when …?”.

First, it shows you the basics of how to debug and analyze on OpenShift 4 using new and updated features and commands.

It then explains some of the more common problems and issues you might face when operating an OpenShift 4 cluster and what to do to solve them.

4.1 Troubleshooting basics

In chapter 3 you got to know the monitoring components, its rules and more. Now, imagine you get an alert. “Where to begin?”, you ask. How do you find out what’s amiss and led to the alert?

That’s what we are going to look at in this first lab.

Task 4.1.1: Cluster operators

As you already know, OpenShift 4 usually consists of about 30 different operators. Each operator is responsible that the actual state of the cluster matches the desired, configured state. If this is not the case, the operator tells us.

Check the cluster operators’ state.

```
oc get clusteroperators
```

> **Note**
>
> For most of the Kubernetes resource types, there’s a short version. In the case of the `clusteroperators` resources, it’s `co`, so you could also check the state with `oc get co`.

This gives you something along the lines of:

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
authentication                             4.1.0     True        False         False      16d
cloud-credential                           4.1.0     True        False         False      16d
cluster-autoscaler                         4.1.0     True        False         False      16d
console                                    4.1.0     True        False         False      16d
dns                                        4.1.0     True        False         False      16d
image-registry                             4.1.0     True        False         False      16d
ingress                                    4.1.0     True        False         False      16d
kube-apiserver                             4.1.0     True        False         False      16d
kube-controller-manager                    4.1.0     True        False         False      16d
kube-scheduler                             4.1.0     True        False         False      16d
machine-api                                4.1.0     True        False         False      16d
machine-config                             4.1.0     True        False         False      16d
marketplace                                4.1.0     True        False         False      16d
monitoring                                 4.1.0     True        False         False      16d
network                                    4.1.0     True        False         False      16d
node-tuning                                4.1.0     True        False         False      16d
openshift-apiserver                        4.1.0     True        False         False      16d
openshift-controller-manager               4.1.0     True        False         False      16d
openshift-samples                          4.1.0     True        False         False      16d
operator-lifecycle-manager                 4.1.0     True        False         False      16d
operator-lifecycle-manager-catalog         4.1.0     True        False         False      16d
operator-lifecycle-manager-packageserver   4.1.0     True        False         False      16d
service-ca                                 4.1.0     True        False         False      16d
service-catalog-apiserver                  4.1.0     True        False         False      16d
service-catalog-controller-manager         4.1.0     True        False         False      16d
storage                                    4.1.0     True        False         False      16d
```

The important part to see here is that all of the operators are available, that they are not progressing and that they are not degraded.

If you make a change to one of the operator’s configuration, the operator usually changes its progressing state to true as long as the configuration change has not finished.

Degraded means that the operator cannot function properly because there is, e.g., a configuration error.

Task 4.1.2: Configuration error

In order to test this behaviour, we are going to introduce a configuration change. Specifically, we want to add an authentication provider to our cluster:

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: github
    mappingMethod: claim
    type: GitHub
    github:
      clientID: ef60e25a4f8e2816a8f9
      clientSecret:
        name: github-secret
      organizations:
      - myorganization
```

Apply the authentication provider configuration to the cluster.

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/04/resources/oauth_cluster.yaml
```

It might take a while, but after a moment, the authentication operator will change its “degraded” state to true. Obviously something seems to be wrong with our introduced configuration; it’d be quite the coincidence something else happened at the same time. But we need to know what exactly, so let’s try to find out.

To do that, let’s have a closer look at the cluster operator responsible for the oauth configuration, the authentication operator. Execute a `describe` on it and look for a hint on what could be wrong.

```
oc describe clusteroperator authentication
```

The output will be rather long. Look for for the `Status` part to see the relevant message:

```
Status:
  Conditions:
    Last Transition Time:  2021-04-23T09:39:40Z
    Message:               OAuthServerConfigObservationDegraded: error validating secret openshift-config/github-secret: secret "github-secret" not found
    Reason:                OAuthServerConfigObservation_Error
    Status:                True
    Type:                  Degraded
    Last Transition Time:  2021-04-22T18:54:38Z
    Message:               All is well
    Reason:                AsExpected
    Status:                False
    Type:                  Progressing
    Last Transition Time:  2021-04-22T18:54:50Z
    Message:               OAuthServerDeploymentAvailable: availableReplicas==2
    Reason:                AsExpected
    Status:                True
    Type:                  Available
```

You can see that the operator’s status first changed from “Available” (`Type` field) to “Progressing” and then to “Degraded”. This means that it tried to apply our configuration change but then failed to do so.

Looking at the topmost “Message” reveals the problem: The configuration defines a secret in `clientSecret` which contains the GitHub client secret. This secret doesn’t exist (on purpose).

Now that you’ve found out what the problem is and how you can get the relevant troubleshooting information, revert the change so the authentication operator goes back to a functioning state.

```
oc patch oauth cluster --type merge -p '{"spec": {"identityProviders": []}}'
```

Task 4.1.3: Node debugging

OpenShift 4 introduces a new capability to debug cluster nodes. Using `oc debug node/<node name>` , a debug pod is started on the specified node and you get a terminal session in that debug pod.

Let’s try this out. Pick a node of your choosing.

```
oc get nodes
```

Then, using the node's name, start the debug session.

```
oc debug node/<node_name>
```

> **Warning**
> Note the `/` between the resource type (`node`) and the name. The command will error out if you don't write it.

After a brief moment you are presented with a root shell on the desired node. However, you're not yet really on the node, you're limited to what the container sees of it. You need to chroot onto the node before you'll be able to use host binaries. The message that appeared when you opened the debug shell conveniently provides the chroot command for us.

```
chroot /host
```

If you're not sure whether you're on the node, there are different possibilities to find out, one of which is looking at the release file.

```
cat /etc/redhat-release
```

If the output says `Red Hat Enterprise Linux release [...]` , you're still contained in the container. As we know, OpenShift nodes use CoreOS as operating system, so the output should read `Red Hat Enterprise Linux CoreOS release [...]` .

We can now do all the things as if we had connected via ssh. Which is still possible but might be less convenient.

One of these things you might want to do is have a look at the running containers on that node. We do this using `crictl` .

```
crictl ps
```

As you probably know from the good old days when you still used Docker on your machines, `ps` lists all running containers.

`crictl` is also aware of pods, so as a next task, list all pods.

```
crictl pods
```

To exit the node debug pod, either simply type `exit` or press ctrl+d until you're back on your own shell.

For more information on `crictl` , check out this debugging how-to or its documentation .

Task 4.1.4: Node logs

You might be tempted now to use the `oc debug node` feature to use it for all kinds of debugging tasks.

However, there is one particularly useful `oc adm` subcommand that comes in especially handy when you want to look at a node's logs.

`oc adm node-logs` allows you to retrieve node logs. By default, the command queries the systemd journal. Using `--path`, you can override this behaviour and use it to show logs within the node's `/var/logs` directory.

Let's use this to query all masters' (`--role master`) kubelet (`--unit`) logs.

```
oc adm node-logs --role master --unit kubelet --tail 10
```

> **Note**
> As with the `journalctl` command you can use `-tail`, `-since` and `-until` to limit the amount of logs or to focus on a certain period of time.

Of course above command can be used for all the other systemd units that are running on a node.

As already mentioned, the `--path` parameter can be used to show logs in the `/var/logs` directory. The following command lists all available log directories:

```
oc adm node-logs --role master --path /
```

This way, we can find a specific log file and finally show its content, e.g. the audit log's:

```
oc adm node-logs --role master --path audit/audit.log
```

> **Note**
> You might have to interrupt the `node-logs` command using `--path` with ctrl+c depending on the number of logs. The `--tail`, `--since` and `--until` parameters cannot be used to with `--path`.

Task 4.1.5: Node resource usage

Belonging to the same category of particularly useful commands is `oc adm top`. It can be used to display usage statistics of images, imagestreams, nodes and pods.

Want to know which image uses the most space on your nodes? `oc adm top images` is your friend. It works very similarly for `ImageStreams` objects.

A command providing you with a nice overview of resource usage on your nodes (without using Prometheus) is `oc adm top node`, which also works with the `--selector` parameter to list all nodes that match the selected label:

```
oc adm top node --selector node-role.kubernetes.io/worker
```

Finally, let's check the uptime app's resource consumption:

```
oc adm top pods --namespace uptime-sprint
```

Task 4.1.6: SSH

You might be wondering why all these new subcommands were introduced with OpenShift 4. Why not simply ssh into a node you want to analyze?

The reason is the introduction of CoreOS as the default underlying operating system of OpenShift 4. CoreOS changes the way we interact with an operating system because it follows the core principles of containers:

- immutability
- statelessness

This means that it is discouraged to ssh into a node because this might introduce manual, undocumented changes to the operating system that could lead to unexpected behaviour. Always apply configuration changes via `MachineConfig` objects instead.

However, there are cases where ssh represents the only viable means of analyzing a malfunctioning node. Imagine the OpenShift API is down or the node's kubelet is not responding. `oc` will not work in these cases. So it still is a good idea to configure ssh keys in those installation configuration files.

In order to connect to any node, first find out the node's hostname. Fortunately, and this should always be the case, AWS uses the fully-qualified hostnames as node names. Get a hostname:

```
```

> **Note**
> Note that the ssh command will not work! This is due to this training's specific network segmentation. We added the command for reference nonetheless.

The only remaining piece left to know about is that you always have to use username `core` when connecting to a CoreOS OpenShift node:

```
```

Troubleshooting references

The OpenShift documentation has multiple troubleshooting documentation pages such as this one which are worth checking out.

4.2 Evictions

This second troubleshooting lab will go into detail on what evictions are, how to analyze and how to avoid them.

Task 4.2.1: Setup

First, let’s get set up. Create a new namespace using the following command:

Warning

It is essential that you use this command or else this lab won’t work!

```
oc create namespace <namespace>
```

Now, deploy a test pod into the freshly created namespace using the provided definition:

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/04/resources/deployment_stress2much.yaml --namespace <namespace>
```

Watch the deployment and wait for a new event after the pod successfully ran for some time (give it 1 to 2 minutes).

```
oc get pods --watch --namespace <namespace>
```

Note

As always with `-watch` / `-w` , terminate the command using ctrl+c.

What you should see is that the pod gets evicted and recreated periodically.

Task 4.2.2: Analysis

So what happened? Why does the pod periodically stop running and show the status `Evicted` ?

Try to find an answer to these questions.

There are at least two ways to find that out: Describing an evicted pod shows its events where you’ll find the reason:

```
oc describe pod <evicted pod> --namespace <namespace>
```

```
...
Events:
  Type     Reason          Age   From               Message
  ----     ------          ----  ----               -------
  Normal   Scheduled       27s   default-scheduler  Successfully assigned eviction-lab/stress2much-945db9-74z7m to host-10-110-0-110
  Normal   AddedInterface  24s   multus             Add eth0 [10.125.2.107/23]
  Normal   Pulling         24s   kubelet            Pulling image "quay.io/acend/stress"
  Normal   Pulled          22s   kubelet            Successfully pulled image "quay.io/acend/stress" in 1.670548529s
  Normal   Created         22s   kubelet            Created container stress
  Normal   Started         22s   kubelet            Started container stress
  Warning  Evicted         7s    kubelet            The node was low on resource: memory. Container stress was using 5093448Ki, which exceeds its request of 0.
  Normal   Killing         7s    kubelet            Stopping container stress
```

Depending on the pods running on the cluster, it may be the case that you already had to scale down or delete the culprit(s). This would mean you don’t have any pods to describe. However, looking at all the collected events from the namespace’s resources, we can still find out what happened:

```
oc get events --namespace <namespace>
```

```
...
34s         Normal    Scheduled          pod/stress2much-945db9-74z7m       Successfully assigned eviction-lab/stress2much-945db9-74z7m to host-10-110-0-110
32s         Normal    AddedInterface     pod/stress2much-945db9-74z7m       Add eth0 [10.125.2.107/23]
32s         Normal    Pulling            pod/stress2much-945db9-74z7m       Pulling image "quay.io/acend/stress"
30s         Normal    Pulled             pod/stress2much-945db9-74z7m       Successfully pulled image "quay.io/acend/stress" in 1.670548529s
30s         Normal    Created            pod/stress2much-945db9-74z7m       Created container stress
30s         Normal    Started            pod/stress2much-945db9-74z7m       Started container stress
15s         Warning   Evicted            pod/stress2much-945db9-74z7m       The node was low on resource: memory. Container stress was using 5093448Ki, which exceeds its request of 0.
15s         Normal    Killing            pod/stress2much-945db9-74z7m       Stopping container stress
...
```

Task 4.2.3: Cleanup

The pod’s eviction and recreation would repeat endlessly, so let’s scale down the deployment.

```
oc scale deployment/stress2much --replicas 0 --namespace <namespace>
```

Notice how many pods were created and evicted in the meantime:

```
...
stress2much-945db9-pt2kr   0/1     Evicted   0          7m33s
stress2much-945db9-qf5gc   0/1     Evicted   0          7m31s
stress2much-945db9-qq52c   0/1     Evicted   0          7m28s
stress2much-945db9-rgtsp   0/1     Evicted   0          7m35s
stress2much-945db9-rq4mh   0/1     Evicted   0          7m30s
stress2much-945db9-rwtwl   0/1     Evicted   0          7m29s
stress2much-945db9-rz7kg   0/1     Evicted   0          7m32s
stress2much-945db9-s8gxl   0/1     Evicted   0          7m34s
stress2much-945db9-sgvqm   0/1     Evicted   0          7m33s
stress2much-945db9-smq8f   0/1     Evicted   0          7m31s
stress2much-945db9-tg4q2   0/1     Evicted   0          7m30s
stress2much-945db9-tm2q5   0/1     Evicted   0          7m29s
stress2much-945db9-tvjfs   0/1     Evicted   0          7m29s
stress2much-945db9-v9grm   0/1     Evicted   0          7m30s
stress2much-945db9-vjb4v   0/1     Evicted   0          7m59s
stress2much-945db9-vk2qk   0/1     Evicted   0          7m31s
stress2much-945db9-vn86d   0/1     Evicted   0          7m32s
stress2much-945db9-wkpkx   0/1     Evicted   0          7m31s
stress2much-945db9-wpf6t   0/1     Evicted   0          7m33s
stress2much-945db9-xq5t2   0/1     Evicted   0          7m34s
stress2much-945db9-z6ddv   0/1     Evicted   0          7m30s
stress2much-945db9-z8j6j   0/1     Evicted   0          7m33s
stress2much-945db9-z8v7c   0/1     Evicted   0          7m32s
stress2much-945db9-zk5pn   0/1     Evicted   0          7m28s
stress2much-945db9-zvz7p   0/1     Evicted   0          7m34s
stress2much-945db9-zwmzk   0/1     Evicted   0          7m34s
...
```

In order to delete all the evicted pods at once, execute the following command:

```
oc delete pods --field-selector=status.phase=Failed --namespace <namespace>
```

Reasons

So we now know that the pod got evicted because it simply used too much memory. In contrast to an out-of-memory event caused by overstepping the defined memory limit on the container or pod though, this event happened because the worker node itself had no memory left.

It’s easy to see why the pod used too much memory:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: stress2much
  name: stress2much
spec:
  replicas: 1
  selector:
    matchLabels:
      app: stress2much
  template:
    metadata:
      labels:
        app: stress2much
    spec:
      containers:
      - name: stress
        image: quay.io/acend/stress
        command: ["stress"]
        args: ["--vm", "1", "--vm-bytes", "8G", "--vm-hang", "1"]
```

Have a good look at the last 3 lines: The deployment created at the beginning of this lab deployed a pod using an image containing the stress-testing tool `stress`. It also instructed the pod to execute `stress` with parameter `--vm-bytes 8G` which means the process tries to allocate 8GB of memory when started.

Limitrange

But why could the pod even allocate this much memory and was not killed before? Let's look at the LimitRange resource we defined earlier in the default project template:

```
oc get limitrange --namespace <namespace>
```

Now you know why it was crucial to create the namespace using `oc create namespace` at the beginning of this lab. This command only creates the namespace but does not respect the default project template. That is why the LimitRange object was never created.

> **Note**
> Notice how OpenShift automatically creates a corresponding Project resource when only a namespace is created.

Kubelet arguments

Now it's clear why the pod used too much memory and why it wasn't stopped from doing so earlier. The last remaining question is, why was it evicted?

Remember the kubelet arguments you configured after installing the cluster in lab 1.2 ? We never looked at them in greater detail, so let's do that now:

```
machineconfiguration.openshift.io/v1
KubeletConfig
metadata:
  name: worker
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  kubeletConfig:
    evictionHard:
      memory.available: 500Mi
      nodefs.available: 10%
      nodefs.inodesFree: 5%
    evictionSoft:
      memory.available: 1Gi
    evictionSoftGracePeriod:
      memory.available: 1m30s
    kubeReserved:
      memory: 1Gi
    systemReserved:
      memory: 1Gi
```

There's only three arguments defined, but those three were what saved our worker nodes from completely crashing.

- `kubeReserved` and `systemReserved` do exactly what their names suggest: They reserve a certain amount of resources for the kubelet and the operating system, respectively. This means that pod workload cannot, e.g., use more than the node's total memory less the reserved values.
- `evictionHard` defines a threshold which, when exceeded, triggers the eviction of pods. Pods are evicted based on their quality of service class which we already looked at during the Kubernetes/OpenShift Basics training.

There's also the `evictionSoft` argument which only starts to evict pods if node resources were over a certain threshold for a defined period of time. However, the `evictionHard` argument should always be present as a last resort in case too many resources are consumed too fast.

Conclusion

A properly configured cluster is well-defended against any kinds of resource overuse. If however you run into any sort of evictions, you now know where to optimize parameters.

Additionally, proper capacity management is always part of operating an OpenShift cluster. So make sure to set one up and add new nodes when needed, or even configure node autoscaling if possible.

- acend gmbh

5. GitOps

This chapter explores how to use GitOps processes and tools. GitOps is a way to do Kubernetes cluster management and application delivery. It works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git and what’s running in a cluster, and if there’s a difference, automatically update or rollback the cluster depending on the case.

Argo CD is such a software agent, or GitOps tool. It is a declarative, GitOps continuous delivery tool for Kubernetes.

Argo CD follows the GitOps pattern of using Git repositories as the source of truth for defining the desired application state. Kubernetes manifests can be specified in several ways:

- kustomize applications
- helm charts
- ksonnet applications
- jsonnet files
- Plain directory of YAML/json manifests
- Any custom config management tool configured as a config management plugin

Argo CD automates the deployment of the desired application states in the specified target environments. Application deployments can track updates to branches, tags, or pinned to a specific version of manifests at a Git commit. See tracking strategies for additional details about the different tracking strategies available.

For a quick 10 minute overview of Argo CD, check out the demo presented to the Sig Apps community meeting:

5.1 Setup

In order to explore GitOps processes, we are going to need appropriate tools: Gitea and ArgoCD.

Task 5.1.1: Gitea

Gitea is a lightweight Git server written in Go. It allows us to easily host a Git repository which we can use as our single source of truth.

Unfortunately, the default Operator catalog sources don’t contain a Gitea operator, but let’s change that.

Create the Gitea Operator by applying the YAML files from the Github repository:

```
oc apply -k https://github.com/rhpds/gitea-operator/OLMDeploy
```

Now check the status of the Gitea Operator in the web console.

- In the **Administrator** view of your web console, navigate to **Operators**, then **Installed Operators**
- Filter for “Gitea Operator” and wait for the Operator to finish its installation successfully

This was just the Operator part. We now have to create a Gitea instance.

First, create a new project.

```
oc new-project gitea
```

The instance you are going to create has the following content:

```yaml
apiVersion: pfe.rhpds.com/v1
kind: Gitea
metadata:
  name: gitea
  namespace: gitea
spec:
  giteaImageTag: latest
  giteaVolumeSize: 1Gi
  giteaSsl: true
  postgresqlVolumeSize: 1Gi
```

Create the instance.

```
oc apply -f https://raw.githubusercontent.com/acend/openshift-operations-training/main/content/en/docs/05/resources/gitea_gitea.yaml
```

That’s it! Watch the pods start.

```
oc -n gitea get pods -w
```

As soon as the postgresql as well as the gitea pods are running, get the hostname from the route.

```
oc -n gitea get route gitea -o go-template='https://{{ .spec.host }}{{ "\n" }}'
```

Open the URL in your browser and register yourself a user. Note the username and password you chose, you will need them later.

- acend gmbh

Task 5.1.2: Argo CD

We are going to install Argo CD in the form of the OpenShift GitOps Operator. Install the GitOps Operator via OperatorHub in OpenShift's web console.

- Head over to the **OperatorHub** on your cluster, filter for "gitops" and choose **Red Hat OpenShift GitOps**
- Click **Install**
- Leave the pre-filled values as-is and again click **Install**

This installs a nearly ready-to-use Argo CD instance. You can see that when looking into the `openshift-gitops` namespace:

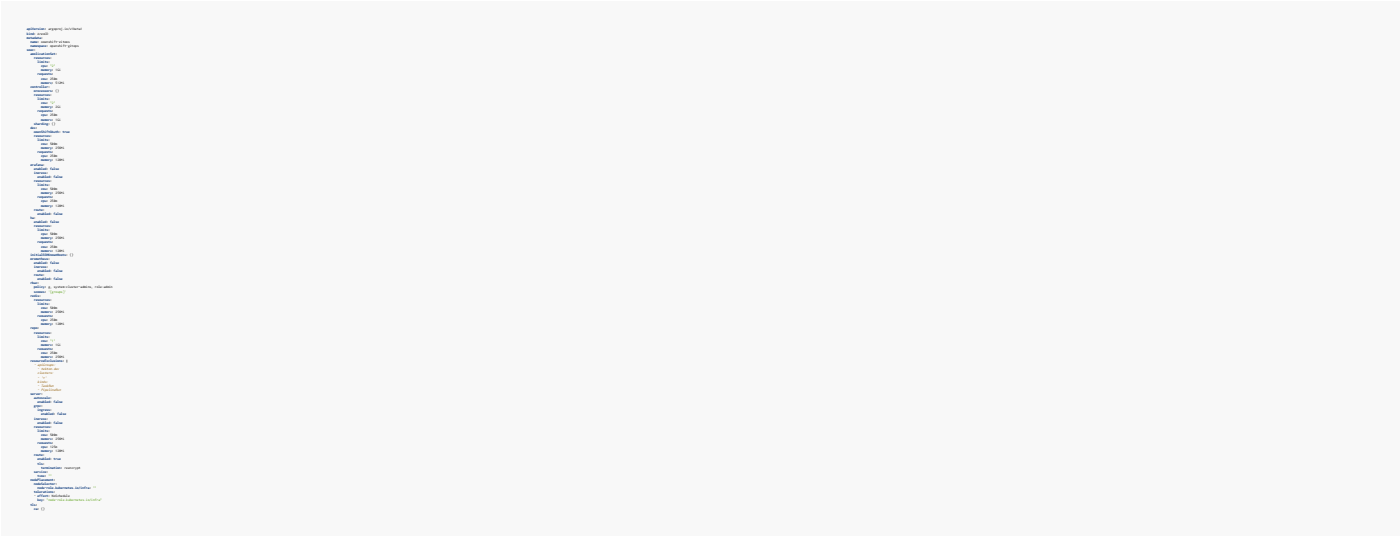
- An `argocd` custom resource named `openshift-gitops` was created
- This in turn led the Operator to create all the pods and routes you can see

What we need to do to make it fully operational is slightly adjust certain parameters in its custom resources configuration:

- Add tolerations and node selectors to make all pods run on infra nodes
- Change the route's termination to reencrypt

Apply the following resources.

- The Argo CD custom resource to change the route termination and add the node placement definition:



- The GitOpsService custom resource to move the Operator itself onto infra nodes as well:

5.2 Argo CD

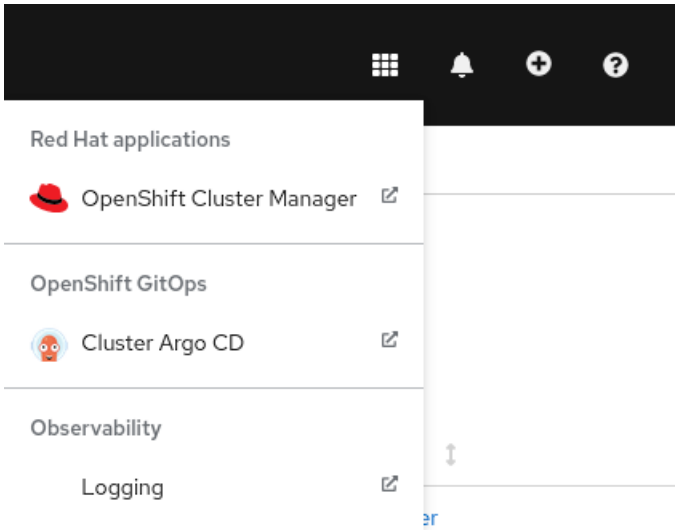
Task 5.2.1: Argo CD

Go out into the new world of GitOps and explore Argo CD.

Find out how to access Argo CD's web interface (which is very handy).

As usual, there are multiple ways to find out what Argo CD's dashboard URL is:

- OpenShift exposes the link via its web console:



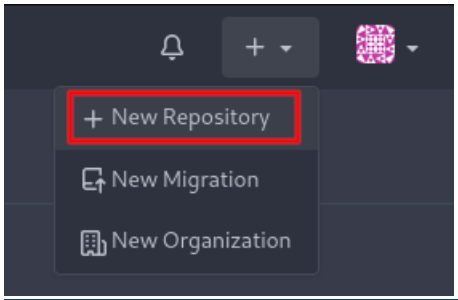
- Or get it via route from the `openshift-gitops` namespace:

Task 5.2.2: Gitea repository

Create a new Git repository on your Gitea instance.

> **Warning**
> If you want to add Secrets with sensitive data, make sure the repository is not public!

- On the upper right corner, click on the **+** icon and then choose **New Repository**:

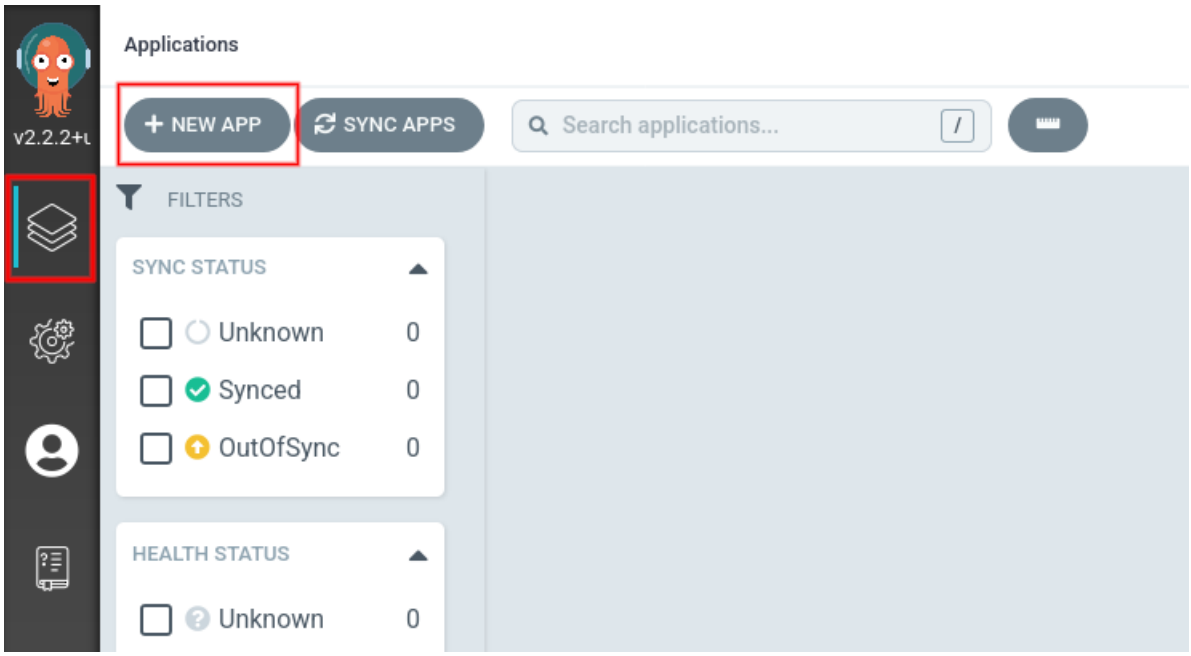


- On the **New Repository** page, choose a name such as “argocd-gitops-test”
- Choose the repo’s visibility
- On the bottom of the page, click on the **Create Repository** button

Task 5.2.3: Argo CD application

Figure out how to create an application based on the Git repository you just created.

First, make sure you are in the **Applications** view, then click on **+ NEW APP**:



Now fill in the appropriate information:

- **GENERAL**
  - **Application Name**: anything of your choosing
  - **Project**: `default`
  - Leave the rest as-is
- **SOURCE**
  - **Repository URL**: The URL to your Git repository
  - **Path**: Indicate in what folder the resource files reside; leave empty if they are in the root folder
  - Leave the rest as-is
- **DESTINATION**
  - **Cluster URL**: `https://kubernetes.default.svc`
  - **Namespace**: Leave empty

Task 5.2.4: Resource management

- Choose some of the resources you created in the course of this training, e.g.:
  - The `KubeletConfig` resources from task 1.3.1
  - The `MachineSet` resource from task 2.1.1
  - The `IngressController` definition
  - The custom Prometheus rule from task 3.3.2
- Add these resource definitions to your Git repository

> **Note**
> If you are unfamiliar with Git, check out Git's tutorial introduction to Git or ask your trainer.

- Create a new directory for your repository files:

```
mkdir clusterconfigs
cd clusterconfigs
```

- Initialize it for usage with Git:

```
git init
git config --add private-repo= git repository's URL>
git push -u origin main
```

- Taking the KubeletConfig from task 1.3.1 as an example, save its definition as a new file inside the Git directory and push it to your Gitea repository:

```
wget https://raw.githubusercontent.com/acend/openshift-training/main/content/en/docs/01/resources/kubeletconfig_master.yaml
git add kubeletconfig_master.yaml
git commit -m "Add kubelet's kubelet configuration"
git push
```

- When visiting your Gitea web interface, you should now see the added file(s)
- Apply your chosen resources using Argo CD

- acend gmbh

6. Cleanup

Only one task left to do!

6.1 Destroy the cluster

Usually our work centers around keeping everything up and running, building and engineering stuff and so on. This is not always easy, so there’s some extra joy when it comes to finally be allowed to do the opposite and destroy something :)

Task 6.1.1: Destroy your cluster

Change into the installation directory you created on day 1, it should be under `/home/ec2-user/ocp4-ops` and have a timestamp as its name.

Destroy the cluster:

```
openshift-install destroy cluster
```

Above command removes all created resources on AWS including VM instances, load balancers etc.

The really only thing left to do now is: Please inform your trainer that you destroyed your cluster.

Thank you!