You can access this page also inside the Remote Desktop by using the icons on the desktop

- Score
- Questions and Answers
- Preview Questions and Answers
- Exam Tips

CKA Simulator A Kubernetes 1.33

Each question needs to be solved on a specific instance other than your main `candidate@terminal`. You'll need to connect to the correct instance via ssh, the command is provided before each question. To connect to a different instance you always need to return first to your main terminal by running the `exit` command, from there you can connect to a different one.

In the real exam each question will be solved on a different instance whereas in the simulator multiple questions will be solved on same instances.

Use `sudo -i` to become root on any node in case necessary.

Question 1 | Contexts

Solve this question on: `ssh cka9412`

You're asked to extract the following information out of kubeconfig file `/opt/course/1/kubeconfig` on `cka9412`:

1. Write all kubeconfig context names into `/opt/course/1/contexts`, one per line
2. Write the name of the current context into `/opt/course/1/current-context`
3. Write the client-certificate of user `account-0027` base64-decoded into `/opt/course/1/cert`

Answer:

All that's asked for here could be extracted by manually reading the kubeconfig file. But we're going to use kubectl for it.

Step 1

First we get all context names:

```
➜ ssh cka9412
```

```
→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config get-contexts
CURRENT     NAME            CLUSTER      AUTHINFO                    NAMESPACE
            cluster-admin   kubernetes   admin@internal
            cluster-w100    kubernetes   account-0027@internal
*           cluster-w200    kubernetes   account-0028@internal


→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config get-contexts -oname
cluster-admin
cluster-w100
cluster-w200


→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config get-contexts -oname >
/opt/course/1/contexts
```

This will result in:

```
# cka9412:/opt/course/1/contexts
cluster-admin
cluster-w100
cluster-w200
```

We could also do extractions using jsonpath:

```
k --kubeconfig /opt/course/1/kubeconfig config view -o yaml

k --kubeconfig /opt/course/1/kubeconfig config view -o jsonpath="{.contexts[*].name}"
```

But it would probably be overkill for this task.

Step 2

Now we query the current context:

```
→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config current-context
cluster-w200

→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config current-context >
/opt/course/1/current-context
```

Which will result in:

```
# cka9412:/opt/course/1/current-context
cluster-w200
```


Step 3

And finally we extract the certificate and write it base64 decoded into the required location:

```
→ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config view -o yaml --raw
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC...
    server: https://10.30.110.30:6443
```

```
    name: cluster1
contexts:
- context:
    cluster: kubernetes
    user: admin@internal
  name: cluster-admin
- context:
    cluster: kubernetes
    user: account-0027@internal
  name: cluster-w100
- context:
    cluster: kubernetes
    user: account-0028@internal
  name: cluster-w200
current-context: cluster-w200
kind: Config
preferences: {}
users:
- name: account-0027@internal
  user:
    client-certificate-data:
```
```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN2RENDQWFRQ0ZIWWRqU1pGS0N5VUNSMUIybmFYQ2cvVWpTSExNQTBHQ1NxR1NJYjNE
UUVCQ3dVQU1CVXgKRXpBBBUkJnTlZCQU1UQ210MVltVnlibVYwW1hNd0hoY05NalF4TURJNE1Ua3dPVFV3V2hjTk1qWXdNekV5TVRrdwpPVFV3
V2pBZ01SNHdIQVlEVlFRERCVmhZMk52ZFcxMExUQXdNamRBYVc1MFpYSnVZd3dnZ2dFaU1BMEdDU3FHC1NJYjNEUUVCUVFVQUE0SUJEd0F3
Z2dFS0FvSUJBUURwVXNRRERFVys0OEF2Wm1LYktTMndtc1phMGd5K2t6aWgKY25EcFpnOG1nTy01MG5OaEhRNCRDQqAjRzNmRHmPbmJ1ajBz
WkdYZitObzhPSjJkVkw5andHU3Q1TlZvTHJqagpUd1BGNjN+KzR+alRCeExCdj64NzlpUFhYazBnaVAzOFRBb1MvL2R0SitPOGlzYlJNbmxi
OWFJNUwySjlYxXHfN
K2c0azJvNmhuM3U3sk0vY05iSEZNNT1MycVNCR3TtMjY2Sk0dk10QytjcHNtcHlvVW5EaEVyMXRUYTR2ZwpsYnB3nnZ4bkpjd010NG4wS2FB
ZVMwajRMMM09DODY5YwxweTFqdkkzQVRqRm1Y2tMRVNMQWdNQkFBRXdEUVlKCktvWklodmNOQQVFFTEJRQURnZ0VCQURRUUxHWVpvVVNyYnBn
RlY2OXNIdk11b3huMllvdDFDTUZCbVFyeHdPbGkKZGVtOTM2cTJaTE1yMzRyUTVyQzF1VFFEcmFXR0E0NHlIbVZaMDd0ZEloa1Yydm9JZXhI
alg5MWdwQytMaXJRcQpJS0d4aW9rOUNLTEU3TlJlRjYzcHAvN0JOZTcvUDZjT1JooME8yRURNNFRnSFhMcFhydDd0ZFBFWHd2ck4xdE1RCno1
YXY5UG81VGQ0VmYwcGFPRHRsYWh3aElaNks3Y3RnvkdUMtkUWxuMXFYRGIvVndxM1Z5WUJBSktsbU91OWwkYmozbm12YzdEOTllOXA0eTRH
RkNrQwxieHY5VEQwVDR5dllnVkZOUlRWYkdBa21hendVSHJmY1FuUlRWZktvegpTZnNZUnk2TDFSS3hqd2g3NEtuaEtKeiswOUpxWHByN01W
ZFFnamgwUmR3PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
```
```
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVk...
...
```

Instead of using `--raw` to see the sensitive certificate information, we could also simply open the kubeconfig file in an editor. No matter how, we copy the whole value of `client-certificate-data` and base64 decode it:

```
➜ candidate@cka9412:~$ echo LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN2RE... | base64 -d >
/opt/course/1/cert
```

Or if we like it automated:

```
➜ candidate@cka9412:~$ k --kubeconfig /opt/course/1/kubeconfig config view --raw -ojsonpath="
{.users[0].user.client-certificate-data}" | base64 -d > /opt/course/1/cert
```

Which will result in:

```
# cka9412:/opt/course/1/cert
-----BEGIN CERTIFICATE-----
MIICvDCCAaQCFHYdjSZFKCyUCR1B2naXCg/UjSHLMA0GCSqGSIb3DQEBCwUAMBUx
EzARBgNVBAMTCmt1YmVybmV0ZXMwHhcNMjQxMDI4MTkwOTUwWhcNMjYwMzEyMTkw
OTUwWjAgMR4wHAYDVQQDDBVhY2NvdW50LTAwMjdAaW50ZXJuYwwwggEiMA0GCSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQDpUsQDDEW+48AvZmKbKS2wmsZa0gy+kzii
cZDpZg8mgO+50jNnHQ4ICqAjG3fFHmPnbuj0sZGXf+ym0J2dVL9jwGSt5NVoLrjj
TwlBG63+k4jRBxLBv6479iPXXk0giP38TAoS//dtJ+O8isbRMnlb9aI5L2JYxHfN
```

```
VL2qrF9arLf1DN+h0havFxn9noJ/iZx/qb/FHgeZqnTf7zR6OouRuWHG523jnTqA
06K+g4k2o6hg3u7JM/cNbHFM7S2qSBDkS266JJtvMtC+cpsmg/eUnDhA21tTa4vg
1bpw6vxnJcwMt4nOKaAeSOj4L3OC869alpy1jvI3ATjFjuckLESLAgMBAAEwDQYJ
KoZIhvcNAQELBQADggEBADQQLGYZoUSrbpgFV69sHvMuoxn2YUt1B5FBmQrxwOli
dem936q2ZLMr34rQ5rC1uTQDraWXa44yHmVZO7tdINkV2voIexHjX91gVC+LirQq
IKGxiok9CKLE7NReF63pp/7BNe7/P6cORh0O2EDM4TgHXLpXrt7tdPEXwvrN1tMQ
z5av9Po5Td4Vf0paODtlahwhIZ6K7ctgVGT1KdQln1qXDb/Vwq3VyYBAJKlmOu9l
bj3nmvc7D99e9p4y4GFCkAlbxv9TDOT4yvYgVFtRTVbGAkmazwUHrfcQnRTVfKoz
SfsYRy6L1RKxjwh74KnhKJz+O9JqXpr7MVdQgjhORdw=
-----END CERTIFICATE-----
```

Task completed.

Question 2 | MinIO Operator, CRD Config, Helm Install

Solve this question on: `ssh cka7968`

Install the MinIO Operator using Helm in *Namespace* `minio`. Then configure and create the *Tenant* CRD:

1. Create *Namespace* `minio`

2. Install Helm chart `minio/operator` into the new *Namespace*. The Helm Release should be called `minio-operator`

3. Update the `Tenant` resource in `/opt/course/2/minio-tenant.yaml` to include `enableSFTP: true` under `features`

4. Create the `Tenant` resource from `/opt/course/2/minio-tenant.yaml`

ℹ It is not required for MinIO to run properly. Installing the Helm Chart and the *Tenant* resource as requested is enough

Answer:

Helm Chart: Kubernetes YAML template-files combined into a single package, *Values* allow customisation

Helm Release: Installed instance of a *Chart*

Helm Values: Allow to customise the YAML template-files in a *Chart* when creating a *Release*

Operator: *Pod* that communicates with the Kubernetes API and might work with `CRDs`

CRD: Custom Resources are extensions of the Kubernetes API

Step 1

First we create the requested *Namespace* `minio`:

```
➜ ssh cka7968

➜ candidate@cka7968:~$ k create ns minio
namespace/minio created
```

Step 2

Now we install the MinIO Helm chart into it and name the release `minio-operator`:

```
→ candidate@cka7968:~$ helm repo list
NAME    URL
minio   http://localhost:6000

→ candidate@cka7968:~$ helm search repo
NAME            CHART VERSION    APP VERSION    DESCRIPTION
minio/operator  6.0.4            v6.0.4         A Helm chart for MinIO Operator

→ candidate@cka7968:~$ helm -n minio install minio-operator minio/operator
NAME: minio-operator
LAST DEPLOYED: Sun Dec 22 17:04:37 2024
NAMESPACE: minio
STATUS: deployed
REVISION: 1
TEST SUITE: None

→ candidate@cka7968:~$ helm -n minio ls
NAME            NAMESPACE    REVISION    ...    STATUS      CHART            APP VERSION
minio-operator  minio        1           ...    deployed    operator-6.0.4   v6.0.4

→ candidate@cka7968:~$ k -n minio get pod
NAME                             READY    STATUS     RESTARTS    AGE
minio-operator-7b595f559d-5hrj5  1/1      Running    0           24s
minio-operator-7b595f559d-sl22g  1/1      Running    0           25s
```

Because we installed the Helm chart there are now some *CRDs* available:

```
→ candidate@cka7968:~$ k get crd
NAME                        CREATED AT
miniojobs.job.min.io        2024-12-22T17:04:38Z
policybindings.sts.min.io   2024-12-22T17:04:38Z
tenants.minio.min.io        2024-12-22T17:04:38Z
```

Just like we can create a *Pod*, we can now create a *Tenant*, *MinIOJob* or *PolicyBinding*. We can also list all available fields for the *Tenant CRD* like this:

```
→ candidate@cka7968:~$ k describe crd tenant
Name:         tenants.minio.min.io
Namespace:
Labels:       app.kubernetes.io/managed-by=Helm
Annotations:  controller-gen.kubebuilder.io/version: v0.15.0
              meta.helm.sh/release-name: minio-operator
              meta.helm.sh/release-namespace: minio
              operator.min.io/version: v6.0.4
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
Metadata:
  Creation Timestamp:  2024-12-22T17:04:38Z
  Generation:          1
  Resource Version:    15190
  UID:                 3407533d-785c-49df-96f2-c03af9f40749
Spec:
  Conversion:
    Strategy:  None
```

```
  Group:          minio.min.io
  Names:
    Kind:         Tenant
...
```

Step 3

We need to update the Yaml in the file which creates a *Tenant* resource:

```
→ candidate@cka7968:~$ vim /opt/course/2/minio-tenant.yaml
```

```yaml
apiVersion: minio.min.io/v2
kind: Tenant
metadata:
  name: tenant
  namespace: minio
  labels:
    app: minio
spec:
  features:
    bucketDNS: false
    enableSFTP: true                    # ADD
  image: quay.io/minio/minio:latest
  pools:
    - servers: 1
      name: pool-0
      volumesPerServer: 0
      volumeClaimTemplate:
        apiVersion: v1
        kind: persistentvolumeclaims
        metadata: { }
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 10Mi
          storageClassName: standard
        status: { }
  requestAutoCert: true
```

We can see available fields for `features` like this:

```
→ candidate@cka7968:~$ k describe crd tenant | grep -i feature -A 20
             Features:
               Properties:
                 Bucket DNS:
                   Type:  boolean
                 Domains:
                   Properties:
                     Console:
                       Type:   string
                     Minio:
                       Items:
                         Type:  string
                       Type:    array
                   Type:        object
                 Enable SFTP:
                   Type:  boolean
```

```
            Type:      object
        Image:
          Type:  string
        Image Pull Policy:
          Type:  string
        Image Pull Secret:
```

Step 4

Finally we can create the *Tenant* resource:

```
➜ candidate@cka7968:~$ k -f /opt/course/2/minio-tenant.yaml apply
tenant.minio.min.io/tenant created

➜ candidate@cka7968:~$ k -n minio get tenant
NAME     STATE                            HEALTH   AGE
tenant   empty tenant credentials                  21s
```

In this scenario we installed an operator using Helm and created a *CRD* with which that operator works. This is a common pattern in Kubernetes.

Question 3 | Scale down StatefulSet

Solve this question on: `ssh cka3962`

There are two *Pods* named `o3db-*` in *Namespace* `project-h800`. The Project H800 management asked you to scale these down to one replica to save resources.

Answer:

If we check the *Pods* we see two replicas:

```
➜ ssh cka3962

➜ candidate@cka3962:~$ k -n project-h800 get pod | grep o3db
o3db-0                                1/1     Running   0          6d19h
o3db-1                                1/1     Running   0          6d19h
```

From their name it looks like these are managed by a *StatefulSet*. But if we're unsure we could also check for the most common resources which manage *Pods*:

```
➜ candidate@cka3962:~ k -n project-h800 get deploy,ds,sts | grep o3db
statefulset.apps/o3db   2/2     6d19h
```

Confirmed, we have to work with a *StatefulSet*. We could also look at the *Pod* labels to find this out:

```
 ➜ candidate@cka3962:~ k -n project-h800 get pod --show-labels | grep o3db
 o3db-0                                    1/1     Running   0           6d19h
 app=nginx,apps.kubernetes.io/pod-index=0,controller-revision-hash=o3db-
 5fbd4bb9cc,statefulset.kubernetes.io/pod-name=o3db-0
 o3db-1                                    1/1     Running   0           6d19h
 app=nginx,apps.kubernetes.io/pod-index=1,controller-revision-hash=o3db-
 5fbd4bb9cc,statefulset.kubernetes.io/pod-name=o3db-1
```

To fulfil the task we simply run:

```
 ➜ candidate@cka3962:~ k -n project-h800 scale sts o3db --replicas 1
 statefulset.apps/o3db scaled

 ➜ candidate@cka3962:~ k -n project-h800 get sts o3db
 NAME    READY   AGE
 o3db    1/1     6d19h
```

The Project H800 management is happy again.

Question 4 | Find Pods first to be terminated

Solve this question on: `ssh cka2556`

Check all available *Pods* in the *Namespace* `project-c13` and find the names of those that would probably be terminated first if the nodes run out of resources (cpu or memory).

Write the *Pod* names into `/opt/course/4/pods-terminated-first.txt`.

Answer:

When available cpu or memory resources on the nodes reach their limit, Kubernetes will look for *Pods* that are using more resources than they requested. These will be the first candidates for termination. If some *Pods* containers have no resource requests/limits set, then by default those are considered to use more than requested. Kubernetes assigns Quality of Service classes to *Pods* based on the defined resources and limits.

Hence we should look for *Pods* without resource requests defined, we can do this with a manual approach:

```
 ➜ ssh cka2556

 ➜ candidate@cka2556:~$ k -n project-c13 describe pod | less -p Requests
```

Or we do something like:

```
 k -n project-c13 describe pod | grep -A 3 -E 'Requests|^Name:'
```

We see that the *Pods* of *Deployment* `c13-3cc-runner-heavy` don't have any resource requests specified. Hence our answer would be:

```
# /opt/course/4/pods-terminated-first.txt
c13-3cc-runner-heavy-65588d7d6-djtv9map
c13-3cc-runner-heavy-65588d7d6-v8kf5map
c13-3cc-runner-heavy-65588d7d6-wwpb4map
```

Automatic way

Not necessary and probably too slow for this task, but to automate this process you could use jsonpath:

```
➜ candidate@cka2556:~$ k -n project-c13 get pod -o jsonpath="{range .items[*]} {.metadata.name}
{.spec.containers[*].resources}{'\n'}"
 c13-2x3-api-c848b775d-7nggw{"requests":{"cpu":"50m","memory":"20Mi"}}
 c13-2x3-api-c848b775d-qrrlp{"requests":{"cpu":"50m","memory":"20Mi"}}
 c13-2x3-api-c848b775d-qtrs7{"requests":{"cpu":"50m","memory":"20Mi"}}
 c13-2x3-web-6989fb8dc6-4nc9z{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-2x3-web-6989fb8dc6-7xfdx{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-2x3-web-6989fb8dc6-98pr6{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-2x3-web-6989fb8dc6-9zpkj{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-2x3-web-6989fb8dc6-j2mgb{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-2x3-web-6989fb8dc6-jcwk9{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-3cc-data-96d47bf85-dc8d4{"requests":{"cpu":"30m","memory":"10Mi"}}
 c13-3cc-data-96d47bf85-f9gd2{"requests":{"cpu":"30m","memory":"10Mi"}}
 c13-3cc-data-96d47bf85-fd9lc{"requests":{"cpu":"30m","memory":"10Mi"}}
 c13-3cc-runner-heavy-8687d66dbb-gnxjh{}
 c13-3cc-runner-heavy-8687d66dbb-przdh{}
 c13-3cc-runner-heavy-8687d66dbb-wqwfz{}
 c13-3cc-web-767b98dd48-5b45q{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-3cc-web-767b98dd48-5vldf{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-3cc-web-767b98dd48-dd7mc{"requests":{"cpu":"50m","memory":"10Mi"}}
 c13-3cc-web-767b98dd48-pb67p{"requests":{"cpu":"50m","memory":"10Mi"}}
```

This lists all *Pod* names and their requests/limits, hence we see the three *Pods* without those defined.

Or we look for the Quality of Service classes:

```
➜ candidate@cka2556:~$ k get pods -n project-c13 -o jsonpath="{range .items[*]}{.metadata.name}
{.status.qosClass}{'\n'}"
 c13-2x3-api-c848b775d-7nggw Burstable
 c13-2x3-api-c848b775d-qrrlp Burstable
 c13-2x3-api-c848b775d-qtrs7 Burstable
 c13-2x3-web-6989fb8dc6-4nc9z Burstable
 c13-2x3-web-6989fb8dc6-7xfdx Burstable
 c13-2x3-web-6989fb8dc6-98pr6 Burstable
 c13-2x3-web-6989fb8dc6-9zpkj Burstable
 c13-2x3-web-6989fb8dc6-j2mgb Burstable
 c13-2x3-web-6989fb8dc6-jcwk9 Burstable
 c13-3cc-data-96d47bf85-dc8d4 Burstable
 c13-3cc-data-96d47bf85-f9gd2 Burstable
 c13-3cc-data-96d47bf85-fd9lc Burstable
 c13-3cc-runner-heavy-8687d66dbb-gnxjh BestEffort
 c13-3cc-runner-heavy-8687d66dbb-przdh BestEffort
 c13-3cc-runner-heavy-8687d66dbb-wqwfz BestEffort
 c13-3cc-web-767b98dd48-5b45q Burstable
 c13-3cc-web-767b98dd48-5vldf Burstable
 c13-3cc-web-767b98dd48-dd7mc Burstable
 c13-3cc-web-767b98dd48-pb67p Burstable
```

Here we see three with BestEffort, which *Pods* get that don't have any memory or cpu limits or requests defined.

A good practice is to always set resource requests and limits. If you don't know the values your containers should have you can find this out using metric tools like Prometheus. You can also use `kubectl top pod` or even `kubectl exec` into the container and use `top` and similar tools.

Question 5 | Kustomize configure HPA Autoscaler

Solve this question on: `ssh cka5774`

Previously the application `api-gateway` used some external autoscaler which should now be replaced with a *HorizontalPodAutoscaler* (*HPA*). The application has been deployed to *Namespaces* `api-gateway-staging` and `api-gateway-prod` like this:

```
kubectl kustomize /opt/course/5/api-gateway/staging | kubectl apply -f -
kubectl kustomize /opt/course/5/api-gateway/prod | kubectl apply -f -
```

Using the Kustomize config at `/opt/course/5/api-gateway` do the following:

1. Remove the *ConfigMap* `horizontal-scaling-config` completely
2. Add *HPA* named `api-gateway` for the *Deployment* `api-gateway` with min `2` and max `4` replicas. It should scale at `50%` average CPU utilisation
3. In prod the *HPA* should have max `6` replicas
4. Apply your changes for staging and prod so they're reflected in the cluster

Answer

Kustomize is a standalone tool to manage K8s Yaml files, but it also comes included with kubectl. The common idea is to have a base set of K8s Yaml and then override or extend it for different overlays, like here done for staging and prod:

```
➜  ssh cka5774

➜  candidate@cka5774:~$ cd /opt/course/5/api-gateway

➜  candidate@cka5774:/opt/course/5/api-gateway$ ls
base  prod  staging
```

Investigate Base

Let's investigate the base first for better understanding:

```
➜  candidate@cka5774:/opt/course/5/api-gateway$ k kustomize base
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
  namespace: NAMESPACE_REPLACE
```

```
---
apiVersion: v1
data:
  horizontal-scaling: "70"
kind: ConfigMap
metadata:
  name: horizontal-scaling-config
  namespace: NAMESPACE_REPLACE
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: NAMESPACE_REPLACE
spec:
  replicas: 1
  selector:
    matchLabels:
      id: api-gateway
  template:
    metadata:
      labels:
        id: api-gateway
    spec:
      containers:
      - image: httpd:2-alpine
        name: httpd
      serviceAccountName: api-gateway
```

Running `kubectl kustomize DIR` will build the whole Yaml based on whatever is defined in the `kustomization.yaml`.

In the case above we did build for the base directory, which produces Yaml that is not expected to be deployed just like that. We can see for example that all resources contain `namespace: NAMESPACE_REPLACE` entries which won't be possible to apply because *Namespace* names need to be lowercase.

But for debugging it can be useful to build the base Yaml.

Investigate Staging

Now we look at the staging directory:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
  namespace: api-gateway-staging
---
apiVersion: v1
data:
  horizontal-scaling: "60"
kind: ConfigMap
metadata:
  name: horizontal-scaling-config
  namespace: api-gateway-staging
---
apiVersion: apps/v1
kind: Deployment
```

```
  metadata:
    labels:
      env: staging
    name: api-gateway
    namespace: api-gateway-staging
  spec:
    replicas: 1
    selector:
      matchLabels:
        id: api-gateway
    template:
      metadata:
        labels:
          id: api-gateway
      spec:
        containers:
        - image: httpd:2-alpine
          name: httpd
        serviceAccountName: api-gateway
```

We can see that all resources now have `namespace: api-gateway-staging`. Also staging seems to change the *ConfigMap* value to `horizontal-scaling: "60"`. And it adds the additional label `env: staging` to the *Deployment*. The rest is taken from base.

This all happens because of the `kustomization.yaml`:

```
# cka5774:/opt/course/5/api-gateway/staging/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - ../base

patches:
  - path: api-gateway.yaml

transformers:
  - |-
    apiVersion: builtin
    kind: NamespaceTransformer
    metadata:
      name: notImportantHere
      namespace: api-gateway-staging
```

- The `resources:` section is the directory on which everything will be based on

- The `patches:` section specifies Yaml files with alterations or additions applied on the base files

- The `transformers:` section in this case sets the *Namespace* for all resources

We should be able to build and deploy the staging Yaml:

```
➡ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | kubectl diff -f -


➡ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | kubectl apply -f -
serviceaccount/api-gateway unchanged
configmap/horizontal-scaling-config unchanged
deployment.apps/api-gateway unchanged
```

Actually we see that no changes were performed, because everything is already deployed:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-staging get deploy,cm
NAME                          READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/api-gateway   1/1      1             1            20m


NAME                                  DATA    AGE
configmap/horizontal-scaling-config   1       20m
configmap/kube-root-ca.crt            1       21m
```

Investigate Prod

Everything said about staging is also true about prod, there are just different values of resources changed. Hence we should also see that there are no changes to be applied:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
  namespace: api-gateway-prod
...
```

We can see that now *Namespace* `api-gateway-prod` is being used.

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | kubectl diff -f -

➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | kubectl apply -f -
serviceaccount/api-gateway unchanged
configmap/horizontal-scaling-config unchanged
deployment.apps/api-gateway unchanged
```

And everything seems to be up to date for prod as well.

Step 1

We need to remove the *ConfigMap* from base, staging and prod because staging and prod both reference it as a patch. If we would only remove it from base we would run into an error when trying to build staging for example:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging
error: no resource matches strategic merge patch "ConfigMap.v1.[noGrp]/horizontal-scaling-config.[noNs]": no
matches for Id ConfigMap.v1.[noGrp]/horizontal-scaling-config.[noNs]; failed to find unique target for patch
ConfigMap.v1.[noGrp]/horizontal-scaling-config.[noNs]
```

So we edit files `base/api-gateway.yaml`, `staging/api-gateway.yaml` and `prod/api-gateway.yaml` and remove the *ConfigMap*. Afterwards we should get no errors and Yaml without that *ConfigMap*:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
  namespace: api-gateway-staging
---
apiVersion: apps/v1
kind: Deployment
```

```
  metadata:
    labels:
      env: staging
    name: api-gateway
    namespace: api-gateway-staging
  spec:
    replicas: 1
    selector:
      matchLabels:
        id: api-gateway
    template:
      metadata:
        labels:
          id: api-gateway
      spec:
        containers:
        - image: httpd:2-alpine
          name: httpd
        serviceAccountName: api-gateway

➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
  namespace: api-gateway-prod
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    env: prod
  name: api-gateway
  namespace: api-gateway-prod
spec:
  replicas: 1
  selector:
    matchLabels:
      id: api-gateway
  template:
    metadata:
      labels:
        id: api-gateway
    spec:
      containers:
      - image: httpd:2-alpine
        name: httpd
      serviceAccountName: api-gateway
```

Step 2

We're going to add the requested *HPA* into the base config file:

```
# cka5774:/opt/course/5/api-gateway/base/api-gateway.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
```

```yaml
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api-gateway
  minReplicas: 2
  maxReplicas: 4
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: api-gateway
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
spec:
  replicas: 1
  selector:
    matchLabels:
      id: api-gateway
  template:
    metadata:
      labels:
        id: api-gateway
    spec:
      serviceAccountName: api-gateway
      containers:
        - image: httpd:2-alpine
          name: httpd
```

Notice that we don't specify a *Namespace* here as done also for the other resources. The *Namespace* will be set by staging and prod overlays automatically.

Step 3

In prod the *HPA* should have max replicas set to 6 so we add this to the prod patch:

```yaml
# cka5774:/opt/course/5/api-gateway/prod/api-gateway.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
spec:
  maxReplicas: 6
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  labels:
    env: prod
```

With that change we should see that staging will have the *HPA* with `maxReplicas: 4` from base, whereas prod will have `maxReplicas: 6`:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | grep maxReplicas -B5
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
  namespace: api-gateway-staging
spec:
  maxReplicas: 4

➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | grep maxReplicas -B5
kind: HorizontalPodAutoscaler
metadata:
  name: api-gateway
  namespace: api-gateway-prod
spec:
  maxReplicas: 6
```

Step 4

Finally we apply the changes, first staging:

```
➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | kubectl diff -f -
diff -u -N /tmp/LIVE-3038173353/autoscaling.v2.HorizontalPodAutoscaler.api-gateway-staging.api-gateway
/tmp/MERGED-332240272/autoscaling.v2.HorizontalPodAutoscaler.api-gateway-staging.api-gateway
--- /tmp/LIVE-3038173353/autoscaling.v2.HorizontalPodAutoscaler.api-gateway-staging.api-gateway 2024-12-23
16:21:47.771211074 +0000
+++ /tmp/MERGED-332240272/autoscaling.v2.HorizontalPodAutoscaler.api-gateway-staging.api-gateway
2024-12-23 16:21:47.772211169 +0000
@@ -0,0 +1,24 @@
+apiVersion: autoscaling/v2
+kind: HorizontalPodAutoscaler
+metadata:
+  creationTimestamp: "2024-12-23T16:21:47Z"
+  name: api-gateway
+  namespace: api-gateway-staging
+  uid: d846f349-e695-4538-b3f8-ba514fc02ea5
+spec:
+  maxReplicas: 4
+  metrics:
+  - resource:
+      name: cpu
+      target:
+        averageUtilization: 50
+        type: Utilization
+    type: Resource
+  minReplicas: 2
+  scaleTargetRef:
+    apiVersion: apps/v1
+    kind: Deployment
+    name: api-gateway
+status:
+  currentMetrics: null
+  desiredReplicas: 0

➜ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | kubectl apply -f -
```

```
servceaccount/api-gateway unchanged
deployment.apps/api-gateway unchanged
horizontalpodautoscaler.autoscaling/api-gateway created


→ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize staging | kubectl diff -f -
```

And next for prod:

```
→ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | kubectl apply -f -
servceaccount/api-gateway unchanged
deployment.apps/api-gateway unchanged
horizontalpodautoscaler.autoscaling/api-gateway created


→ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | kubectl diff -f -
```

We notice that the *HPA* was created as expected, but nothing was done with the *ConfigMap* that we removed from the Yaml files earlier. We need to delete the remote *ConfigMaps* manually, why is explained in more detail at the end of this solution.

```
→ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-staging get cm
NAME                         DATA     AGE
horizontal-scaling-config    1        61m
kube-root-ca.crt             1        61m

→ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-staging delete cm horizontal-scaling-config
configmap "horizontal-scaling-config" deleted

→ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-prod get cm
NAME                         DATA     AGE
horizontal-scaling-config    2        61m
kube-root-ca.crt             1        62m

→ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-prod delete cm horizontal-scaling-config
configmap "horizontal-scaling-config" deleted
candidate@cka5774:/opt/course/5/api-gateway$
```

Done!

Diff output after solution complete

After deleting the *ConfigMaps* manually we should not see any changes when running a diff. This is because the *ConfigMap* does not exist any longer in our Yaml and we already applied all changes. But we might see something like this:

```
→ candidate@cka5774:/opt/course/5/api-gateway$ k kustomize prod | kubectl diff -f -
diff -u -N /tmp/LIVE-849078037/apps.v1.Deployment.api-gateway-prod.api-gateway /tmp/MERGED-
   2513424623/apps.v1.Deployment.api-gateway-prod.api-gateway
--- /tmp/LIVE-849078037/apps.v1.Deployment.api-gateway-prod.api-gateway 2024-12-23 16:37:44.763088538 +0000
+++ /tmp/MERGED-2513424623/apps.v1.Deployment.api-gateway-prod.api-gateway      2024-12-23
   16:37:44.766088823 +0000
@@ -6,7 +6,7 @@
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":
   {"env":"prod"},"name":"api-gateway","namespace":"api-gateway-prod"},"spec":{"replicas":1,"selector":
   {"matchLabels":{"id":"api-gateway"}},"template":{"metadata":{"labels":{"id":"api-gateway"}},"spec":
   {"containers":[{"image":"httpd:2-alpine","name":"httpd"}],"serviceAccountName":"api-gateway"}}}}
      creationTimestamp: "2024-12-23T15:34:06Z"
-   generation: 2
```

```
 +   generation: 3
     labels:
       env: prod
     name: api-gateway
@@ -15,7 +15,7 @@
     uid: ca3b43c9-d33b-4bdc-98ae-172cd9ee8cdb
   spec:
     progressDeadlineSeconds: 600
 -   replicas: 2
 +   replicas: 1
     revisionHistoryLimit: 10
     selector:
       matchLabels:
```

Above we can see that we would change the replicas from 2 to 1. This is because the *HPA* already set the replicas to the `minReplicas` that we defined and it's different than the default `replicas:` of the *Deployment*:

```
→ candidate@cka5774:/opt/course/5/api-gateway$ k -n api-gateway-prod get hpa
NAME            ...    MINPODS    MAXPODS    REPLICAS    AGE
api-gateway    ...    2          6          2           15m
```

This means each time we deploy our Kustomize built Yaml, the replicas that the *HPA* applied would be overwritten, which is not cool. It does not matter for the scoring of this question but to prevent this we could simply remove the `replicas:` setting from the *Deployment* in base, staging and prod.

Kustomize / Helm and State

We had to delete the remote *ConfigMaps* manually. Kustomize won't delete remote resources if they only exist remote. This is because it does not keep any state and hence doesn't know which remote resources were created by Kustomize or by anything else.

Helm will remove remote resources if they only exist remote and if they were created by Helm. It can do this because it keeps a state of all performed changes.

Both approaches have pros and cons:

- Kustomize is less complex by not having to manage state, but might need more manual work cleaning up

- Helm can keep better track of remote resources, but things can get complex and messy if there is a state error or mismatch. State changes (Helm actions) at the same time need to be prevented or accounted for.

Question 6 | Storage, PV, PVC, Pod volume

Solve this question on: `ssh cka7968`

Create a new *PersistentVolume* named `safari-pv`. It should have a capacity of *2Gi*, accessMode *ReadWriteOnce*, hostPath `/Volumes/Data` and no storageClassName defined.

Next create a new *PersistentVolumeClaim* in *Namespace* `project-t230` named `safari-pvc`. It should request *2Gi* storage, accessMode *ReadWriteOnce* and should not define a storageClassName. The *PVC* should bound to the *PV* correctly.

Finally create a new *Deployment* `safari` in *Namespace* `project-t230` which mounts that volume at `/tmp/safari-data`. The *Pods* of that *Deployment* should be of image `httpd:2-alpine`.

Answer

```
→ ssh cka7968
```

```
→ candidate@cka7968:~$ vim 6_pv.yaml
```

Find an example from https://kubernetes.io/docs and alter it:

```
# cka7968:/home/candidate/6_pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
 name: safari-pv
spec:
 capacity:
   storage: 2Gi
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: "/Volumes/Data"
```

> ℹ Using the hostPath volume type presents many security risks, avoid if possible. Be aware that data stored in the hostPath directory will not be shared across nodes. The data available for a *Pod* depends on which node the *Pod* is scheduled.

Then create it:

```
→ candidate@cka7968:~$ k -f 6_pv.yaml create
persistentvolume/safari-pv created
```

Next the *PersistentVolumeClaim*:

```
→ candidate@cka7968:~ vim 6_pvc.yaml
```

Find an example from the K8s Docs and alter it:

```
# cka7968:/home/candidate/6_pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: safari-pvc
  namespace: project-t230
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

Then create:

```
→ candidate@cka7968:~$ k -f 6_pvc.yaml create
persistentvolumeclaim/safari-pvc created
```

And check that both have the status Bound:

```
➜ candidate@cka7968:~$ k -n project-t230 get pv,pvc
NAME                        CAPACITY  ... STATUS  CLAIM                       ...
persistentvolume/safari-pv  2Gi       ... Bound   project-t230/safari-pvc ...

NAME                              STATUS  VOLUME       CAPACITY ...
persistentvolumeclaim/safari-pvc  Bound   safari-pv    2Gi         ...
```

Next we create a *Deployment* and mount that volume:

```
➜ candidate@cka7968:~$ k -n project-t230 create deploy safari --image=httpd:2-alpine --dry-run=client -o
yaml > 6_dep.yaml

➜ candidate@cka7968:~$ vim 6_dep.yaml
```

Alter the yaml to mount the volume:

```
# cka7968:/home/candidate/6_dep.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: safari
  name: safari
  namespace: project-t230
spec:
  replicas: 1
  selector:
    matchLabels:
      app: safari
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: safari
    spec:
      volumes:                          # add
      - name: data                      # add
        persistentVolumeClaim:          # add
          claimName: safari-pvc         # add
      containers:
      - image: httpd:2-alpine
        name: container
        volumeMounts:                   # add
        - name: data                    # add
          mountPath: /tmp/safari-data   # add
```

```
➜ candidate@cka7968:~$ k -f 6_dep.yaml create
deployment.apps/safari created
```

We can confirm it's mounting correctly:

```
➜ candidate@cka7968:~$ k -n project-t230 describe pod safari-b499cc5b9-x7d7h | grep -A2 Mounts:
    Mounts:
      /tmp/safari-data from data (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-xght8 (ro)
```

Question 7 | Node and Pod Resource Usage

Solve this question on: `ssh cka5774`

The metrics-server has been installed in the cluster. Write two bash scripts which use `kubectl`:

1. Script `/opt/course/7/node.sh` should show resource usage of *Nodes*

2. Script `/opt/course/7/pod.sh` should show resource usage of *Pods* and their containers

Answer:

The command we need to use here is top:

```
➜ ssh cka5774

➜ candidate@cka5774:~$ k top -h
Display resource (CPU/memory) usage.

 The top command allows you to see the resource consumption for nodes or pods.

 This command requires Metrics Server to be correctly configured and working on the server.

Available Commands:
  node          Display resource (CPU/memory) usage of nodes
  pod           Display resource (CPU/memory) usage of pods

Usage:
  kubectl top [flags] [options]

Use "kubectl top <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all commands).
```

We see that the metrics server provides information about resource usage:

```
➜ candidate@cka5774:~$ k top node
NAME           CPU(cores)    CPU%    MEMORY(bytes)    MEMORY%
cka5774        104m          10%     1121Mi           60%
```

We create the first file, ensure to **not** use aliases but instead the full command names:

```
# cka5774:/opt/course/7/node.sh
kubectl top node
```

For the second file we might need to check the docs again:

```
➜  candidate@cka5774:~$ k top pod -h
Display resource (CPU/memory) usage of pods.
...
    --containers=false:
        If present, print usage of containers within a pod.
...
```

With this we can finish this task:

```
# cka5774:/opt/course/7/pod.sh
kubectl top pod --containers=true
```

Question 8 | Update Kubernetes Version and join cluster

Solve this question on: `ssh cka3962`

Your coworker notified you that node `cka3962-node1` is running an older Kubernetes version and is not even part of the cluster yet.

1. Update the node's Kubernetes to the exact version of the controlplane
2. Add the node to the cluster using kubeadm

> **i** You can connect to the worker node using `ssh cka3962-node1` from `cka3962`

Answer:

Update Kubernetes to controlplane version

Search in the docs for kubeadm upgrade:

```
➜  ssh cka3962

➜  candidate@cka3962:~$ k get node
NAME      STATUS    ROLES           AGE     VERSION
cka3962   Ready     control-plane   3h4m    v1.33.1
```

The controlplane node seems to be running Kubernetes 1.33.1.

```
➜  candidate@cka3962:~$ ssh cka3962-node1

➜  candidate@cka3962-node1:~$ sudo -i

➜  root@cka3962-node1:~# kubectl version
Client Version: v1.32.5
Kustomize Version: v5.5.0
```

```
The connection to the server localhost:8080 was refused - did you specify the right host or port?

➜  root@cka3962-node1:~# kubelet --version
Kubernetes v1.32.5

➜  root@cka3962-node1:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"33", EmulationMajor:"", EmulationMinor:"",
MinCompatibilityMajor:"", MinCompatibilityMinor:"", GitVersion:"v1.33.1",
GitCommit:"8adc0f041b8e7ad1d30e29cc59c6ae7a15e19828", GitTreeState:"clean", BuildDate:"2025-05-
15T08:25:51Z", GoVersion:"go1.24.2", Compiler:"gc", Platform:"linux/amd64"}
```

Above we can see that kubeadm is already installed in the exact needed version, otherwise we would need to install it using `apt install kubeadm=1.33.1-1.1`.

With the correct kubeadm version we can continue:

```
➜  root@cka3962-node1:~# kubeadm upgrade node
couldn't create a Kubernetes client from file "/etc/kubernetes/kubelet.conf": failed to load admin
kubeconfig: open /etc/kubernetes/kubelet.conf: no such file or directory
To see the stack trace of this error execute with --v=5 or higher
```

This is usually the proper command to upgrade a worker node. But as mentioned in the question description, this node is not yet part of the cluster. Hence there is nothing to update. We'll add the node to the cluster later using `kubeadm join`. For now we can continue with updating kubelet and kubectl:

```
➜  root@cka3962-node1:~# apt update
Hit:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.32/deb  InRelease
Hit:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.33/deb  InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.

➜  root@cka3962-node1:~# apt show kubectl -a | grep 1.33

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Version: 1.33.2-1.1
APT-Sources: https://pkgs.k8s.io/core:/stable:/v1.33/deb  Packages
Version: 1.33.1-1.1
APT-Sources: https://pkgs.k8s.io/core:/stable:/v1.33/deb  Packages
Version: 1.33.0-1.1
APT-Sources: https://pkgs.k8s.io/core:/stable:/v1.33/deb  Packages

➜  root@cka3962-node1:~# apt install kubectl=1.33.1-1.1 kubelet=1.33.1-1.1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libevent-core-2.1-7t64 libjq1 libonig5 pastebinit python3-newt run-one
  squashfs-tools
Use 'apt autoremove' to remove them.
The following packages will be upgraded:
  kubectl kubelet
2 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 27.5 MB of archives.
After this operation, 7074 kB of additional disk space will be used.
```

```
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.33/deb   kubectl
1.33.1-1.1 [11.7 MB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.33/deb   kubelet
1.33.1-1.1 [15.9 MB]
Fetched 27.5 MB in 1s (40.1 MB/s)
(Reading database ... 72153 files and directories currently installed.)
Preparing to unpack .../kubectl_1.33.1-1.1_amd64.deb ...
Unpacking kubectl (1.33.1-1.1) over (1.32.5-1.1) ...
Preparing to unpack .../kubelet_1.33.1-1.1_amd64.deb ...
Unpacking kubelet (1.33.1-1.1) over (1.32.5-1.1) ...
Setting up kubectl (1.33.1-1.1) ...
Setting up kubelet (1.33.1-1.1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.


➜ root@cka3962-node1:~# kubelet --version
Kubernetes v1.33.1
```

Now that we're up to date with kubeadm, kubectl and kubelet we can restart the kubelet:

```
➜ root@cka3962-node1:~# service kubelet restart

➜ root@cka3962-node1:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; pre>
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: activating (auto-restart) (Result: exit-code) since Thu 2025->
       Docs: https://kubernetes.io/docs/
    Process: 13058 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KU>
   Main PID: 13058 (code=exited, status=1/FAILURE)
        CPU: 83ms
lines 1-9/9 (END)
```

These errors occur because we still need to run `kubeadm join` to join the node into the cluster. Let's do this in the next step.

Add cka3962-node1 to cluster

First we log into the controlplane node and generate a new TLS bootstrap token, also printing out the join command:

```
→ ssh cka3962


→ candidate@cka3962:~$ sudo -i


→ root@cka3962:~# kubeadm token create --print-join-command
kubeadm join 192.168.100.31:6443 --token u1n13d.x68l08ofgamv754u --discovery-token-ca-cert-hash
sha256:8945ece6fdf32ae6ee85483acd7ade2e1ce878615bc8972cd5e86031e3d99325


→ root@cka3962:~# kubeadm token list
TOKEN                    TTL          ...
2tzjdq.83vn35llq1lch9b9  20h          ...
u1n13d.x68l08ofgamv754u  23h          ...
xylrd3.d3x7g0aajrvf0lqn  <forever>    ...
```

We see the expiration of 23h for our token, we could adjust this by passing the ttl argument.

Next we connect again to `cka3962-node1` and simply execute the join command from above:

```
→ root@cka3962:~# ssh cka3962-node1


→ root@cka3962-node1:~# kubeadm join 192.168.100.31:6443 --token u1n13d.x68l08ofgamv754u --discovery-token-
ca-cert-hash sha256:8945ece6fdf32ae6ee85483acd7ade2e1ce878615bc8972cd5e86031e3d99325
[preflight] Running pre-flight checks
[preflight] Reading configuration from the "kubeadm-config" ConfigMap in namespace "kube-system"...
[preflight] Use 'kubeadm init phase upload-config --config your-config-file' to re-upload it.
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.502892103s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap


This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.


Run 'kubectl get nodes' on the control-plane to see this node join the cluster.


→ root@cka3962-node1:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; pre>
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Thu 2025-06-19 12:10:37 UTC; 14s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 13316 (kubelet)
      Tasks: 10 (limit: 1113)
     Memory: 23.9M (peak: 24.1M)
        CPU: 1.914s
     CGroup: /system.slice/kubelet.service
             └─13316 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernet>
...
```

ℹ️ If you have troubles with `kubeadm join` you might need to run `kubeadm reset` before

Finally we check the node status:

```
➜ root@cka3962:~# k get node
NAME            STATUS      ROLES           AGE     VERSION
cka3962         Ready       control-plane   3h11m   v1.33.1
cka3962-node1   NotReady    <none>          20s     v1.33.1
```

Give it a bit of time till the node is ready.

```
➜ root@cka3962:~# k get node
NAME            STATUS      ROLES           AGE     VERSION
cka3962         Ready       control-plane   3h11m   v1.33.1
cka3962-node1   Ready       <none>          38s     v1.33.1
```

We see `cka3962-node1` is now available and up to date.

Question 9 | Contact K8s Api from inside Pod

Solve this question on: `ssh cka9412`

There is *ServiceAccount* `secret-reader` in *Namespace* `project-swan`. Create a *Pod* of image `nginx:1-alpine` named `api-contact` which uses this *ServiceAccount*.

Exec into the *Pod* and use `curl` to manually query all *Secrets* from the Kubernetes Api.

Write the result into file `/opt/course/9/result.json`.

Answer:

https://kubernetes.io/docs/tasks/run-application/access-api-from-pod

You can find information in the K8s Docs by searching for "curl api" for example.

Create Pod which uses ServiceAccount

First we create the *Pod*:

```
➜ ssh cka9412

➜ candidate@cka9412:~$ k run api-contact --image=nginx:1-alpine --dry-run=client -o yaml > 9.yaml

➜ candidate@cka9412:~$ vim 9.yaml
```

Add the serviceAccountName and *Namespace*:

```
# cka9412:/home/candidate/9.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: api-contact
```

```
    name: api-contact
    namespace: project-swan          # add
  spec:
    serviceAccountName: secret-reader   # add
    containers:
    - image: nginx:1-alpine
      name: api-contact
      resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Always
  status: {}
```

Create it:

```
→ candidate@cka9412:~$ k -f 9.yaml apply
pod/api-contact created
```

Contact K8s Api from inside Pod

Once in the container we can connect to the K8s Api using `curl`, it's usually available via the *Service* named `kubernetes` in *Namespace* `default`. Because of K8s internal DNS resolution we can use the url `kubernetes.default`.

> ℹ️ Otherwise we can find the K8s Api IP via environment variables inside the *Pod*, simply run `env`

So we can try to contact the K8s Api:

```
→ candidate@cka9412:~$ k -n project-swan exec api-contact -it -- sh

→ / # curl https://kubernetes.default
curl: (60) SSL peer certificate or SSH remote key was not OK
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the webpage mentioned above.

→ / # curl -k https://kubernetes.default
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {},
  "code": 403
}~ $

→ / # curl -k https://kubernetes.default/api/v1/secrets
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "secrets is forbidden: User \"system:anonymous\" cannot list resource \"secrets\" in API group
\"\" at the cluster scope",
  "reason": "Forbidden",
```

```
      "details": {
        "kind": "secrets"
      },
      "code": 403
    }
```

The first command fails because of an untrusted certificate, but we can ignore this with `-k` for this scenario. We explain at the end how we can add the correct certificate instead of having to use the insecure `-k` option.

The last command shows 403 forbidden, this is because we are not passing any authorisation information. For the K8s Api we are connecting as `system:anonymous`, which should not have permission to perform the query. We want to change this and connect using the *Pod's ServiceAccount* named `secret-reader`.

Use ServiceAccount Token to authenticate

We find the token at `/var/run/secrets/kubernetes.io/serviceaccount`, so we do:

```
➜ / # TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)

➜ / # curl -k https://kubernetes.default/api/v1/secrets -H "Authorization: Bearer ${TOKEN}"
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "4881"
  },
  "items": [
    {
...
    {
      "metadata": {
        "name": "read-me",
        "namespace": "project-swan",
...
```

Now we're able to list all *Secrets* as the *Pod's ServiceAccount* `secret-reader`.

For troubleshooting we could also check if the *ServiceAccount* is actually able to list *Secrets*:

```
➜ candidate@cka9412:~$ k auth can-i get secret --as system:serviceaccount:project-swan:secret-reader
yes
```

Store result at requested location

We write the full result into `/opt/course/9/result.json`:

```
# cka9412:/opt/course/9/result.json
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "4881"
  },
  "items": [
    {
```

```
  ...
    {
      "metadata": {
        "name": "read-me",
        "namespace": "project-swan",
        "uid": "f7c9a279-9609-4f9a-aa30-d29e175b7a6c",
        "resourceVersion": "3380",
        "creationTimestamp": "2024-12-05T15:11:58Z",
        "managedFields": [
          {
            "manager": "kubectl-create",
            "operation": "Update",
            "apiVersion": "v1",
            "time": "2024-12-05T15:11:58Z",
            "fieldsType": "FieldsV1",
            "fieldsV1": {
              "f:data": {
                ".": {},
                "f:token": {}
              },
              "f:type": {}
            }
          }
        ]
      },
      "data": {
        "token": "ZjMyMDEzOTYtZjVkOC00NTg0LWE2ZjEtNmYyZGZkYjM4NzVl"
      },
      "type": "Opaque"
    }
  ]
}
  ...
```

The easiest way would probably be to copy and paste the result manually. But if it's too long or not possible we could also do:

```
→ / # curl -k https://kubernetes.default/api/v1/secrets -H "Authorization: Bearer ${TOKEN}" > result.json

→ / # exit

→ candidate@cka9412:~$ k -n project-swan exec api-contact -it -- cat result.json > /opt/course/9/result.json
```

Connect via HTTPS with correct CA

To connect without `curl -k` we can specify the CertificateAuthority (CA):

```
→ / # CACERT=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt

→ / # curl --cacert ${CACERT} https://kubernetes.default/api/v1/secrets -H "Authorization: Bearer ${TOKEN}"
```

Question 10 | RBAC ServiceAccount Role RoleBinding

Solve this question on: `ssh cka3962`

Create a new *ServiceAccount* `processor` in *Namespace* `project-hamster`. Create a *Role* and *RoleBinding*, both named `processor` as well. These should allow the new *SA* to only create *Secrets* and *ConfigMaps* in that *Namespace*.

Answer:

Let's talk a little about RBAC resources

A *ClusterRole|Role* defines a set of permissions and **where it is available**, in the whole cluster or just a single *Namespace*.

A *ClusterRoleBinding|RoleBinding* connects a set of permissions with an account and defines **where it is applied**, in the whole cluster or just a single *Namespace*.

Because of this there are 4 different RBAC combinations and 3 valid ones:

1. *Role + RoleBinding* (available in single *Namespace*, applied in single *Namespace*)
2. *ClusterRole + ClusterRoleBinding* (available cluster-wide, applied cluster-wide)
3. *ClusterRole + RoleBinding* (available cluster-wide, applied in single *Namespace*)
4. *Role + ClusterRoleBinding* (**NOT POSSIBLE:** available in single *Namespace*, applied cluster-wide)

To the solution

We first create the *ServiceAccount*:

```
➜  ssh cka3962

➜  candidate@cka3962:~$ k -n project-hamster create sa processor
serviceaccount/processor created
```

For the *Role* we can first view examples:

```
k -n project-hamster create role -h
```

So we execute:

```
➜  candidate@cka3962:~$ k -n project-hamster create role processor --verb=create --resource=secret --resource=configmap
role.rbac.authorization.k8s.io/processor created
```

Which will create a *Role* like:

```
# kubectl -n project-hamster create role processor --verb=create --resource=secret --resource=configmap
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: processor
  namespace: project-hamster
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  - configmaps
  verbs:
  - create
```

Now we bind the *Role* to the *ServiceAccount*, and for this we can also view examples:

```
k -n project-hamster create rolebinding -h # examples
```

So we create it:

```
→ candidate@cka3962:~$ k -n project-hamster create rolebinding processor --role processor --serviceaccount
project-hamster:processor
rolebinding.rbac.authorization.k8s.io/processor created
```

This will create a *RoleBinding* like:

```
# kubectl -n project-hamster create rolebinding processor --role processor --serviceaccount project-
hamster:processor
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: processor
  namespace: project-hamster
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: processor
subjects:
- kind: ServiceAccount
  name: processor
  namespace: project-hamster
```

To test our RBAC setup we can use `kubectl auth can-i`:

```
k auth can-i -h # examples
```

Like this:

```
→ candidate@cka3962:~$ k -n project-hamster auth can-i create secret --as system:serviceaccount:project-
hamster:processor
yes

→ candidate@cka3962:~$ k -n project-hamster auth can-i create configmap --as system:serviceaccount:project-
hamster:processor
yes

→ candidate@cka3962:~$ k -n project-hamster auth can-i create pod --as system:serviceaccount:project-
hamster:processor
no

→ candidate@cka3962:~$ k -n project-hamster auth can-i delete secret --as system:serviceaccount:project-
hamster:processor
no

→ candidate@cka3962:~$ k -n project-hamster auth can-i get configmap --as system:serviceaccount:project-
hamster:processor
no
```

Done.

Question 11 | DaemonSet on all Nodes

Solve this question on: `ssh cka2556`

Use *Namespace* `project-tiger` for the following. Create a *DaemonSet* named `ds-important` with image `httpd:2-alpine` and labels `id=ds-important` and `uuid=18426a0b-5f59-4e10-923f-c0e078e82462`. The *Pods* it creates should request 10 millicore cpu and 10 mebibyte memory. The *Pods* of that *DaemonSet* should run on all nodes, also controlplanes.

Answer:

As of now we aren't able to create a *DaemonSet* directly using `kubectl`, so we create a *Deployment* and just change it up:

```
➜ ssh cka2556

➜ candidate@cka2556:~$ k -n project-tiger create deployment --image=httpd:2.4-alpine ds-important --dry-run=client -o yaml > 11.yaml
```

Or we could search for a *DaemonSet* example yaml in the K8s docs and alter it to our needs.

We adjust the yaml to:

```
# cka2556:/home/candidate/11.yaml
apiVersion: apps/v1
kind: DaemonSet                                    # change from Deployment to Daemonset
metadata:
  creationTimestamp: null
  labels:                                          # add
    id: ds-important                               # add
    uuid: 18426a0b-5f59-4e10-923f-c0e078e82462     # add
  name: ds-important
  namespace: project-tiger                         # important
spec:
  #replicas: 1                                      # remove
  selector:
    matchLabels:
      id: ds-important                             # add
      uuid: 18426a0b-5f59-4e10-923f-c0e078e82462   # add
  #strategy: {}                                     # remove
  template:
    metadata:
      creationTimestamp: null
      labels:
        id: ds-important                           # add
        uuid: 18426a0b-5f59-4e10-923f-c0e078e82462 # add
    spec:
      containers:
      - image: httpd:2-alpine
        name: ds-important
        resources:
          requests:                                # add
            cpu: 10m                               # add
            memory: 10Mi                           # add
      tolerations:                                 # add
      - effect: NoSchedule                         # add
        key: node-role.kubernetes.io/control-plane # add
  #status: {}                                       # remove
```

It was requested that the *DaemonSet* runs on all nodes, so we need to specify the toleration for this.

Let's give it a go:

```
➜  candidate@cka2556:~$ k -f 11.yaml create
daemonset.apps/ds-important created

➜  candidate@cka2556:~$ k -n project-tiger get ds
NAME            DESIRED    CURRENT    READY    UP-TO-DATE    AVAILABLE    NODE SELECTOR    AGE
ds-important    3          3          3        3             3            <none>           8s

➜  candidate@cka2556:~$ k -n project-tiger get pod -l id=ds-important -o wide
NAME                 READY    STATUS     ...    NODE              ...
ds-important-26456   1/1      Running    ...    cka2556-node2     ...
ds-important-wnt5p   1/1      Running    ...    cka2556           ...
ds-important-wrbjd   1/1      Running    ...    cka2556-node1     ...
```

Above we can see one *Pod* on each node, including the controlplane one.

Question 12 | Deployment on all Nodes

Solve this question on: `ssh cka2556`

Implement the following in *Namespace* `project-tiger`:

- Create a *Deployment* named `deploy-important` with `3` replicas

- The *Deployment* and its *Pods* should have label `id=very-important`

- First container named `container1` with image `nginx:1-alpine`

- Second container named `container2` with image `google/pause`

- There should only ever be **one** *Pod* of that *Deployment* running on **one** worker node, use `topologyKey: kubernetes.io/hostname` for this

> ℹ️ Because there are two worker nodes and the *Deployment* has three replicas the result should be that the third *Pod* won't be scheduled. In a way this scenario simulates the behaviour of a *DaemonSet*, but using a *Deployment* with a fixed number of replicas

Answer:

There are two possible ways, one using `podAntiAffinity` and one using `topologySpreadConstraint`.

PodAntiAffinity

The idea here is that we create a "Inter-pod anti-affinity" which allows us to say a *Pod* should only be scheduled on a node where another *Pod* of a specific label (here the same label) is not already running.

Let's begin by creating the *Deployment* template:

```
→ ssh cka2556

→ candidate@cka2556:~$ k -n project-tiger create deployment --image=nginx:1-alpine deploy-important --dry-
run=client -o yaml > 12.yaml
```

Then change the yaml to:

```
# cka2556:/home/candidate/12.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    id: very-important              # change
  name: deploy-important
  namespace: project-tiger          # important
spec:
  replicas: 3                       # change
  selector:
    matchLabels:
      id: very-important            # change
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        id: very-important          # change
    spec:
      containers:
      - image: nginx:1-alpine
        name: container1            # change
        resources: {}
      - image: google/pause         # add
        name: container2            # add
      affinity:                                     # add
        podAntiAffinity:                            # add
          requiredDuringSchedulingIgnoredDuringExecution:   # add
          - labelSelector:                          # add
              matchExpressions:                     # add
              - key: id                             # add
                operator: In                        # add
                values:                             # add
                - very-important                    # add
            topologyKey: kubernetes.io/hostname     # add
status: {}
```

Specify a topologyKey, which is a pre-populated Kubernetes label, you can find this by describing a node.

TopologySpreadConstraints

We can achieve the same with `topologySpreadConstraints`. Best to try out and play with both.

```
# cka2556:/home/candidate/12.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    id: very-important              # change
```

```yaml
  name: deploy-important
  namespace: project-tiger         # important
spec:
  replicas: 3                      # change
  selector:
    matchLabels:
      id: very-important           # change
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        id: very-important         # change
    spec:
      containers:
      - image: nginx:1-alpine
        name: container1           # change
        resources: {}
      - image: google/pause        # add
        name: container2           # add
      topologySpreadConstraints:            # add
      - maxSkew: 1                           # add
        topologyKey: kubernetes.io/hostname  # add
        whenUnsatisfiable: DoNotSchedule     # add
        labelSelector:                       # add
          matchLabels:                       # add
            id: very-important               # add
status: {}
```

Apply and Run

Let's run it:

```
➜  candidate@cka2556:~$ k -f 12.yaml create
deployment.apps/deploy-important created
```

Then we check the *Deployment* status where it shows 2/3 ready count:

```
➜  candidate@cka2556:~$ k -n project-tiger get deploy -l id=very-important
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deploy-important  2/3     3            2           19s
```

And running the following we see one *Pod* on each worker node and one not scheduled.

```
➜  candidate@cka2556:~$ k -n project-tiger get pod -o wide -l id=very-important
NAME                                READY   STATUS    ...   IP           NODE
deploy-important-78f98b75f9-5s6js   0/2     Pending   ...   <none>       <none>
deploy-important-78f98b75f9-657hx   2/2     Running   ...   10.44.0.33   cka2556-node1
deploy-important-78f98b75f9-9bz8q   2/2     Running   ...   10.36.0.20   cka2556-node2
```

If we kubectl describe the not scheduled *Pod* it will show us the reason `didn't match pod anti-affinity rules`:

```
Warning  FailedScheduling  119s (x2 over 2m1s)  default-scheduler  0/3 nodes are available: 1 node(s) had
untolerated taint {node-role.kubernetes.io/control-plane: }, 2 node(s) didn't match pod anti-affinity rules.
preemption: 0/3 nodes are available: 1 Preemption is not helpful for scheduling, 2 No preemption victims
found for incoming pod.
```

Or our topologySpreadConstraints reason `didn't match pod topology spread constraints`:

```
Warning  FailedScheduling  20s (x2 over 22s)  default-scheduler  0/3 nodes are available: 1 node(s) had
untolerated taint {node-role.kubernetes.io/control-plane: }, 2 node(s) didn't match pod topology spread
constraints. preemption: 0/3 nodes are available: 1 Preemption is not helpful for scheduling, 2 No
preemption victims found for incoming pod.
```

Question 13 | Gateway Api Ingress

Solve this question on: `ssh cka7968`

The team from Project r500 wants to replace their Ingress (networking.k8s.io) with a Gateway Api (gateway.networking.k8s.io) solution. The old Ingress is available at `/opt/course/13/ingress.yaml`.

Perform the following in *Namespace* `project-r500` and for the already existing *Gateway*:

1. Create a new *HTTPRoute* named `traffic-director` which replicates the routes from the old Ingress

2. Extend the new *HTTPRoute* with path `/auto` which redirects to mobile if the User-Agent is exactly `mobile` and to desktop otherwise

The existing *Gateway* is reachable at `http://r500.gateway:30080` which means your implementation should work for these commands:

```
curl r500.gateway:30080/desktop
curl r500.gateway:30080/mobile
curl r500.gateway:30080/auto -H "User-Agent: mobile"
curl r500.gateway:30080/auto
```

Answer:

Comparing for example the older *Ingress* (networking.k8s.io/v1) and newer *HTTPRoute* (gateway.networking.k8s.io/v1) *CRDs* then they look quite similar in what they offer. They have a different config structure but provide the same idea of functionality.

The magic of the Gateway Api comes more to shine because of further resources (*GRPCRoute*, *TCPRoute*) and the architecture which is designed to be more flexible and extendable. This will provide better integration into existing cloud infrastructure and providers like GCP or AWS will be able to develop their own Gateway Api implementations.

Investigate CRDs

It was mentioned that a *Gateway* already exists, let's verify this:

```
➜ ssh cka7968

➜ candidate@cka7968:~$ k get crd
NAME                                          CREATED AT
clientsettingspolicies.gateway.nginx.org      2024-12-28T13:11:21Z
gatewayclasses.gateway.networking.k8s.io      2024-12-28T13:11:21Z
gateways.gateway.networking.k8s.io            2024-12-28T13:11:21Z
grpcroutes.gateway.networking.k8s.io          2024-12-28T13:11:21Z
```

```
httproutes.gateway.networking.k8s.io              2024-12-28T13:11:22Z
nginxgateways.gateway.nginx.org                   2024-12-28T13:11:23Z
nginxproxies.gateway.nginx.org                    2024-12-28T13:11:23Z
observabilitypolicies.gateway.nginx.org           2024-12-28T13:11:23Z
referencegrants.gateway.networking.k8s.io         2024-12-28T13:11:23Z
snippetsfilters.gateway.nginx.org                 2024-12-28T13:11:23Z


➜ candidate@cka7968:~$ k get gateway -A
NAMESPACE       NAME    CLASS    ADDRESS    PROGRAMMED    AGE
project-r500    main    nginx               True          2m


➜ candidate@cka7968:~$ k get gatewayclass -A
NAME     CONTROLLER                                        ACCEPTED    AGE
nginx    gateway.nginx.org/nginx-gateway-controller        True        2m12s
```

We can see that various *CRDs* from gateway.networking.k8s.io are available. In this scenario we'll only work directly with *HTTPRoute* which we need to create. It will reference the existing *Gateway* `main` which references the existing *GatewayClass* `nginx` :

```
➜ candidate@cka7968:~$ k -n project-r500 get gateway main -oyaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
...
  name: main
  namespace: project-r500
spec:
  gatewayClassName: nginx
  listeners:
  - allowedRoutes:
      namespaces:
        from: Same
    name: http
    port: 80
    protocol: HTTP
...
```

Investigate URL reachability

We can already contact the *Gateway* like this:

```
➜ candidate@cka7968:~$ curl r500.gateway:30080
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

We receive a 404 because no routes have been defined yet. We receive this 404 from a Nginx because the Gateway Api implementation in this scenario has been done via the Nginx Gateway Fabric. But for this scenario it wouldn't matter if another implementation (Traefik, Envoy, ...) would've been used, because all will work with the same Gateway Api *CRDs*.

The url `r500.gateway:30080` is reachable because of a static entry in `/etc/hosts` which points to the only node in the cluster. And on that node, as well as on all others if there would be more, port 30080 is open because of a NodePort *Service*:

```
→ candidate@cka7968:~$ k -n nginx-gateway get svc
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP    PORT(S)         AGE
nginx-gateway   NodePort   10.103.36.0    <none>         80:30080/TCP    58m
```

Step 1

Now we'll have a look at the provided *Ingress* Yaml which we need to convert:

```
→ candidate@cka7968:~$ vim /opt/course/13/ingress.yaml
```

```
# cka7968:/opt/course/13/ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: traffic-director
spec:
  ingressClassName: nginx
  rules:
    - host: r500.gateway
      http:
        paths:
          - backend:
              service:
                name: web-desktop
                port:
                  number: 80
            path: /desktop
            pathType: Prefix
          - backend:
              service:
                name: web-mobile
                port:
                  number: 80
            path: /mobile
            pathType: Prefix
```

We can see two paths `/desktop` and `/mobile` which point to the K8s *Services* `web-desktop` and `web-mobile` . Based on this we create a *HTTPRoute* which replicates the behaviour and in which we reference the existing *Gateway*:

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: traffic-director
  namespace: project-r500
spec:
  parentRefs:
    - name: main    # use the name of the existing Gateway
  hostnames:
    - "r500.gateway"
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /desktop
      backendRefs:
        - name: web-desktop
          port: 80
    - matches:
        - path:
            type: PathPrefix
```

```
        value: /mobile
      backendRefs:
        - name: web-mobile
          port: 80
```

After creation we can test:

```
→ candidate@cka7968:~$ k -n project-r500 get httproute
NAME               HOSTNAMES          AGE
traffic-director   ["r500.gateway"]   7s

→ candidate@cka7968:~$ curl r500.gateway:30080/desktop
Web Desktop App

→ candidate@cka7968:~$ curl r500.gateway:30080/mobile
Web Mobile App

→ candidate@cka7968:~$ curl r500.gateway:30080
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

This looks like what we want!

Step 2

Now things get more interesting and we need to add new path `/auto` which redirects depending on the User-Agent. The User-Agent is handled as a HTTP header and we only have to check for the exact value, hence we can extend our *HTTPRoute* like this:

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: traffic-director
  namespace: project-r500
spec:
  parentRefs:
    - name: main
  hostnames:
    - "r500.gateway"
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /desktop
      backendRefs:
        - name: web-desktop
          port: 80
    - matches:
        - path:
            type: PathPrefix
            value: /mobile
      backendRefs:
        - name: web-mobile
          port: 80
# NEW FROM HERE ON
    - matches:
```

```
      - path:
          type: PathPrefix
          value: /auto
        headers:
        - type: Exact
          name: user-agent
          value: mobile
      backendRefs:
      - name: web-mobile
        port: 80
  - matches:
      - path:
          type: PathPrefix
          value: /auto
      backendRefs:
      - name: web-desktop
        port: 80
```

We added two new rules, the first redirects to mobile conditionally on header value and the second redirects to desktop.

If the question text mentions something like "add one new path /auth" then this doesn't necessarily mean just one entry in the rules array, it can depend on conditions. We added at first the following rule:

```
  - matches:
      - path:
          type: PathPrefix
          value: /auto
        headers:
        - type: Exact
          name: user-agent
          value: mobile
      backendRefs:
      - name: web-mobile
        port: 80
```

Note that we use `- path:` and `header:`, not `- path:` and `- header:`. This means both `path` and `header` will be connected **AND**. So only if the path is `/auto` **AND** the header user-agent is `mobile` we route to mobile.

If we would do the following then these would be connected **OR** and it would be wrong for this question:

```
# WRONG EXAMPLE for explanation
  - matches:
      - path:
          type: PathPrefix
          value: /auto
      - headers:              # WRONG because now path and header are connected OR
        - type: Exact
          name: user-agent
          value: mobile
      backendRefs:
      - name: web-mobile
        port: 80
```

The next rule we added is the one for desktop, at the very end:

```
  - matches:
      - path:
          type: PathPrefix
          value: /auto
      backendRefs:
      - name: web-desktop
        port: 80
```

In this one we don't have to check any header value again because the question required that "otherwise" traffic should be redirected to desktop. So this acts as a "catch all" for route `/auto` .

We need to understand that the order of rules matters. If we would add the desktop rule before the mobile one it wouldn't work because no requests would ever reach the mobile rule.

Our solution should result in this:

```
➜ candidate@cka7968:~$ curl -H "User-Agent: mobile" r500.gateway:30080/auto
Web Mobile App

➜ candidate@cka7968:~$ curl -H "User-Agent: something" r500.gateway:30080/auto
Web Desktop App

➜ candidate@cka7968:~$ curl r500.gateway:30080/auto
Web Desktop App
```

Great, Gateway Api ftw!

Question 14 | Check how long certificates are valid

Solve this question on: `ssh cka9412`

Perform some tasks on cluster certificates:

1. Check how long the kube-apiserver server certificate is valid using openssl or cfssl. Write the expiration date into `/opt/course/14/expiration` . Run the `kubeadm` command to list the expiration dates and confirm both methods show the same one

2. Write the `kubeadm` command that would renew the kube-apiserver certificate into `/opt/course/14/kubeadm-renew-certs.sh`

Answer:

First let's find that certificate:

```
➜ ssh cka9412

➜ candidate@cka9412:~$ sudo -i

➜ root@cka9412:~# find /etc/kubernetes/pki | grep apiserver
/etc/kubernetes/pki/apiserver-etcd-client.key
/etc/kubernetes/pki/apiserver-kubelet-client.key
/etc/kubernetes/pki/apiserver-etcd-client.crt
/etc/kubernetes/pki/apiserver.key
/etc/kubernetes/pki/apiserver-kubelet-client.crt
/etc/kubernetes/pki/apiserver.crt
```

Next we use openssl to find out the expiration date:

```
➜  root@cka9412:~# openssl x509 -noout -text -in /etc/kubernetes/pki/apiserver.crt | grep Validity -A2
        Validity
            Not Before: Oct 29 14:14:27 2024 GMT
            Not After : Oct 29 14:19:27 2025 GMT
```

There we have it, so we write it in the required location:

```
# cka9412:/opt/course/14/expiration
Oct 29 14:19:27 2025 GMT
```

And we use kubeadm to get the expiration to compare:

```
➜  root@cka9412:~# kubeadm certs check-expiration | grep apiserver
apiserver                  Oct 29, 2025 14:19 UTC   356d     ca          no
apiserver-etcd-client      Oct 29, 2025 14:19 UTC   356d     etcd-ca     no
apiserver-kubelet-client   Oct 29, 2025 14:19 UTC   356d     ca          no
```

Looking good, both are the same.

And finally we write the command that would renew the kube-apiserver certificate into the requested location:

```
# cka9412:/opt/course/14/kubeadm-renew-certs.sh
kubeadm certs renew apiserver
```

Question 15 | NetworkPolicy

Solve this question on: `ssh cka7968`

There was a security incident where an intruder was able to access the whole cluster from a single hacked backend *Pod*.

To prevent this create a *NetworkPolicy* called `np-backend` in *Namespace* `project-snake`. It should allow the `backend-*` *Pods* only to:

- Connect to `db1-*` *Pods* on port `1111`
- Connect to `db2-*` *Pods* on port `2222`

Use the `app` *Pod* labels in your policy.

> ℹ️ All *Pods* in the *Namespace* run plain Nginx images. This allows simple connectivity tests like: `k -n project-snake exec POD_NAME -- curl POD_IP:PORT`

> ℹ️ For example, connections from `backend-*` *Pods* to `vault-*` *Pods* on port `3333` should no longer work

Answer:

First we look at the existing *Pods* and their labels:

```
→ ssh cka7968

→ candidate@cka7968:~$ k -n project-snake get pod
NAME        READY    STATUS     RESTARTS    AGE
backend-0   1/1      Running    0           8d
db1-0       1/1      Running    0           8d
db2-0       1/1      Running    0           8d
vault-0     1/1      Running    0           8d

→ candidate@cka7968:~$ k -n project-snake get pod -L app
NAME        READY    STATUS     RESTARTS    AGE     APP
backend-0   1/1      Running    0           8d      backend
db1-0       1/1      Running    0           8d      db1
db2-0       1/1      Running    0           8d      db2
vault-0     1/1      Running    0           8d      vault
```

We test the current connection situation and see nothing is restricted:

```
→ candidate@cka7968:~$ k -n project-snake get pod -o wide
NAME        READY    STATUS     RESTARTS    AGE     IP              ...
backend-0   1/1      Running    0           8d      10.44.0.24     ...
db1-0       1/1      Running    0           8d      10.44.0.25     ...
db2-0       1/1      Running    0           8d      10.44.0.23     ...
vault-0     1/1      Running    0           8d      10.44.0.22     ...

→ candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.25:1111
database one

→ candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.23:2222
database two

→ candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.22:3333
vault secret storage
```

Now we create the *NP* by copying and changing an example from the K8s Docs:

```yaml
# cka7968:/home/candidate/15_np.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: np-backend
  namespace: project-snake
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Egress                    # policy is only about Egress
  egress:
    -                           # first rule
      to:                       # first condition "to"
      - podSelector:
          matchLabels:
            app: db1
      ports:                    # second condition "port"
      - protocol: TCP
```

```
        port: 1111
    -                               # second rule
      to:                           # first condition "to"
      - podSelector:
          matchLabels:
            app: db2
      ports:                        # second condition "port"
      - protocol: TCP
        port: 2222
```

The *NP* above has two rules with two conditions each, it can be read as:

```
allow outgoing traffic if:
   (destination pod has label app=db1 AND port is 1111)
   OR
   (destination pod has label app=db2 AND port is 2222)
```

Wrong example

Now let's shortly look at a wrong example:

```
# WRONG
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: np-backend
  namespace: project-snake
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Egress
  egress:
    -                               # first rule
      to:                           # first condition "to"
      - podSelector:                    # first "to" possibility
          matchLabels:
            app: db1
      - podSelector:                    # second "to" possibility
          matchLabels:
            app: db2
      ports:                        # second condition "ports"
      - protocol: TCP                   # first "ports" possibility
        port: 1111
      - protocol: TCP                   # second "ports" possibility
        port: 2222
```

The *NP* above has one rule with two conditions and two condition-entries each, it can be read as:

```
allow outgoing traffic if:
   (destination pod has label app=db1 OR destination pod has label app=db2)
   AND
   (destination port is 1111 OR destination port is 2222)
```

Using this *NP* it would still be possible for `backend-*` *Pods* to connect to `db2-*` *Pods* on port 1111 for example which should be forbidden.

Create NetworkPolicy

We create the correct *NP*:

```
➜  candidate@cka7968:~$ k -f 15_np.yaml create
```

And to verify:

```
➜  candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.25:1111
database one

➜  candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.23:2222
database two

➜  candidate@cka7968:~$ k -n project-snake exec backend-0 -- curl -s 10.44.0.22:3333
^C
```

Also helpful to use `kubectl describe` on the *NP* to see how K8s has interpreted the policy.

Question 16 | Update CoreDNS Configuration

Solve this question on: `ssh cka5774`

The CoreDNS configuration in the cluster needs to be updated:

1. Make a backup of the existing configuration Yaml and store it at `/opt/course/16/coredns_backup.yaml`. You should be able to fast recover from the backup
2. Update the CoreDNS configuration in the cluster so that DNS resolution for `SERVICE.NAMESPACE.custom-domain` will work exactly like and in addition to `SERVICE.NAMESPACE.cluster.local`

Test your configuration for example from a *Pod* with `busybox:1` image. These commands should result in an IP address:

```
nslookup kubernetes.default.svc.cluster.local
nslookup kubernetes.default.svc.custom-domain
```

Answer:

We have a look at the CoreDNS *Pods*:

```
➜  ssh cka5774

➜  candidate@cka5774:~$ k -n kube-system get deploy,pod
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns      2/2     2            2           42h
...


NAME                             READY   STATUS    RESTARTS   AGE
pod/coredns-74f75f8b69-c4z47     1/1     Running   0          42h
pod/coredns-74f75f8b69-wsnfr     1/1     Running   0          42h
...
```

It looks like CoreDNS is running as a *Deployment* with two replicas.

Step 1

CoreDNS uses a *ConfigMap* by default when installed via Kubeadm. Creating a backup is always a good idea before performing sensitive changes:

```
➜  candidate@cka5774:~$ k -n kube-system get cm
NAME                                 DATA   AGE
coredns                              1      42h
...

➜  candidate@cka5774:~$ k -n kube-system get cm coredns -oyaml > /opt/course/16/coredns_backup.yaml
```

The current configuration looks like this:

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
            lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
            ttl 30
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            max_concurrent 1000
        }
        cache 30 {
            disable success cluster.local
            disable denial cluster.local
        }
        loop
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
...
```

Step 2

We update the config:

```
→ candidate@cka5774:~$ k -n kube-system edit cm coredns
```

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
            lameduck 5s
        }
        ready
        kubernetes custom-domain cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
            ttl 30
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            max_concurrent 1000
        }
        cache 30 {
            disable success cluster.local
            disable denial cluster.local
        }
        loop
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2024-12-26T20:35:11Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "262"
  uid: c76d208f-1bc8-4c0f-a8e8-a8bfa440870e
```

Note that we added `custom-domain` in the same line where `cluster.local` is already defined.

Now we need to restart the *Deployment*:

```
→ candidate@cka5774:~$ k -n kube-system rollout restart deploy coredns
deployment.apps/coredns restarted

→ candidate@cka5774:~$ k -n kube-system get pod
NAME                        READY   STATUS    RESTARTS    AGE
coredns-77d6976b98-jkvqn    1/1     Running   0           13s
coredns-77d6976b98-zdxw8    1/1     Running   0           13s
...
```

We should see both *Pods* restarted and running without errors, this is only the case if there are no syntax errors in the CoreDNS config.

To test the updated configuration we create a *Pod*, image `busybox:1` contains `nslookup` already:

```
→ candidate@cka5774:~$ k run bb --image=busybox:1 -- sh -c 'sleep 1d'
```

```
→ candidate@cka5774:~$ k exec -it bb -- sh

→ / # nslookup kubernetes.default.svc.custom-domain
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   kubernetes.default.svc.custom-domain
Address: 10.96.0.1

→ / # nslookup kubernetes.default.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1
```

We see that now `kubernetes.default.svc.custom-domain` and `kubernetes.default.svc.cluster.local` resolve to IP address `10.96.0.1`. Which is the Kubernetes *Service* in the `default` *Namespace*:

```
→ candidate@cka5774:~$ k -n default get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1     <none>         443/TCP    43h
```

This *Service* is often used from *Pods* that need to communicate with the K8s Api, like operators.

Recover from backup

If we messed something up we could do:

```
→ candidate@cka5774:~$ k diff -f /opt/course/16/coredns_backup.yaml
diff -u -N /tmp/LIVE-591628213/v1.ConfigMap.kube-system.coredns /tmp/MERGED-4230802928/v1.ConfigMap.kube-
system.coredns
--- /tmp/LIVE-591628213/v1.ConfigMap.kube-system.coredns         2024-12-28 16:14:03.158949709 +0000
+++ /tmp/MERGED-4230802928/v1.ConfigMap.kube-system.coredns      2024-12-28 16:14:03.159949781 +0000
@@ -7,7 +7,7 @@
            lameduck 5s
        }
        ready
-        kubernetes custom-domain cluster.local in-addr.arpa ip6.arpa {
+        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
            ttl 30

→ candidate@cka5774:~$ k delete -f /opt/course/16/coredns_backup.yaml
configmap "coredns" deleted

→ candidate@cka5774:~$ k apply -f /opt/course/16/coredns_backup.yaml
configmap/coredns created

→ candidate@cka5774:~$ k -n kube-system rollout restart deploy coredns
deployment.apps/coredns restarted

→ candidate@cka5774:~$ k -n kube-system get pod
NAME                        READY    STATUS     RESTARTS      AGE
coredns-79f94f8fc8-h8z7t    1/1      Running    0             11s
coredns-79f94f8fc8-tj7hg    1/1      Running    0             10s
```

...

But this only works if a backup is available!

Question 17 | Find Container of Pod and check info

Solve this question on: `ssh cka2556`

In *Namespace* `project-tiger` create a *Pod* named `tigers-reunite` of image `httpd:2-alpine` with labels `pod=container` and `container=pod`. Find out on which node the *Pod* is scheduled. Ssh into that node and find the containerd container belonging to that *Pod*.

Using command `crictl`:

1. Write the ID of the container and the `info.runtimeType` into `/opt/course/17/pod-container.txt`

2. Write the logs of the container into `/opt/course/17/pod-container.log`

    ℹ️ You can connect to a worker node using `ssh cka2556-node1` or `ssh cka2556-node2` from `cka2556`

Answer:

    ℹ️ In this environment `crictl` can be used for container management. In the real exam this could also be `docker`. Both commands can be used with the same arguments.

First we create the *Pod*:

```
➜ ssh cka2556

➜ candidate@cka2556:~$ k -n project-tiger run tigers-reunite --image=httpd:2-alpine --labels
"pod=container,container=pod"
pod/tigers-reunite created
```

Next we find out the node it's scheduled on:

```
➜ candidate@cka2556:~$ candidate@cka2556:~$ k -n project-tiger get pod -o wide
NAME                                  READY    ...    NODE
tigers-for-rent-web-57558cfbf8-4tldr  1/1      ...    cka2556-node1
tigers-for-rent-web-57558cfbf8-5pz4z  1/1      ...    cka2556-node2
tigers-reunite                        1/1      ...    cka2556-node1
```

Here it's `cka2556-node1` so we ssh into that node and and check the container info:

```
→ candidate@cka2556:~$ ssh cka2556-node1

→ candidate@cka2556-node1:~$ sudo -i

→ root@cka2556-node1:~# crictl ps | grep tigers-reunite
ba62e5d465ff0   a7ccaadd632cf   2 minutes ago   Running   tigers-reunite   ...
```

Step 1

Having the container we can `crictl inspect` it for the runtimeType:

```
→ root@cka2556-node1:~# crictl inspect ba62e5d465ff0 | grep runtimeType
    "runtimeType": "io.containerd.runc.v2",
```

Now we create the requested file on `cka2556`:

```
# cka2556:/opt/course/17/pod-container.txt
ba62e5d465ff0 io.containerd.runc.v2
```

Step 2

Finally we query the container logs:

```
→ root@cka2556-node1:~# crictl logs ba62e5d465ff0
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.44.0.29. Set
the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.44.0.29. Set
the 'ServerName' directive globally to suppress this message
[Tue Oct 29 15:12:57.211347 2024] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.62 (Unix) configured
-- resuming normal operations
[Tue Oct 29 15:12:57.211841 2024] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D FOREGROUND'
```

Here we run `crictl logs` on the worker node and copy the content manually, that works if it's not a lot of logs. Otherwise we could write the logs into a file on `cka2556-node1` and download the file via scp from `cka2556`.

The file should look like this:

```
# cka2556:/opt/course/17/pod-container.log
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.44.0.37. Set
the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.44.0.37. Set
the 'ServerName' directive globally to suppress this message
[Mon Sep 13 13:32:18.555280 2021] [mpm_event:notice] [pid 1:tid 139929534545224] AH00489: Apache/2.4.41
(Unix) configured -- resuming normal operations
[Mon Sep 13 13:32:18.555610 2021] [core:notice] [pid 1:tid 139929534545224] AH00094: Command line: 'httpd -D
FOREGROUND'
```

CKA Simulator Preview Kubernetes 1.33

This is a preview of the CKA Simulator content. The full CKA Simulator is available in two versions: A and B. Each version contains at least 17 different questions. These preview questions are in addition to the provided ones and can also be solved in the interactive environment.

Preview Question 1 | ETCD Information

Solve this question on: `ssh cka9412`

The cluster admin asked you to find out the following information about etcd running on `cka9412` :

- Server private key location
- Server certificate expiration date
- Is client certificate authentication enabled

Write these information into `/opt/course/p1/etcd-info.txt`

Answer:

Find out etcd information

Let's check the nodes:

```
➜ ssh cka9412

➜ candidate@cka9412:~$ k get node
NAME            STATUS    ROLES           AGE    VERSION
cka9412         Ready     control-plane   9d     v1.33.1
cka9412-node1   Ready     <none>          9d     v1.33.1
```

First we check how etcd is setup in this cluster:

```
➜ candidate@cka9412:~$ sudo -i

➜ root@cka9412:~# k -n kube-system get pod
NAME                              READY    STATUS     RESTARTS    AGE
coredns-6f4c58b94d-djpgr          1/1      Running    0           8d
coredns-6f4c58b94d-ds6ch          1/1      Running    0           8d
etcd-cka9412                      1/1      Running    0           9d
kube-apiserver-cka9412            1/1      Running    0           9d
kube-controller-manager-cka9412   1/1      Running    0           9d
```

```
kube-proxy-7zhtk                    1/1    Running   0            9d
kube-proxy-nbzrt                    1/1    Running   0            9d
kube-scheduler-cka9412              1/1    Running   0            9d
weave-net-h7n8j                     2/2    Running   1 (9d ago)   9d
weave-net-rbhgl                     2/2    Running   1 (9d ago)   9d
```

We see it's running as a *Pod*, more specific a static *Pod*. So we check for the default kubelet directory for static manifests:

```
➜  root@cka9412:~# find /etc/kubernetes/manifests/
/etc/kubernetes/manifests/
/etc/kubernetes/manifests/kube-controller-manager.yaml
/etc/kubernetes/manifests/kube-apiserver.yaml
/etc/kubernetes/manifests/etcd.yaml
/etc/kubernetes/manifests/kube-scheduler.yaml


➜  root@cka9412:~# vim /etc/kubernetes/manifests/etcd.yaml
```

So we look at the yaml and the parameters with which etcd is started:

```
# cka9412:/etc/kubernetes/manifests/etcd.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/etcd.advertise-client-urls: https://192.168.100.21:2379
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.100.21:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt         # server certificate
    - --client-cert-auth=true                                 # enabled
    - --data-dir=/var/lib/etcd
    - --experimental-initial-corrupt-check=true
    - --experimental-watch-progress-notify-interval=5s
    - --initial-advertise-peer-urls=https://192.168.100.21:2380
    - --initial-cluster=cka9412=https://192.168.100.21:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key          # server private key
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.100.21:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://192.168.100.21:2380
    - --name=cka9412
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: registry.k8s.io/etcd:3.5.15-0
    imagePullPolicy: IfNotPresent
...
```

We see that client authentication is enabled and also the requested path to the server private key, now let's find out the expiration of the server certificate:

```
→ root@cka9412:~# openssl x509  -noout -text -in /etc/kubernetes/pki/etcd/server.crt | grep Validity -A2
        Validity
            Not Before: Oct 29 14:14:27 2024 GMT
            Not After : Oct 29 14:19:27 2025 GMT
```

There we have it. Let's write the information into the requested file:

```
# /opt/course/p1/etcd-info.txt
Server private key location: /etc/kubernetes/pki/etcd/server.key
Server certificate expiration date: Oct 29 14:19:27 2025 GMT
Is client certificate authentication enabled: yes
```

Preview Question 2 | Kube-Proxy iptables

Solve this question on: `ssh cka2556`

You're asked to confirm that kube-proxy is running correctly. For this perform the following in *Namespace* `project-hamster`:

1. Create *Pod* `p2-pod` with image `nginx:1-alpine`
2. Create *Service* `p2-service` which exposes the *Pod* internally in the cluster on port `3000->80`
3. Write the iptables rules of node `cka2556` belonging the created *Service* `p2-service` into file `/opt/course/p2/iptables.txt`
4. Delete the *Service* and confirm that the iptables rules are gone again

Answer:

Step 1: Create the *Pod*

First we create the *Pod*:

```
→ ssh cka2556


→ candidate@cka2556:~$ k -n project-hamster run p2-pod --image=nginx:1-alpine
pod/p2-pod created
```

Step 2: Create the *Service*

Next we create the *Service*:

```
➜  candidate@cka2556:~$ k -n project-hamster expose pod p2-pod --name p2-service --port 3000 --target-port 80

➜  candidate@cka2556:~$ k -n project-hamster get pod,svc,ep
NAME                   READY    STATUS     RESTARTS     AGE
pod/p2-pod             1/1      Running    0            2m31s


NAME                   TYPE          CLUSTER-IP        EXTERNAL-IP    PORT(S)      AGE
service/p2-service     ClusterIP     10.105.128.247    <none>         3000/TCP     1s


NAME                   ENDPOINTS          AGE
endpoints/p2-service   10.44.0.31:80      1s
```

We should see that *Pods* and *Services* are connected, hence the *Service* should have *Endpoints*.

(Optional) Confirm kube-proxy is running and is using iptables

The idea here is to find the kube-proxy container and check its logs:

```
➜  candidate@cka2556:~$ sudo -i

➜  root@cka2556$ crictl ps | grep kube-proxy
67cccaf8310a1   505d571f5fd56   9 days ago       Running     kube-proxy ...

➜  root@cka2556~# crictl logs 67cccaf8310a1
I1029 14:10:23.984360       1 server_linux.go:66] "Using iptables proxy"
...
```

This could be repeated on each controlplane and worker node where the result should be the same.

Step 3: Check kube-proxy is creating iptables rules

Now we check the iptables rules on every node first manually:

```
➜  root@cka2556:~# iptables-save | grep p2-service
-A KUBE-SEP-55IRFJIRWHLCQ6QX -s 10.44.0.31/32 -m comment --comment "project-hamster/p2-service" -j KUBE-
MARK-MASQ
-A KUBE-SEP-55IRFJIRWHLCQ6QX -p tcp -m comment --comment "project-hamster/p2-service" -m tcp -j DNAT --to-
destination 10.44.0.31:80
-A KUBE-SERVICES -d 10.105.128.247/32 -p tcp -m comment --comment "project-hamster/p2-service cluster IP" -m
tcp --dport 3000 -j KUBE-SVC-U5ZRKF27Y7YDAZTN
-A KUBE-SVC-U5ZRKF27Y7YDAZTN ! -s 10.244.0.0/16 -d 10.105.128.247/32 -p tcp -m comment --comment "project-
hamster/p2-service cluster IP" -m tcp --dport 3000 -j KUBE-MARK-MASQ
-A KUBE-SVC-U5ZRKF27Y7YDAZTN -m comment --comment "project-hamster/p2-service -> 10.44.0.31:80" -j KUBE-SEP-
55IRFJIRWHLCQ6QX
# Warning: iptables-legacy tables present, use iptables-legacy-save to see them
```

Great. Now let's write these logs into the requested file:

```
➜  root@cka2556:~# iptables-save | grep p2-service > /opt/course/p2/iptables.txt
```

Delete the *Service* and confirm iptables rules are gone

Delete the *Service* and confirm the iptables rules are gone::

```
→ root@cka2556:~# k -n project-hamster delete svc p2-service
service "p2-service" deleted

→ root@cka2556:~# iptables-save | grep p2-service

→ root@cka2556:~#
```

Kubernetes *Services* are implemented using iptables rules (with default config) on all nodes. Every time a *Service* has been altered, created, deleted or *Endpoints* of a *Service* have changed, the kube-apiserver contacts every node's kube-proxy to update the iptables rules according to the current state.

Preview Question 3 | Change Service CIDR

Solve this question on: `ssh cka9412`

1. Create a *Pod* named `check-ip` in *Namespace* `default` using image `httpd:2-alpine`
2. Expose it on port `80` as a ClusterIP *Service* named `check-ip-service`. Remember/output the IP of that *Service*
3. Change the Service CIDR to `11.96.0.0/12` for the cluster
4. Create a second *Service* named `check-ip-service2` pointing to the same *Pod*

ℹ️ The second *Service* should get an IP address from the new CIDR range

Answer:

Let's create the *Pod* and expose it:

```
→ ssh cka9412

→ candidate@cka9412:~$ k run check-ip --image=httpd:2-alpine
pod/check-ip created

→ candidate@cka9412:~$ k expose pod check-ip --name check-ip-service --port 80
```

And check the *Service* IP:

```
→ candidate@cka9412:~$ k get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
check-ip-service   ClusterIP   10.109.84.110    <none>        80/TCP    13s
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP   9d
```

Now we change the *Service* CIDR in the kube-apiserver manifest:

```
→ candidate@cka9412:~$ sudo -i

→ root@cka9412:~# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
# cka9412:/etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.100.21
...
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-cluster-ip-range=11.96.0.0/12                # change
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
...
```

We wait for the kube-apiserver to be restarted, which can take a minute:

```
→ root@cka9412:~# watch crictl ps

→ root@cka9412:~# kubectl -n kube-system get pod | grep api
kube-apiserver-cka9412            1/1       Running   0                20s
```

Now we do the same for the controller manager:

```
→ root@cka9412:~# vim /etc/kubernetes/manifests/kube-controller-manager.yaml
```

```
# /etc/kubernetes/manifests/kube-controller-manager.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-cidr=10.244.0.0/16
    - --cluster-name=kubernetes
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
```

```
        - --kubeconfig=/etc/kubernetes/controller-manager.conf
        - --leader-elect=true
        - --node-cidr-mask-size=24
        - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
        - --root-ca-file=/etc/kubernetes/pki/ca.crt
        - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
        - --service-cluster-ip-range=11.96.0.0/12          # change
        - --use-service-account-credentials=true
```

We wait for the kube-controller-manager to be restarted, which can take a minute:

```
➜  root@cka9412:~# watch crictl ps


➜  root@cka9412:~# kubectl -n kube-system get pod | grep controller
kube-controller-manager-cka9412    1/1     Running   0                   39s
```

Checking our *Service* again:

```
➜  root@cka9412:~$ k get svc
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
check-ip-service    ClusterIP   10.109.84.110    <none>         80/TCP     5m3s
kubernetes          ClusterIP   10.96.0.1        <none>         443/TCP    9d
```

Nothing changed so far. Now we create the second *Service*:

```
➜  root@cka9412:~$ k expose pod check-ip --name check-ip-service2 --port 80
```

And check again:

```
➜  root@cka9412:~$ k get svc
NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
service/check-ip-service    ClusterIP   10.109.84.110    <none>         80/TCP     5m55s
service/check-ip-service2   ClusterIP   11.105.52.114    <none>         80/TCP     29s
service/kubernetes          ClusterIP   10.96.0.1        <none>         443/TCP    9d

NAME                          ENDPOINTS               AGE
endpoints/check-ip-service    10.44.0.3:80            5m55s
endpoints/check-ip-service2   10.44.0.3:80            29s
endpoints/kubernetes          192.168.100.21:6443     9d
```

There we go, the new *Service* got an IP of the updated range assigned. We also see that both *Services* have our *Pod* as endpoint.

CKA Tips Kubernetes 1.33

In this section we'll provide some tips on how to handle the CKA exam and browser terminal.

Knowledge

Study all topics as proposed in the curriculum till you feel comfortable with all.

General

- Study all topics as proposed in the curriculum till you feel comfortable with all

- Do 1 or 2 test session with this CKA Simulator. Understand the solutions and maybe try out other ways to achieve the same thing.

- Setup your aliases, be fast and breath `kubectl`

- The majority of tasks in the CKA will also be around creating Kubernetes resources, like it's tested in the CKAD. So preparing a bit for the CKAD can't hurt.

- Learn and Study the in-browser scenarios on https://killercoda.com/killer-shell-cka (and maybe for CKAD https://killercoda.com/killer-shell-ckad)

- Imagine and create your own scenarios to solve

Components

- Understanding Kubernetes components and being able to fix and investigate clusters: https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster

- Know advanced scheduling: https://kubernetes.io/docs/concepts/scheduling/kube-scheduler

- When you have to fix a component (like kubelet) in one cluster, just check how it's setup on another node in the same or even another cluster. You can copy config files over etc

- If you like you can look at Kubernetes The Hard Way once. But it's NOT necessary to do, the CKA is not that complex. But KTHW helps understanding the concepts

- You should install your own cluster using kubeadm (one controlplane, one worker) in a VM or using a cloud provider and investigate the components

- Know how to use Kubeadm to for example add nodes to a cluster

- Know how to create an Ingress resources

- Know how to snapshot/restore ETCD from another machine

CKA Exam Info

Read the Curriculum

https://github.com/cncf/curriculum

Read the Handbook

https://docs.linuxfoundation.org/tc-docs/certification/lf-handbook2

Read the important tips

https://docs.linuxfoundation.org/tc-docs/certification/tips-cka-and-ckad

Read the FAQ

https://docs.linuxfoundation.org/tc-docs/certification/faq-cka-ckad

Kubernetes documentation

Get familiar with the Kubernetes documentation and be able to use the search. Allowed resources are:

- https://kubernetes.io/docs

- https://kubernetes.io/blog

- https://helm.sh/docs

- https://gateway-api.sigs.k8s.io

  ℹ️ Verify the list here

The Exam UI / Remote Desktop

The real exam, as well as the simulator, provides a Remote Desktop (XFCE) on Ubuntu/Debian. Coming from OSX/Windows there will be changes in copy&paste for example.

Official Information

ExamUI: Performance Based Exams

Lagging

There could be some lagging, definitely make sure you are using a good internet connection because your webcam and screen are transferring all the time.

Kubectl autocompletion and commands

The following are installed or pre-configured, verify the list here:

- `kubectl` with `k` alias and Bash autocompletion
- `yq` or YAML processing
- `curl` and `wget` for testing web services
- `man` and man pages for further documentation

  ℹ️ You're allowed to install tools, like `tmux` for terminal multiplexing or `jq` for JSON processing

Copy & Paste

Copy and pasting will work like normal in a Linux Environment:

What always works: copy+paste using right mouse context menu What works in Terminal: Ctrl+Shift+c and Ctrl+Shift+v What works in other apps like Firefox: Ctrl+c and Ctrl+v

Score

There are 15-20 questions in the exam. Your results will be automatically checked according to the handbook. If you don't agree with the results you can request a review by contacting the Linux Foundation Support.

Notepad & Flagging Questions

You can flag questions to return to later. This is just a marker for yourself and won't affect scoring. You also have access to a simple notepad in the browser which can be used to store any kind of plain text. It might make sense to use this and write down additional information about flagged questions. Instead of using the notepad you could also open Mousepad (XFCE application inside the Remote Desktop) or create a file with Vim.

VSCodium

You can use VSCodium to edit files and you can also use its terminal to run commands. You're not allowed to install any VSCodium extensions.

Servers

Each question needs to be solved on a specific instance other than your main terminal. You'll need to connect to the correct instance via ssh, the command is provided before each question.

PSI Bridge

Starting with PSI Bridge:

- The exam will now be taken using the PSI Secure Browser, which can be downloaded using the newest versions of Microsoft Edge, Safari, Chrome, or Firefox
- Multiple monitors will no longer be permitted
- Use of personal bookmarks will no longer be permitted

The new ExamUI includes improved features such as:

- A remote desktop configured with the tools and software needed to complete the tasks
- A timer that displays the actual time remaining (in minutes) and provides an alert with 30, 15, or 5 minute remaining
- The content panel remains the same (presented on the Left Hand Side of the ExamUI)

Read more here.

Terminal Handling

Bash Aliases

In the real exam, each question has to be solved on a different instance to which you connect via ssh. This means it's not advised to configure bash aliases because they wouldn't be available on the instances accessed by ssh.

Be fast

Use the `history` command to reuse already entered commands or use even faster history search through **Ctrl r** .

If a command takes some time to execute, like sometimes `kubectl delete pod x`. You can put a task in the background using **Ctrl z** and pull it back into foreground running command `fg` .

You can delete *pods* fast with:

```
k delete pod x --grace-period 0 --force
```

Vim

Be great with vim.

Settings

In case you face a situation where vim is not configured properly and you face for example issues with pasting copied content you should be able to configure via `~/.vimrc` or by entering manually in vim settings mode:

```
set tabstop=2
set expandtab
set shiftwidth=2
```

The `expandtab` make sure to use spaces for tabs.

Note that changes in `~/.vimrc` will not be transferred when connecting to other instances via ssh.

Toggle vim line numbers

When in `vim` you can press **Esc** and type `:set number` or `:set nonumber` followed by **Enter** to toggle line numbers. This can be useful when finding syntax errors based on line - but can be bad when wanting to mark&copy by mouse. You can also just jump to a line number with **Esc** `:22` + **Enter**.

Copy&Paste

Get used to copy/paste/cut with vim:

```
Mark lines: Esc+V (then arrow keys)
Copy marked lines: y
Cut marked lines: d
Past lines: p or P
```

Indent multiple lines

To indent multiple lines press **Esc** and type `:set shiftwidth=2`. First mark multiple lines using `Shift v` and the up/down keys. Then to indent the marked lines press `>` or `<`. You can then press `.` to repeat the action.

About

FAQ

Support

Store

Pricing

Legal Notice / Impressum

Privacy Policy / Datenschutz

Terms / AGB

CONTENT

CKS

CKA

CKAD

LFCS

LINKS