# Tips----

Recently I have successfully passed exam for [Certified Kubernetes Administrator (CKA)](https://www.cncf.io/certification/cka/) and I would like to share learning tips and tricks as well as resources that helped me prepare and pass the exam.

# Resources

The only resource needed to pass the exam was [Certified Kubernetes Administrator (CKA) with Practice Tests](https://kodekloud.com/p/certified-kubernetes-administrator-with-practice-tests). Very comprehensive and in depth course that contains all needed information and practice labs to pass the exam! Mumshad, the author of the course and owner of [KodeKloud](https://kodekloud.com/)did really great work designing it.

There is also one more resource I would like to mention, it is not needed to pass or prepare for the exam, but content is really great and it deserves a shout out. Videos from [Just me and Opensource](https://www.youtube.com/user/wenkatn/featured) YouTube channel. Author does really great work creating hands on practical videos tackling various kubernetes related topics.

# Tip #1: Create your own kubernetes cluster to play around

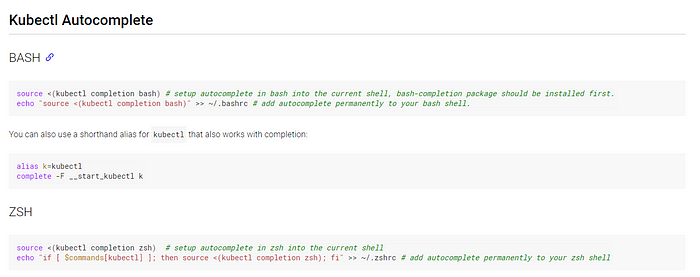
Practice is very important for the exam, so use every opportunity to get familiar with kubectl command line tool and YAML files. One option is to setup local kubernetes cluster using for example: [minikube](https://kubernetes.io/docs/tasks/tools/install-minikube/), [microk8s](https://microk8s.io/docs/)or [kind](https://kind.sigs.k8s.io/). All of those are great options to quickly spin up kubernetes cluster on your machine.

If your machine does not have enough resources or you simply do not want to install anything locally, you can quickly spin up kubernetes cluster in any cloud provider. You can check out my blog where I describe how to setup AKS cluster <https://medium.com/@piotrzan/try-kubernetes-in-cloud-for-free-e5e431c507a7>, check my GitHub repos with instructions how to deploy managed kubernetes on [Azure](https://github.com/Piotr1215/terraform-aks)or [GCP](https://github.com/Piotr1215/terraform-gcp)

# Tip #2: Kubectl alias and bash/zsh completion

During the exam you are required to demonstrate practical knowledge of kubernetes and all this under time pressure. Each second counts, so one of the most important tips that helped me were those about speed.

Make sure to setup alias for kubeclt and bash/zsh completion. This information is available on the kubernetes documentation page. It will help you be much faster with commands and most importantly completion sources pod and other kubernetes objects names that you don’t need to copy or type.



# Tip #3: Take advantage of kubernetes documentation

During the exam you can have one additional chrome tab open with kubernetes documentation page, kubernetes blog and GitHub artifacts for the page. It helped me to prepare bookmarks to quickly locate part of YAML to copy and paste to terminal.

You can get the bookmarks from [my GitHub gist](https://gist.github.com/Piotr1215/016ba7218a1a949574786fb9b92382c1) and import them into your browser! Best way is to clone the gist as I’m making minor updates.

# Tip #4: Know vim (or nano) editors well

A lot of exam tasks required editing existing YAML files and ability to use vim (in my case) or nano text editors quickly and efficiently is very important. There are plenty of pages with keyboard shortcuts.

# ‍Tip #5: Questions priority

Each question is “weighted” and for CKA you have on average 6 minutes per question. If you find yourself stuck, it’s better to note question number in notepad available within the environment and come back to it later.

# ‍Tip #6: Get very familiar with those kubectl tricks

# ‍Tip #7: Stay calm

I’m not sure about you, but performance based exams always make me nervous and stress often gets best of me. It is helpful to remember that CKA gives you one free retry, so even if you don’t pass first time, don’t worry.

# Bonus

This is not strictly needed for the exam (at least I didn’t need to use the below commands), but it might be useful:

*‍I wish you best of luck in getting your CKA certification!*

https://miro.medium.com/v2/resize:fit:700/0*Piks8Tu6xUYpF4DU

**Follow us on**[**Twitter**](https://twitter.com/joinfaun)🐦**and**[**Facebook**](https://www.facebook.com/faun.dev/)👥**and**[**Instagram**](https://instagram.com/fauncommunity/)📷 **and join our**[**Facebook**](https://www.facebook.com/groups/364904580892967/)**and**[**Linkedin**](https://www.linkedin.com/company/faundev)**Groups**💬**.**

**To join our community Slack team chat**🗣️ **read our weekly Faun topics**🗞️,**and connect with the community**📣**click here⬇**

[Write](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Fmedium.com%2Fnew-story&source=---top_nav_layout_nav-----------------------new_post_topnav------------------)

Sign up

[Sign in](https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Fmedium.com%2F%40wattsdave%2Fretake-certified-kubernetes-administrator-cka-bf4cbccbafbc&source=post_page---top_nav_layout_nav-----------------------global_nav------------------)

https://miro.medium.com/v2/resize:fill:32:32/1*dmbNkD5D-u45r44go_cf0g.png

# Retake: Certified Kubernetes Administrator (CKA)

[[Dave Watts](https://medium.com/@wattsdave?source=post_page---byline--bf4cbccbafbc---------------------------------------)](https://medium.com/@wattsdave?source=post_page---byline--bf4cbccbafbc---------------------------------------)

[Dave Watts](https://medium.com/@wattsdave?source=post_page---byline--bf4cbccbafbc---------------------------------------)

Follow

4 min read

·

Apr 27, 2025

74

2

I recently retook the CKA after it had expired. Thankfully, I already had the exam paid for by my previous employer (much appreciation to them), so it was just a matter of revisiting the material from three years ago and working out what changes I needed to adjust for.

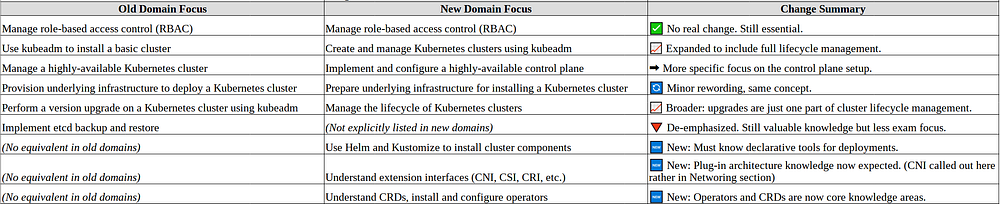
# CKA Retake

My CKA expired in early March, and I got around to retaking it towards the end of the month. It was a bit tricky this time because the CKA had changed some of the domains and competencies in February. This always makes things harder, as training material usually lags a few months behind.

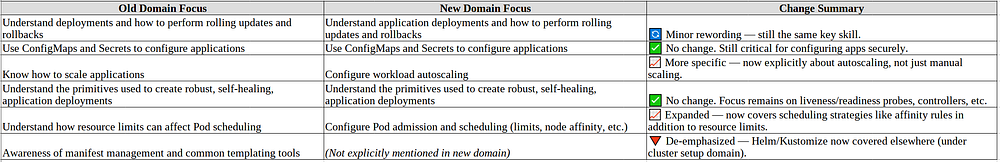
The good thing about the CKA, though, is that a lot of the new material is already covered through other services. Case in point: KodeKloud was able to respond quickly, since topics like Helm and Kustomize were already included in their other courses.

# CKA Competences: OLD vs NEW

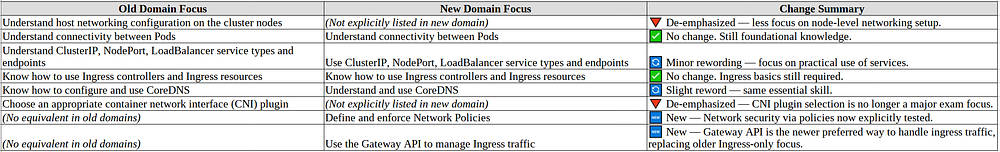
Below is a summary I used to help focus my study, as I was quite nervous about how the wording changes had emphasized or de-emphasized certain areas.



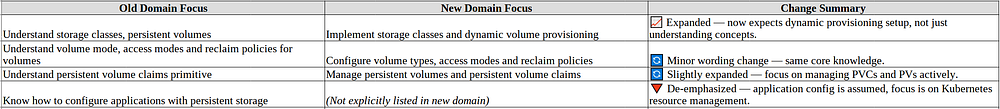
25% Cluster Architecture, Installation & Configuration: CKA Old vs New



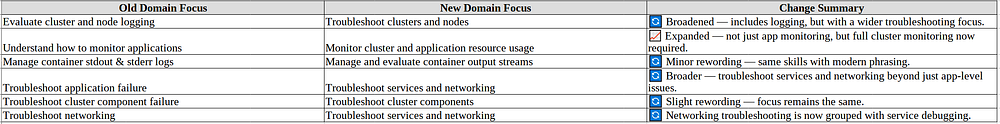
15% Workloads and Scheduling: CKA Old vs New



20% Services & Networking: CKA Old vs New



10% — Storage: CKA Old vs New



30% — Troubleshooting: CKA Old vs New

# Changes Since I Originally Took CKA in March 2022

The first thing I needed to do was get across the changes in Kubernetes itself since 2022. Back then, the version I used was 1.23, and this time around I was tested on 1.32. I found I needed to properly cover the following topics:

* **Gateway API** (and how it compares to Ingress)
* **Horizontal Pod Autoscaler** and **Vertical Pod Autoscaler**
* **Helm** (although I’d already covered this in CKAD and through work)
* **CRDs** (I did cover them the first time around, but they were specifically called out in the updated competencies)
* **Changes to how Service Accounts are used with Pods**
* **Troubleshooting** felt more general — having a good understanding of Linux was really useful.

In general, the updated competencies felt broader and more generic, which concerned me a bit leading into the exam. There were also some very specific topics called out, like CRDs and plugins (e.g., CNI, CRI).

# Killer.sh CKA Changes

One thing I forgot to mention earlier — the Killer.sh CKA simulator has changed since I first used it. It used to be one big exam with about 25 questions, but now it’s split into two separate exams. Both of them are closer in question count to the real exam with 17 in each. So you get one go at CKA Simulator A and CKA Simulator B, compared to previously you had two sessions on the same exam simulator.



New Killer.sh Simulators

I went through both, and they definitely felt more balanced than before — not quite as punishing as the old one, but still solid prep. The newer format lines up better with the updated CKA domains, and the questions cover newer stuff like Gateway API and Kustomize, which helped ease some of the nerves.

But to be honest, after sitting the real exam, I didn’t feel as prepared as I expected to be. The simulator helped, for sure, but it didn’t fully reflect the broader, slightly more abstract nature of some of the real exam questions — especially stuff that drew more heavily on practical Linux knowledge or troubleshooting unfamiliar edge cases. So while the updated simulator is still worth doing (and probably essential), don’t rely on it alone.

# How was the Exam?

So firstly, I only just passed and did worse than I did three years ago. But I actually felt I had done better in this attempt then when I did it three years ago. I must of butchered a couple of questions somehow.

Leading into the exam, I got a bit spooked reading people’s experiences on Reddit about how much harder the exam had become. I think that’s true in a general sense — you actually need a broader understanding of Kubernetes and Linux than you did previously to pass.

That’s the kicker: there were questions I did okay on because I’d done my LFCS exam and spent time playing around with Kubernetes over the last three years — questions you might not be fully prepared for if you only used something like KodeKloud.

Overall though, the depth the exam questions for Kubernetes itself feels less than it did three years ago. Goodluck!

# Prority

Find the Pod with the highest priority in Namespace management and delete it.

Tip

Solution

k -n management get pod -o yaml | grep -i priority -B 20

k -n management delete pod sprinter

CHECK

* Exam Desktop
* Editor
* Tab 1
* [**+**](https://killercoda.com/killer-shell-cka/scenario/scheduling-priority)

60 min

[](https://www.faun.dev/join/?utm_source=medium.com%2Ffaun&utm_medium=medium&utm_campaign=faunmediumbanner)

## If this post was helpful, please click the clap 👏 button below a few times to show your support for the author! ⬇

[Kubernetes](https://medium.com/tag/kubernetes?source=post_page-----ca868fc678c9---------------------------------------)

[Cka](https://medium.com/tag/cka?source=post_page-----ca868fc678c9---------------------------------------)

[Cncf](https://medium.com/tag/cncf?source=post_page-----ca868fc678c9---------------------------------------)

[Cloud Computing](https://medium.com/tag/cloud-computing?source=post_page-----ca868fc678c9---------------------------------------)

[Certification](https://medium.com/tag/certification?source=post_page-----ca868fc678c9---------------------------------------)

426

5

[[](https://faun.pub/?source=post_page---post_publication_info--ca868fc678c9---------------------------------------)](https://faun.pub/?source=post_page---post_publication_info--ca868fc678c9---------------------------------------)

## **[Published in FAUN — Developer Community 🐾](https://faun.pub/?source=post_page---post_publication_info--ca868fc678c9---------------------------------------)**

[21K followers](https://faun.pub/followers?source=post_page---post_publication_info--ca868fc678c9---------------------------------------)

·[Last published 2 days ago](https://faun.pub/deploying-wazuh-on-docker-simplifying-security-monitoring-configuration-1d4cc3ec8870?source=post_page---post_publication_info--ca868fc678c9---------------------------------------)

We help developers learn and grow by keeping them up with what matters. 👉 [www.faun.dev](http://www.faun.dev/)

Follow

[[](https://piotrzan.medium.com/?source=post_page---post_author_info--ca868fc678c9---------------------------------------)](https://piotrzan.medium.com/?source=post_page---post_author_info--ca868fc678c9---------------------------------------)

## **[Written by Piotr](https://piotrzan.medium.com/?source=post_page---post_author_info--ca868fc678c9---------------------------------------)**

[6.9K followers](https://piotrzan.medium.com/followers?source=post_page---post_author_info--ca868fc678c9---------------------------------------)

·[66 following](https://medium.com/@piotrzan/following?source=post_page---post_author_info--ca868fc678c9---------------------------------------)

My mission is to share enthusiasm and appreciation for digital technology. I want to inspire you to try tools, create and use software and learn new things.

Follow

## **Responses (5)**

# ----end

# configuration

k cluster-info --kubeconfig=./.kubeconfig

## Export useful variables

alias k=kubectl  
export dry='--dry-run=client -o=yaml'  
export oy='-o=yaml'  
alias kn='kubectl config set-context --current --namespace '  
export ETCDCTL\_API=3

This is so that you can call $dry to export yaml files instead of creating the objects

# Make a pod yaml  
k run <pod-name> --image=<image> $dry > pod.yaml  
  
# Apply  
k apply -f ./pod.yaml  
  
# Get it back as yaml  
k get po <pod-name> $oy

# Workloads

## Creating pods

k run <pod-name> --image=<image> $dry

## Creating pods with security context

Build a yaml output and add this as part of `

## Creating a deployment

k create deploy <deploy> --replicas=<n> --image=<image> $dry

# Initiating a cluster with kubeadm

## Install kubelet and kubeadm

<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init/>

apt update  
apt search kubeadm kubelet  
apt install kubeadm=1.26.0-00 kubelet=1.26.0-00

## **Init kubeadm in the controlplane**

IP\_ADDRESS=$(ifconfig eth0 | grep 'inet ' | cut -d: -f2 | awk '{print $2}')  
  
kubeadm init \  
 --apiserver-advertise-address=$IP\_ADDRESSS \  
 --apiserver-cert-extra-sans=controlplane \  
 --pod-network-cidr=10.244.0.0/16

## **Make a token from the controlplane:**

<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-token/>

controlplane ~ ✖ kubeadm token create --print-join-command  
kubeadm join 192.15.211.6:6443 --token XXX --discovery-token-ca-cert-hash sha256:1493d93e085bcaa30819bc10958c54ff69a2ebea37a00632fb37c0621fc40139

## **Join from a worker node**

<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-join/>

workernode$ kubeadm join 192.15.211.6:6443 --token XXX --discovery-token-ca-cert-hash sha256:1493d93e085bcaa30819bc10958c54ff69a2ebea37a00632fb37c0621fc40139

## Back in the control plane, check the nodes:

controlplane ~ ➜ k get no  
NAME STATUS ROLES AGE VERSION  
controlplane NotReady control-plane 8m4s v1.26.0  
node01 NotReady <none> 8s v1.26.0

## **Install a CNI (eg: flannel)**

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

# Cluster Maintenance

## Backing up etcd

<https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>

export ETCDCTL\_API=3   
etcdctl --endpoints $ENDPOINTS \  
 --cert=/etc/kubernetes/pki/etcd/server.crt \  
 --key=/etc/kubernetes/pki/etcd/server.key \  
 --cacert=/etc/kubernetes/pki/etcd/ca.crt \  
 snapshot save <output>

Endpoints can be found in:

cat /etc/kubernetes/manifests/etcd.yaml | grep listen-client-url

## Restoring ectd

Extract the db output with:

export ETCDCTL\_API=3   
etcdctl --endpoints $ENDPOINTS \  
 --cert=/etc/kubernetes/pki/etcd/server.crt \  
 --key=/etc/kubernetes/pki/etcd/server.key \  
 --cacert=/etc/kubernetes/pki/etcd/ca.crt \  
 snapshot restore <output>

Then mount the output directory in the static pod: /etc/kubernetes/manifest/etcd.yaml

volumeMounts:  
 - mountPath: <your-output-directory> # Change this  
 name: etcd-data

## Creating a new user

Create the keys:

<https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#create-private-key>

openssl genrsa -out myuser.key 2048  
openssl req -new -key myuser.key -out myuser.csr

Create a CSR k8s object: <https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#create-certificatesigningrequest>

cat <<EOF  
apiVersion: certificates.k8s.io/v1  
kind: CertificateSigningRequest  
metadata:  
 name: myuser  
spec:  
 request: $(cat myuser.csr | base64 | tr -d "\n")  
 signerName: kubernetes.io/kube-apiserver-client  
 expirationSeconds: 86400 # one day  
 usages:  
 - client auth  
EOF

Save that as a csr.yaml file

Apply it:

k apply -f ./csr.yaml

Approve the CSR:

<https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#approve-certificate-signing-request>

kubectl certificate approve myuser

## Creating a role

k create role --help  
  
kubectl create role $dry --verb=<verb1,verb2,verb3> --resource=<resource1,resource2> <role>

## Create role binding

k create rolebinding $dry --user=<user> --role=<role> <role-name>

## Check using the auth can-i command

k auth can-i create pods --as=<user>

## Running upgrades

<https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/>

**Upgrading the control plane:**

# Unhold  
apt-mark unhold kubeadm  
  
# Update apt  
apt-get update  
  
# Find packages  
apt show kubeadm  
  
# Install what is available  
apt-get install -y kubeadm=1.xx.0-00  
  
# Upgrade kubeadm  
kubeadm upgrade plan  
kubeadm apply v.1.xx.0  
  
# Drain the control plane  
k drain <control-plane> --ignore-daemonsets  
  
# Install kubelet and kubectl updates  
apt-get update && apt-get install -y kubelet=1.26.x-00 kubectl=1.26.x-00 && \  
apt-mark hold kubelet kubectl  
  
# Restart kubelet  
systemctl daemon-reload  
systemctl restart kubelet  
  
# Uncordon the node  
k uncordon <control-plane>

**Upgrading a worker node**

ssh worker-node  
  
# Note that there is a difference with this step  
kubeadm upgrade node  
  
# Drain the node  
k drain <node> --ignore-daemonsets  
  
# Update apt  
apt-get update  
apt-get install -y kubelet=1.xx.x-xx kubectl=1.xx.x-xx  
  
# Restart the kubelet  
systemctl daemon-reload  
systemctl restart kubelet  
  
# Uncordon the node  
k uncordon <node>

# Networking

## Working with the ip command

Find ip address of nodes:

# Get ip address through kubectl  
k get no -o wide  
  
ssh <node>  
  
# Find address and mac address of node  
ip a | grep -C 3 <ip-address>  
  
# Find network device  
ip link

<https://www.cyberciti.biz/faq/linux-ip-command-examples-usage-syntax/>

## Find status of network device

ip link show <device>  
  
# For example:  
ip link show cni0  
3: cni0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue state UP mode DEFAULT group default qlen 1000  
 link/ether 1a:f8:aa:77:8f:53 brd ff:ff:ff:ff:ff:ff

## Finding the IP address of the gateway out to the internet

ip route show default  
default via 172.25.0.1 dev eth1

## Find the port of kube scheduler

netstat -nplt can be useful to find out what ports are open <https://www.howtogeek.com/513003/how-to-use-netstat-on-linux/>

netstat -nplt | grep scheduler  
tcp 0 0 127.0.0.1:10259 0.0.0.0:\* LISTEN 3317/kube-scheduler

In this case, we see that the scheduler is open on port 10259

## Network policies

Network policies let you specify ingress and egress rules.

For example, this will only allow http traffic from anywhere into port 80.

<https://kubernetes.io/docs/concepts/services-networking/network-policies/>

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: test-network-policy  
 namespace: default  
spec:  
 podSelector:  
 matchLabels:  
 role: db # Lets you specify labels  
 run: pod-name # Depends on the label of your pod  
 policyTypes:  
 - Ingress  
 ingress:  
 - ports:  
 - protocol: TCP  
 port: 80

# Services

<https://kubernetes.io/docs/concepts/services-networking/service/>

## CoreDNS

<https://kubernetes.io/docs/concepts/services-networking/service/#dns>

## Using nslookup to validate the service is reachable from a pod

Get the service:

k describe svc web-service   
Name: web-service  
Namespace: default  
Labels: <none>  
Annotations: <none>  
Selector: label=value # Pods with this label will receive this service  
Type: ClusterIP  
IP Family Policy: SingleStack  
IP Families: IPv4  
IP: 10.99.70.136  
IPs: 10.99.70.136  
Port: <unset> 80/TCP  
TargetPort: 80/TCP  
Endpoints: 10.244.0.5:80  
Session Affinity: None  
Events: <none>

Exec/ run into a pod:

k exec -ti <pod>

Verify that you can look up the service with nslookup

nslookup web-service

Name: web-service  
Address 1: 10.99.70.136 web-service.default.svc.cluster.local

## Port vs target port

This is usually a confusing thing

Port: is the incoming port to the service

TargetPort: is the target port pointing to a deployment/pods that the service forwards connections to.

[**Note**](https://kubernetes.io/docs/concepts/services-networking/service/#defining-a-service)**:** A Service can map any incoming port to a targetPort. By default and for convenience, the targetPort is set to the same value as the port field.

For example:

apiVersion: v1  
kind: Service  
metadata:  
 name: mysql-service  
 namespace: beta  
spec:  
 ports:  
 - port: 3306  
 targetPort: 3306

# Common troubleshooting tips

Pods not scheduling? Check that pods in kube-system are running correctly.t

k get po -n kube-system

Would you like to know the metrics per node?

k top node

Metrics per pod

k top pod --containers=true

Kubelet not running? Restart it:

# Check that the config is correct  
cat /etc/systemd/system/kubelet.service.d/10-kubeadm-conf  
  
# Check the logs  
journalctl -u kubelet  
  
# Restart  
systemctl restart kubelet

Side note, what is systemd?<https://en.wikipedia.org/wiki/Systemd>,

Side-side note: <https://en.wikipedia.org/wiki/System_D>

“… a manner of responding to challenges that require one to have the ability to think quickly, to adapt, and to improvise when getting a job done.

[George Orwell](https://en.wikipedia.org/wiki/George_Orwell) described the term débrouillard as something the lowest-level kitchen workers, the [plongeur](https://en.wiktionary.org/wiki/plongeur#French)s, wanted to be called, indicating that they were people who would get the job done, no matter what.” 🤔

# Kublet issue

ssh node01

service kubelet status

sh node01

cat /var/log/syslog | grep kubelet

ssh node01

# maybe any interesting config files?

find / | grep kubeadm

eb 21 16:50:41 node01 kubelet[9516]: E0221 16:50:41.994746 9516 run.go:74] "command failed"

err="failed to parse kubelet flag: unknown flag: --improve-speed"

ssh node01

service kubelet status

We should check the logs:

grep kubelet /var/log/syslog

It shows us the error:

Feb 21 16:50:41 node01 kubelet[9516]: E0221 16:50:41.994746 9516 run.go:74] "command failed"

err="failed to parse kubelet flag: unknown flag: --improve-speed"

To fix we remove the unknown flag in /var/lib/kubelet/kubeadm-flags.env :

KUBELET\_KUBEADM\_ARGS="--container-runtime-endpoint=unix:///var/run/containerd/containerd.sock --pod-infra-container-image=registry.k8s.io/pause:3.9"

And the status should be good again:

service kubelet restart

service kubelet status

CHECK

In controlplane node, something problem with kubelet configuration files, fix that issue

You can ssh controlplane

location: /var/lib/kubelet/config.yaml and /etc/kubernetes/kubelet.conf

crictl ps -a | grep controller-manager

controlplane:~$ kubectl auth can-i get rs --as=system:serviceaccount:default:dev-sa

no

controlplane:~$ kubectl auth can-i get rs --as=system:serviceaccount:default:dev-sa

# cotroller manager

tip

* kubectl log
* /var/log/pods
* /var/log/containers
* crictl ps + crictl logs
* docker ps + docker logs (in case when Docker is used)
* kubelet logs: /var/log/syslog or journalctl

For your changes to apply you might have to:

1. move the kube-controller-manager.yaml out of the manifests directory
2. wait for the container to be gone (watch crictl ps )
3. move the manifest back in and wait for the container coming back up

kubectl -n kube-system get pod

We should check the logs:

kubectl -n kube-system logs kube-controller-manager-controlplane

It shows us the error:

Error: unknown flag: --project-sidecar-insertion

We could also check:

/var/log/pods/kube-system\_kube-controller-manager-controlplane\_\*/kube-controller-manager/\*.log

To fix we remove the unknown argument in /etc/kubernetes/manifests/kube-controller-manager.yaml :

apiVersion: v1

kind: Pod

metadata:

...

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=192.168.0.0/16

- --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

**- --project-sidecar-insertion=true # REMOVE**

- --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt

If we wait some time the *Pod* should start again, or we force restart via kubelet by moving the manifest out of the manifest directory:

cd /etc/kubernetes/manifests

mv kube-controller-manager.yaml ..

sleep 5

mv ../kube-controller-manager.yaml .

We need to wait till the *Pod* is Running and all containers are Ready.

# Api server crash not responsive

Log Locations

Solution

# always make a backup !

cp /etc/kubernetes/manifests/kube-apiserver.yaml ~/kube-apiserver.yaml.ori

# make the change

vim /etc/kubernetes/manifests/kube-apiserver.yaml

# wait till container restarts

watch crictl ps

# check for apiserver pod

k -n kube-system get pod

Apiserver is not coming back, we messed up!

# check pod logs

cat /var/log/pods/kube-system\_kube-apiserver-controlplane\_a3a455d471f833137588e71658e739da/kube-apiserver/X.log

> 2022-01-26T10:41:12.401641185Z stderr F Error: unknown flag: --this-is-very-wrong

og locations to check:

* /var/log/pods
* /var/log/containers
* crictl ps + crictl logs
* docker ps + docker logs (in case when Docker is used)
* kubelet logs: /var/log/syslog or journalctl

Now undo the change and continue

cp /etc/kubernetes/manifests/kube-apiserver.yaml ~/kube-apiserver.yaml.ori

# make the change

vim /etc/kubernetes/manifests/kube-apiserver.yaml

# wait till container restarts

watch crictl ps

# check for apiserver pod

k -n kube-system get pod

Apiserver is not coming back, we messed up!

# 1) if we would check the /var directory

cat /var/log/pods/kube-system\_kube-apiserver-controlplane\_e24b3821e9bdc47a91209bfb04056993/kube-apiserver/X.log

> Err: connection error: desc = "transport: Error while dialing dial tcp: address this-is-very-wrong: missing port in address". Reconnecting...

# 2) but here we want to find other ways, so we check the container logs

crictl ps # maybe run a few times, because the apiserver container get's restarted

crictl logs f669a6f3afda2

> Error while dialing dial tcp: address this-is-very-wrong: missing port in address. Reconnecting...

# 3) what about syslogs

journalctl | grep apiserver # nothing specific

cat /var/log/syslog | grep apiserver # nothing specific

Now undo the change and continue

# smart people use a backup

cp ~/kube-apiserver.yaml.ori /etc/kubernetes/manifests/kube-apiserv

# smart people use a backup

cp ~/kube-apiserver.yaml.ori /etc/kubernetes/manifests/kube-apiserver.yaml

NEXT

* Exam Desktop
* Editor
* Tab 1
* [**+**](https://killercoda.com/killer-shell-cka/scenario/apiserver-crash)

60 min

Check what the logs say.

Fix the Apiserver again.

Log Locations

Log locations to check:

* /var/log/pods
* /var/log/containers
* crictl ps + crictl logs
* docker ps + docker logs (in case when Docker is used)
* kubelet logs: /var/log/syslog or journalctl

Solution

BACKNEXT

* Exam Desktop
* Editor
* Tab 1
* [**+**](https://killercoda.com/killer-shell-cka/scenario/apiserver-crash)

45 min

# VIM

e look at important Vim settings if you like to work with YAML during the K8s exams.

### **Settings**

First create or open (if already exists) file .vimrc :

vim ~/.vimrc

Now enter (in insert-mode activated with i) the following lines:

set expandtab

set tabstop=2

set shiftwidth=2

Save and close the file by pressing Esc followed by :x and Enter.

### **Explanation**

Whenever you open Vim now as the current user, these settings will be used.

If you ssh onto a different server, these settings will **not** be transferred.

Settings explained:

expandtab: use spaces for tab

tabstop: amount of spaces used for tab

shiftwidth: amount of spaces used during indentation

cd /etc/kubernetes/manifests

mv kube-controller-manager.yaml ..

sleep 5

mv ../kube-controller-manager.yaml .

# kublet configuration and missconfig

tips

ssh node01

service kubelet status

ssh node01

cat /var/log/syslog | grep kubelet

ssh node01

# maybe any interesting config files?

find / | grep kubeadm

ssh node01

service kubelet status

We should check the logs:

grep kubelet /var/log/syslog

# logs from multicontainer

kubectl exec tester-0 -- curl tester.level-1000.svc.cluster.local

kubectl exec tester-0 -- curl tester.level-1001.svc.cluster.local

kubectl exec tester-0 -- curl tester.level-1002.svc.cluster.local

prority class   
apiVersion: v1

kind: Pod

metadata:

labels:

run: important

name: important

namespace: lion

spec:

**priorityClassName: level3**

containers:

- image: nginx:1.21.6-alpine

name: important

**resources:**

**requests:**

**memory: 1Gi**

dnsPolicy: ClusterFirst

restartPolicy: Always

We can also see logs about this procedure like: Preempted by lion/important on node controlplane

k get events -A --sort-by='{.metadata.creationTimestamp}'

----simulator  
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/cer

# yq e '.users[] | select(.name == "account-0027") | .user."client-certificate-data"' /opt/course/1/kubeconfig | base64 -d > /opt/course/1/cer

# If you don’t have yq, you can extract and decode using awk:

# bash

# Copy

# Edit

# awk '/name: account-0027/{f=1} f && /client-certificate-data:/{print $2; exit}' /opt/course/1/kubeconfig | base64 -d > /opt/course/1/cer

# ✅ Confirm it worked:

# bash

# Copy

# Edit

# file /opt/course/1/cer

# Output should say something like:

# bash

# Copy

# Edit

# /opt/course/1/cer: PEM certificate

k --kubeconfig /opt/course/1/kubeconfig config view --raw -ojsonpath="{.users[0].user.client-certificate-data}" | base64 -d > /opt/course/1/cert

https://cloudutsuk.com/posts/certification/cka/cka-pratice-test-1/

# Let me know if the kubeconfig uses a client-certificate file path instead of embedded data — that would require a different step.

Solve this question on: ssh cka7968 2

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

ℹ️ It is not required for MinIO to run properly. Installing the Helm Chart

project-h800

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

Solve this question on: ssh cka3962

4

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.  
  
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.

Solve this question on: ssh cka5774 5

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 cluste

api-gateway-hpa-staging.yaml

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: api-gateway

namespace: staging

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

🔁 Step 2: Create the HPA manifest for prod

yaml

Copy

Edit

# api-gateway-hpa-prod.yaml

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: api-gateway

namespace: prod

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: api-gateway

minReplicas: 2

maxReplicas: 6

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

✅ Step 3: Apply Both Manifests to the Cluster

bash

Copy

Edit

kubectl apply -f api-gateway-hpa-staging.yaml

kubectl apply -f api-gateway-hpa-prod.yaml

🔍 Verify

bash

Copy

Edit

kubectl get hpa -n staging

kubectl get hpa -n prod

You should see the api-gateway HPA configured in both namespaces with correct min/max replicas.

# Cli

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

6 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.

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.

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

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

ℹ️ You can connect to the worker node using ssh cka3962-node1 from cka3962

olve 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

ℹ️ You can connect to the worker node using ssh cka3962-node1 from cka3962

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.

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 Namesp

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 controlplane



>

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

⚑ Flag this

>

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

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

/opt/course/13/ingress.yaml

/opt/course/13/ingress.yaml

/opt/course/13/ingress.yaml



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

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:

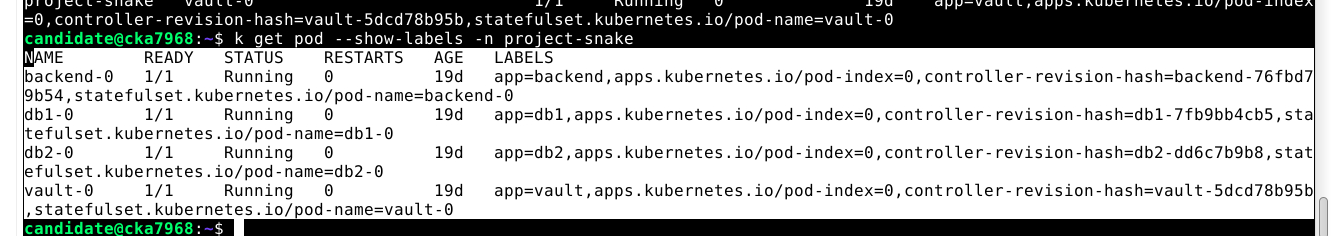
* Connect to db1-\* *Pods* on port 1111
* Connect to db2-\* *Pods* on port 2222

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

⚑ Flag this

> 

Solve this question on: ssh cka5774

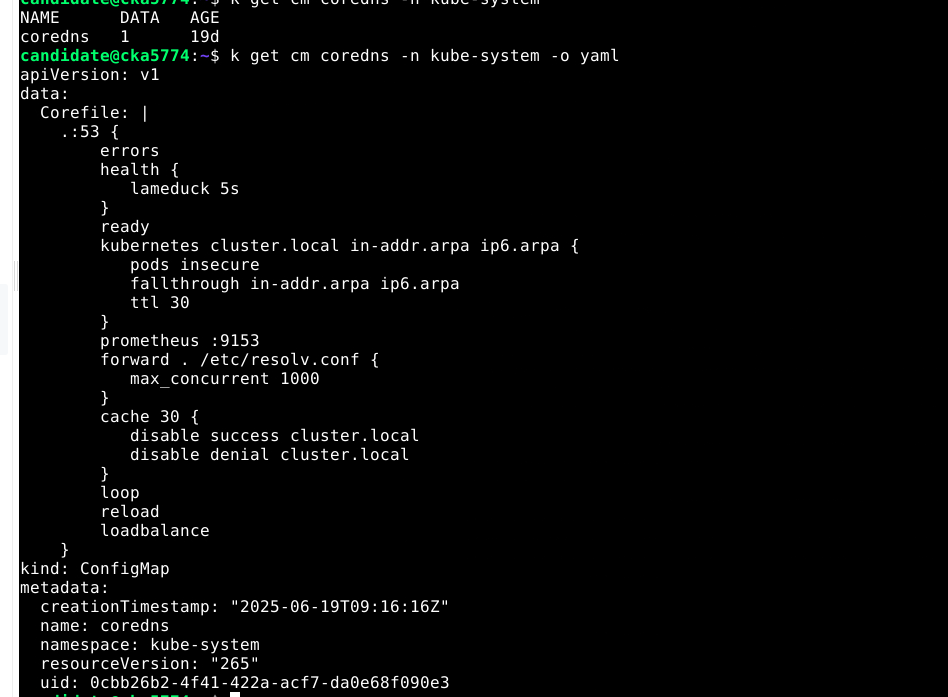
The CoreDNS configuration in the cluster needs to be updated:

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



n 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

First, find the Pod using:

bash

CopyEdit

crictl pods | grep tigers-reunite

This gives you the **Pod ID** (first column).

Then list containers in that pod:

bash

CopyEdit

crictl ps -p <POD-ID>

Note the **container ID** (first column).

### 5️⃣ Write the Container ID and Runtime Type

bash

CopyEdit

crictl inspect <CONTAINER-ID> | grep -E '"id":|"runtimeType"'

Example output:

json

CopyEdit

"id": "123456789abcdef",

"runtimeType": "io.containerd.runc.v2"

Now write this info to:

bash

CopyEdit

echo "123456789abcdef io.containerd.runc.v2" > /opt/course/17/pod-container.txt

(Use the actual container ID and runtimeType.)

### 6️⃣ Write Container Logs

bash

CopyEdit

crictl logs <CONTAINER-ID> > /opt/course/17/pod-container.log

## ✅ Final Summary

| **Task** | **Command Summary** |
| --- | --- |
| Create Pod | kubectl run with labels |
| Find Node | kubectl get pod -o wide |
| SSH to Node | ssh cka2556-nodeX |
| Get Container ID | crictl pods + crictl ps -p <POD-ID> |
| Write ID + Runtime | crictl inspect → /opt/course/17/pod-container.txt |
| Write logs | crictl logs → /opt/course/17/pod-container.log |

Let me know if you'd like a one-liner to extract ID + runtime or if the pod is stuck in Pending.

# Ask ChatGPT

# Crd

Resources from CRDs can be namespaced, so here we have to look for objects in all *Namespaces*.

k get crd > /root/crds

k get db-backups -A > /root/db-backups

# commands

## secret decoding

$ kubectl -n database-ns get secrets database-data -o jsonpath**=**'{.data.DB\_PASSWORD}' | base64 -d

secret

$ kubectl -n database-ns get secrets database-data -o jsonpath**=**'{.data.DB\_PASSWORD}' | base64 -d | tee decoded.txt

secret

## sorting top

* + Topsort pods

$ kubectl top pod -A --sort-by cpu --no-headers | head -n 1 | awk '{print $2","$1}'

kube-apiserver-controlplane,kube-system

$ kubectl top pod -A --sort-by cpu --no-headers | head -n 1 | awk '{print $2","$1}' | tee high\_cpu\_pod.txt

kube-apiserver-controlplane,kube-system

$ kubectl top pod -A --sort-by cpu

# Backup etc

etcd-controlplane pod is running in kube-system environment, take backup and store it in /opt/cluster\_backup.db file, and also store backup console output store it in backup.txt

ssh controlplane

$ kubectl -n kube-system get pod etcd-controlplane -o yaml

$ kubectl -n kube-system get pod etcd-controlplane -o yaml

*# 1. 进入 Pod，执行备份操作*

$ kubectl -n kube-system exec -it etcd-controlplane -- sh

*# 这里执行失败：一直卡着，没有进行备份。*

pod$ ETCDCTL\_API**=**3 etcdctl snapshot save /cluster\_backup.db \

**>** --endpoints**=**127.0.0.1:2379 \

*# 这里执行失败：一直卡着，没有进行备份。*

$ ETCDCTL\_API**=**3 etcdctl snapshot sav /opt/cluster\_backup.db \

**>** --endpoints**=**172.30.1.2:2379 \

*# 这里执行失败：一直卡着，没有进行备份。*

$ ETCDCTL\_API**=**3 etcdctl snapshot sav /opt/cluster\_backup.db \

**>** --endpoints**=**172.30.1.2:2379 \

kubectl get node -o yaml | grep NodeName

kubectl.exe top pod -A --sort-by cpu

kubectl delete -f pod.yaml --force --grace-period 0

helm

helm repo list

helm repo update

helm search repo nginx –versions

helm search repo nginx –versions

bitnami/nginx   18.2.0         1.27.1         NGINX Open Source is a web server that

helm show values bitnami/apache # will show a long list of all possible value-settings

helm show values bitnami/apache | yq

helm -n mercury install internal-issue-report-apache bitnami/apache \

--set replicaCount=2 \

--set image.debug=true

*Secrets* won't be created automatically for \*ServiceAccounts, but it's possible to create a *Secret* manually and attach it to a *ServiceAccount* by setting the correct annotation on the *Secret*. This was done for this task.



k -n neptune get sa # get overview

k -n neptune get secrets # shows all secrets of namespace

k -n neptune get secrets -oyaml | grep annotations -A 1 # shows secrets with first annotation

If a *Secret* belongs to a *ServiceAccount*, it'll have the annotation kubernetes.io/service-account.name. Here the *Secret* we're looking for is neptune-secret-1.



➜ k -n neptune get secret neptune-secret-1 -o yaml

apiVersion: v1

# Wazne error

k -n neptune describe pod api-new-c32-7d64747c87-zh648 | grep -i error

change pod into deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: holy-api        # name stays the same

namespace: pluto # important

spec:

replicas: 3 # 3 replicas

selector:

  matchLabels:

    id: holy-api # set the correct selector

template:

   # => from here down it's the same as the pods metadata: and spec: sections

  metadata:

    labels:

      id: holy-api

    name: holy-api

  - name: SECRET1\_USER # add

    valueFrom: # add

      secretKeyRef:

# create side car pod

- name: logs

      emptyDir: {}

    initContainers:

    - name: init

      image: bash:5.0.11

      command: ['bash', '-c', 'echo init > /var/log/cleaner/cleaner.log']

      volumeMounts:

      - name: logs

        mountPath: /var/log/cleaner

    - name: logger-con                                                # add

      image: busybox:1.31.0                                           # add

      restartPolicy: Always                                           # add

      command: ["sh", "-c", "tail -f /var/log/cleaner/cleaner.log"]   # add

      volumeMounts:                                                   # add

      - name: logs                                                    # add

        mountPath: /var/log/cleaner                                   # add

## Logging sidecar

- name: logs

      emptyDir: {}

    initContainers:

    - name: init

      image: bash:5.0.11

      command: ['bash', '-c', 'echo init > /var/log/cleaner/cleaner.log']

      volumeMounts:

      - name: logs

        mountPath: /var/log/cleaner

    - name: logger-con                                                # add

      image: busybox:1.31.0                                           # add

      restartPolicy: Always                                           # add

      command: ["sh", "-c", "tail -f /var/log/cleaner/cleaner.log"]   # add

      volumeMounts:                                                   # add

      - name: logs                                                    # add

        mountPath: /var/log/cleaner                                   # add

## InitContainer

Create an *InitContainer* named init-con which also mounts that volume and creates a file index.html with content check this out! in the root of the mounted volume. For this test we ignore that it doesn't contain valid html.

The *InitContainer* should be using image busybox:1.31.0. Test your implementation for example using curl from a temporary nginx:alpine *Pod*.

spec:

    volumes:

    - name: web-content

      emptyDir: {}

    initContainers: # initContainer start

    - name: init-con

      image: busybox:1.31.0

      command: ['sh', '-c', 'echo "check this out!" > /tmp/web-content/index.html']

      volumeMounts:

      - name: web-content

        mountPath: /tmp/web-content # initContainer end

    containers:

    - image: nginx:1.17.3-alpine

      name: nginx

      volumeMounts:

      - name: web-content

        mountPath: /usr/share/nginx/html

      ports:

      - containerPort: 80

# Service issues: # k -n mars edit service manager-api-svc

apiVersion: v1

kind: Service

metadata:

...

labels:

  app: manager-api-svc

name: manager-api-svc

namespace: mars

...

spec:

clusterIP: 10.3.244.121

ports:

- name: 4444-80

  port: 4444

  protocol: TCP

  targetPort: 80

selector:

   #id: manager-api-deployment # wrong selector, needs to point to pod!

  id: manager-api-pod

sessionAffinity: None

type: ClusterIP

networkpolicy

# 20\_np1.yaml

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: np1

namespace: venus

spec:

podSelector:

  matchLabels:

    id: frontend # label of the pods this policy should be applied on

policyTypes:

- Egress # we only want to control egress

egress:

- to: # 1st egress rule

  - podSelector: # allow egress only to pods with api label

      matchLabels:

        id: api

- ports: # 2nd egress rule

  - port: 53 # allow DNS UDP

    protocol: UDP

  - port: 53 # allow DNS TCP

    protocol: TCP

# label pod with specific label

k -n sun get pod -l type=runner # only pods with label runner

k label -h # help

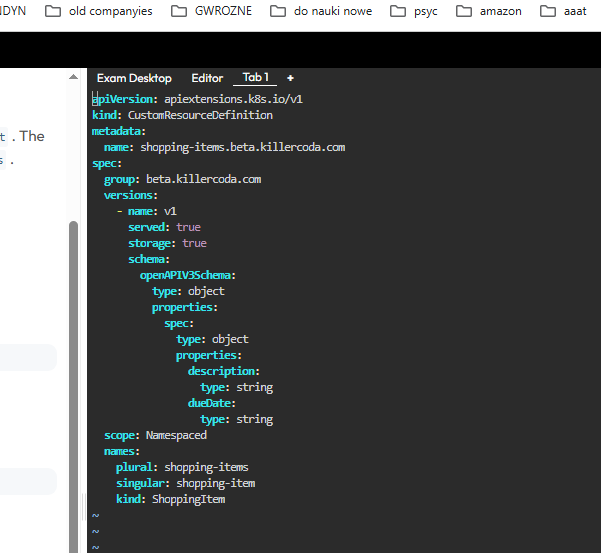
k -n sun label pod -l type=runner protected=true # run for label runner

k -n sun label pod -l type=worker protected=true # run for label worker

# crd

k get crd > /root/crds

k get db-backups -A > /root/db-backups

new crd   




k get shopping-item

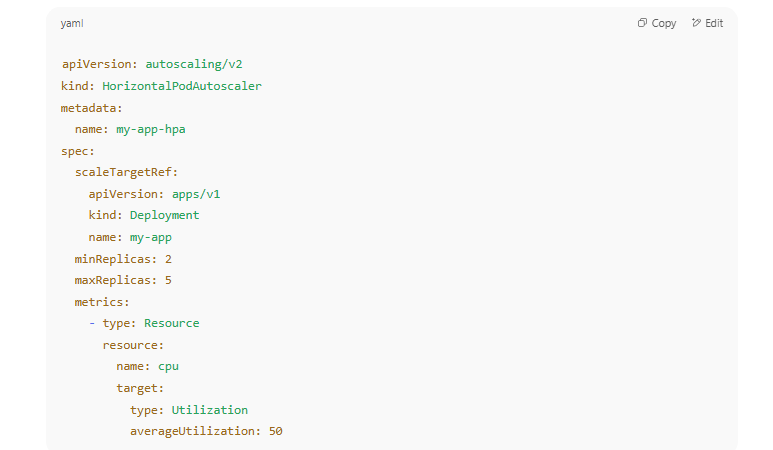
k get shopping-item bananas –oyaml

k delete crd shopping-items.beta.killercoda.com

# HPA

kubectl autoscale deployment redis-prod --cpu-percent=80 --min=5 --max=10

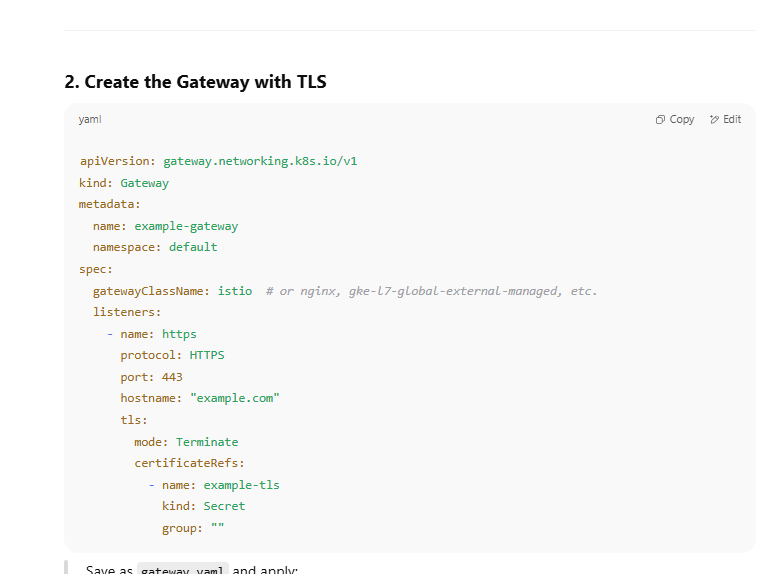
kubectl autoscale deployment my-app --cpu-percent=50 --min=2 --max=5



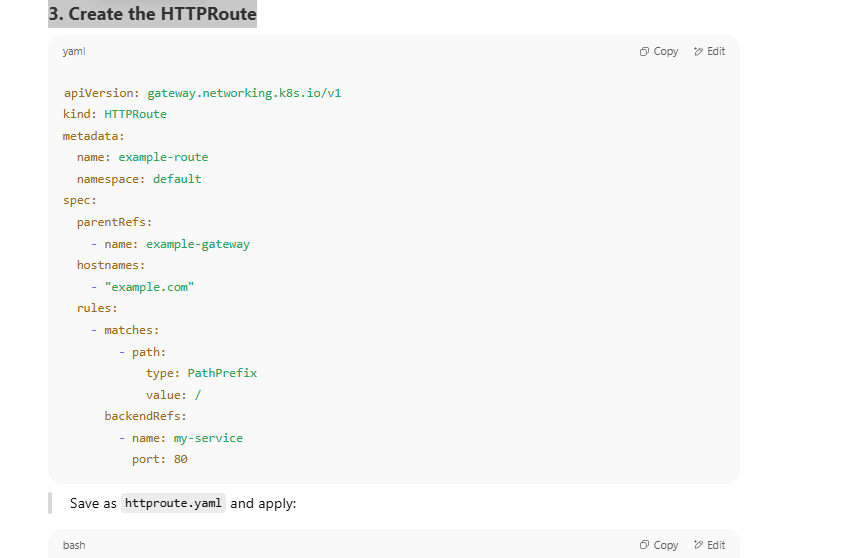
# Create a Gateway

with TLS and an HTTPRoute, matching the existing Ingress resource in the environment. Delete the Ingress after creating the Gateway.





## Create the HTTPRoute



kubectl get gateway

kubectl get httproute

kubectl describe gateway example-gateway

kubectl describe httproute example-route

Helm template helm repo add myrepo https://example.com/charts

helm repo update

✅ Step 2: Generate Helm template and save to file

bash

Copy

Edit

helm template myrelease myrepo/mychart \

--version 1.2.3 \

--namespace my-namespace \

--set image.tag=1.21.0,replicaCount=3 \

> mychart-rendered.yaml

📌 This command:

Renders the Helm template using version 1.2.3

Targets namespace my-namespace

# kubectl get priorityclass

Identify the user-defined one (not the system ones like system-cluster-critical or system-node-critical).

Example output:

text

Copy

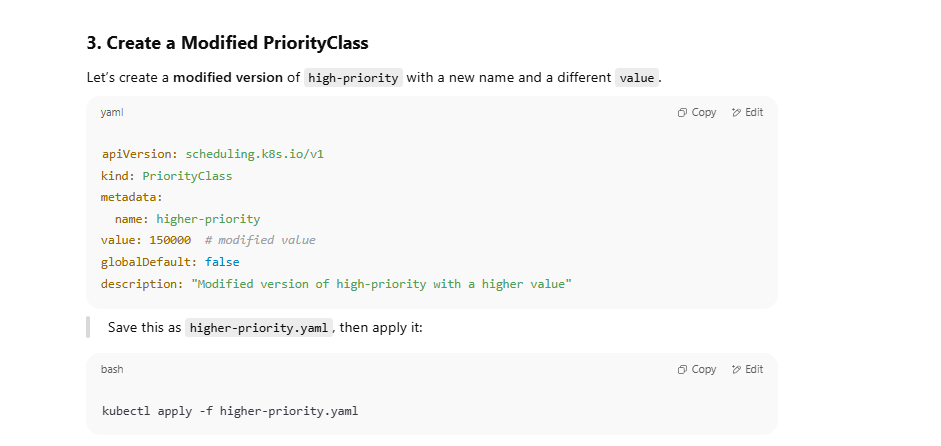
Edit

NAME VALUE GLOBALDEFAULT AGE

high-priority 100000 false 5d

1. Inspect the Existing PriorityClass

kubectl get priorityclass high-priority -o yaml



kubectl get priorityclass

You should now see both high-priority and higher-priority in the list.

kubectl exec -n ingress-nginx <nginx-pod-name> -- cat /etc/nginx/nginx.conf | grep ssl\_protocols

You should see:

nginx

Copy

Edit

ssl\_protocols TLSv1.2 TLSv1.3;

# api error

you're using systemd-based systems:

bash

CopyEdit

sudo systemctl status kube-apiserver

Or if running as a **static pod** (typical in kubeadm setups):

bash

CopyEdit

kubectl get pods -n kube-system

Look for kube-apiserver pod status (CrashLoopBackOff, Pending, etc.).

Check logs:

bash

CopyEdit

kubectl logs -n kube-system kube-apiserver-<node-name>

If not using kubectl (because API is down), check static pod logs directly:

bash

CopyEdit

sudo journalctl -u kubelet | grep kube-apiserver

Or:

bash

CopyEdit

docker logs kube-apiserver # If using Docker

### 🔍 2. **Common kube-apiserver Issues**

#### ✅ Check /etc/kubernetes/manifests/kube-apiserver.yaml

Look for common problems:

* Invalid flags
* Bad certificates
* Incorrect etcd address

Example working snippet:

yaml

CopyEdit

- --etcd-servers=https://127.0.0.1:2379

- --client-ca-file=/etc/kubernetes/pki/ca.crt

- --tls-cert-file=/etc/kubernetes/pki/apiserver.crt

- --tls-private-key-file=/etc/kubernetes/pki/apiserver.key

### 🔧 Fix if:

* Etcd endpoint is wrong or unreachable
* TLS certs missing or expired
* Misconfigured --advertise-address

Make changes and kubelet will automatically restart the static pod.

### 🔍 3. **Check kube-scheduler**

Check pod or service status:

bash

CopyEdit

kubectl get pods -n kube-system

kubectl logs -n kube-system kube-scheduler-<node-name>

Or on host (if kube-apiserver is down):

bash

CopyEdit

cat /etc/kubernetes/manifests/kube-scheduler.yaml

Verify:

* Correct --kubeconfig
* No conflicting --bind-address or --secure-port
* The scheduler is not trying to connect to a dead API server

### ✅ 4. **Check API Server Connectivity from Scheduler**

Try this manually:

bash

CopyEdit

curl --cacert /etc/kubernetes/pki/ca.crt https://127.0.0.1:6443/healthz

You should get ok.

If not, then the API server is not healthy or not listening correctly.

### 🔄 5. **Restart Services (if applicable)**

If using systemd instead of static pods:

bash

CopyEdit

sudo systemctl restart kube-apiserver

sudo systemctl restart kube-scheduler

Otherwise, kubelet will auto-restart static pods once manifest files are updated correctly.

### ✅ 6. **Verify Fixes**

Once corrected:

bash

CopyEdit

kubectl get componentstatuses

kubectl get nodes

kubectl get pods -n kube-system

You should see all components (scheduler, controller-manager, etcd) as Healthy.

## 🛠 Common Fix Summary

| **Issue** | **Fix** |
| --- | --- |
| Invalid TLS certs | Regenerate or reconfigure certs |
| Incorrect etcd endpoint | Update --etcd-servers in kube-apiserver.yaml |
| Corrupted kubeconfig | Regenerate kubeconfig or fix the path in manifest |
| Port conflict | Ensure 6443 (API) and 10259 (scheduler) are not in use |
| Misconfigured manifests | Edit YAML files in /etc/kubernetes/manifests |

Let me know if you want to share logs or configs, and I can help you analyze the root cause precisely.

## Affinity

kubectl -f /root/hobby.yaml delete --force --grace-period 0

apiVersion: v1

kind: Pod

metadata:

labels:

level: hobby

name: hobby-project

spec:

containers:

- image: nginx:alpine

name: c

**affinity:**

**podAffinity:**

**preferredDuringSchedulingIgnoredDuringExecution:**

**- weight: 100**

**podAffinityTerm:**

**labelSelector:**

**matchExpressions:**

**- key: level**

**operator: In**

**values:**

**- restricted**

**topologyKey: kubernetes.io/hostname**

# DS

apiVersion: apps/v1

kind: DaemonSet

metadata:

**name: configurator**

**namespace: configurator**

labels:

k8s-app: configurator

spec:

selector:

matchLabels:

name: configurator

template:

metadata:

labels:

name: configurator

spec:

containers:

- name: configurator

**image: bash**

command:

- sh

- -c

**- 'echo aba997ac-1c89-4d64 > /mount/config && sleep 1d'**

volumeMounts:

- name: vol

**mountPath: /mount**

volumes:

- name: vol

hostPath:

apiVersion: v1

kind: Pod

metadata:

labels:

level: hobby

name: hobby-project

spec:

containers:

- image: nginx:alpine

name: c

**affinity:**

**podAntiAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**- labelSelector:**

**matchExpressions:**

**- key: level**

**operator: In**

**values:**

**- restricted**

**topologyKey: kubernetes.io/hostname**

apiVersion: v1

kind: Pod

metadata:

labels:

level: hobby

name: hobby-project

spec:

containers:

- image: nginx:alpine

name: c

**affinity:**

**podAntiAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**- labelSelector:**

**matchExpressions:**

**- key: level**

**operator: In**

**values:**

**- restricted**

**topologyKey: kubernetes.io/hostname**

# install cluster

kubeadm init --kubernetes-version=1.33.3 --pod-network-cidr 192.168.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem

cp /etc/kubernetes/admin.conf /root/.kube/config

kubectl version

kubectl get pod -A

* Exam Desktop
* Editor
* Tab 1
* [**+**](https://killercoda.com/killer-shell-cka/scenario/cluster-setup)

# Join node

kubeadm token create --print-join-command

# generate a new token OR use the one printed out "kubeadm init" before

kubeadm token create --print-join-command

ssh node-summer

kubeadm join 172.30.1.2:6443 --token ...

exit

# cluster upgrade

# see current versions

kubectl get node

kubectl version

Tip 2

kubeadm upgrade -h

apt-cache show kubeadm

# show available versions

apt-cache show kubeadm

# can be different for you

apt-get install kubeadm=1.33.3-1.1

# could be a different version for you, it can also take a bit to finish!

kubeadm upgrade apply v1.33.3

Next we update kubectl and kubelet :

# can be a different version for you

apt-get install kubectl=1.33.3-1.1 kubelet=1.33.3-1.1

service kubelet restart

Upgrade Node node01 to the same version as controlplane .

Tip

ssh node01

kubeadm upgrade -h

Solution

ssh node01

# can be a different version for you

apt-get install kubeadm=1.33.3-1.1

kubeadm upgrade node

Next we update kubectl and kubelet :

Upgrade Node node01 to the same version as controlplane .

ssh node01

# can be a different version for you

apt-get install kubelet=1.33.3-1.1

service kubelet restart

# cert

kubeadm certs check-expiration

Using kubeadm, renew the certificates of the apiserver and scheduler.conf .

Static pod move

We scp the static *Pod* manifest to our local *Node*, which is already the controlplane in this case.

scp node01:/etc/kubernetes/manifests/resource-reserver.yaml .

ssh node01 -- rm /etc/kubernetes/manifests/resource-reserver.yaml

We should only see one custom static *Pod* with the correct changes:

kubectl get pod -A

# etc backup

something is not working at the moment on controlplane node(Cause NotReady state), check that and etcd-controlplane pod is running in kube-system environment, take backup and store it in /opt/cluster\_backup.db file, and also store backup

$ kubectl -n kube-system get pod

NAME READY STATUS RESTARTS AGE

calico-kube-controllers-94fb6bc47-rxh7x 1/1 Running 2 **(**5m15s ago**)** 13d

canal-phldr 2/2 Running 2 **(**5m15s ago**)** 13d

canal-zl4tq 2/2 Running 2 **(**5m15s ago**)** 13d

coredns-57888bfdc7-685jj 1/1 Running 1 **(**5m15s ago**)** 13d

coredns-57888bfdc7-bbwzr 1/1 Running 1 **(**5m15s ago**)** 13d

etcd-controlplane 1/1 Running 2 **(**5m15s ago**)** 13d

kube-apiserver-controlplane 1/1 Running 2 **(**5m15s ago**)** 13d

kube-controller-manager-controlplane 1/1 Running 2 **(**5m15s ago**)** 13d

kube-proxy-2mfwz 1/1 Running 2 **(**5m15s ago**)** 13d

kube-proxy-z2ps8 1/1 Running 1 **(**5m15s ago**)** 13d

kube-scheduler-controlplane 1/1 Running 2 **(**5m15s ago**)** 13d

$ kubectl exec -it etcd-controlplane -- sh

Error from server **(**NotFound**)**: pods "etcd-controlplane" not found

kubectl -n kube-system exec -it etcd-controlplane -- sh

error: Internal error occurred: error sending request: Post "https://172.30.1.2:10250/exec/kube-system/etcd-controlplane/etcd?command=sh&input=1&output=1&tty=1": dial tcp 172.30.1.2:10250: connect: connection refused

$ ETCDCTL\_API**=**3 etcdctl --endpoints**=**https://127.0.0.1:2379 endpoint status --write-out**=**table --cert**=**/etc/kubernetes/pki/etcd/server.crt --key**=**/etc/kubernetes/pki/etcd/server.key --cacert**=**/etc/kubernetes/pki/etcd/ca.crt

+------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+

| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |

+------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+

| https://127.0.0.1:2379 | 264d7b068180479b | 3.5.15 | 6.2 MB | true | false | 5 | 2889 | 2889 | |

+------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+

kubectl get nodes

$ sudo systemctl status kubelet.service

$ cat /etc/kubernetes/kubelet.conf

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https://172.30.1.2:6443

name: kubernetes

contexts:

- context:

cluster: kubernetes

user: system:node:controlplane

name: system:node:controlplane@kubernetes

current-context: system:node:controlplane@kubernetes

kind: Config

preferences: {}

users:

- name: system:node:controlplane

user:

client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem

client-key: /var/lib/kubelet/pki/kubelet-client-current.pem

$ cat /var/lib/kubelet/config.yaml

apiVersion: kubelet.config.k8s.io/v1beta1

kubectl get pods -n kube-system | grep kube-apiserver

## 9. Troubleshooting - Controller Manager Issue

[**Troubleshooting - Controller Manager Issue**](https://killercoda.com/sachin/course/CKA/controller-manager-issue)

video-app deployment replicas 0. fix this issue

expected: 2 replicas

*# @author k* get deployments.apps video-app -o wide

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

video-app 0/2 0 0 94s redis redis:7.2.1 app**=**video-app

$ kubectl describe deployments.apps video-app

Name: video-app

Namespace: default

CreationTimestamp: Wed, 12 Feb 2025 13:32:03 +0000

Labels: app**=**video-app

Annotations: <none>

Selector: app**=**video-app

Replicas: 2 desired | 0 updated | 0 total | 0 available | 0 unavailable

StrategyType: RollingUpdate

MinReadySeconds: 0

RollingUpdateStrategy: 25% max unavailable, 25% max surge

Pod Template:

Labels: app**=**video-app

Containers:

redis:

Image: redis:7.2.1

Port: <none>

Host Port: <none>

Environment: <none>

Mounts: <none>

Volumes: <none>

Node-Selectors: <none>

Tolerations: <none>

Events: <none>

$ kubectl get deployments.apps video-app -o yaml

apiVersion: apps/v1

kind: Deployment

metadata:

creationTimestamp: "2025-02-12T13:32:03Z"

generation: 1

labels:

app: video-app

name: video-app

namespace: default

resourceVersion: "2031"

uid: 6bb0d745-f901-4f19-abc8-52007c724c77

spec:

progressDeadlineSeconds: 600

replicas: 2

revisionHistoryLimit: 10

selector:

matchLabels:

app: video-app

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

creationTimestamp: null

labels:

app: video-app

spec:

containers:

- image: redis:7.2.1

imagePullPolicy: IfNotPresent

name: redis

resources: **{}**

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

dnsPolicy: ClusterFirst

restartPolicy: Always

schedulerName: default-scheduler

securityContext: **{}**

terminationGracePeriodSeconds: 30

status: **{}**

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

controlplane Ready control-plane 20h v1.31.0

node01 Ready <none> 20h v1.31.0

$ kubectl get nodes -o wide

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

controlplane Ready control-plane 20h v1.31.0 172.30.1.2 <none> Ubuntu 20.04.5 LTS 5.4.0-131-generic containerd://1.7.22

node01 Ready <none> 20h v1.31.0 172.30.2.2 <none> Ubuntu 20.04.5 LTS 5.4.0-131-generic containerd://1.7.22

$ kubectl describe nodes | grep Taints

Taints: node-role.kubernetes.io/control-plane:NoSchedule

Taints: <none>

$ kubectl get componentstatuses

Warning: v1 ComponentStatus is deprecated **in** v1.19+

NAME STATUS MESSAGE ERROR

controller-manager Unhealthy Get "https://127.0.0.1:10257/healthz": dial tcp 127.0.0.1:10257: connect: connection refused

scheduler Healthy ok

etcd-0 Healthy ok

$ kubectl get pods -n kube-system | grep kube-controller-manager

kube-controller-manager-controlplane 0/1 CrashLoopBackOff 7 **(**2m22s ago**)** 13m

$ kubectl -n kube-system describe pod kube-controller-manager-controlplane

Name: kube-controller-manager-controlplane

Namespace: kube-system

Priority: 2000001000

Priority Class Name: system-node-critical

Node: controlplane/172.30.1.2

Start Time: Wed, 12 Feb 2025 13:31:16 +0000

Labels: component**=**kube-controller-manager

tier**=**control-plane

Annotations: kubernetes.io/config.hash: b0c098fd6896ecf1d8a30f03b739da5f

kubernetes.io/config.mirror: b0c098fd6896ecf1d8a30f03b739da5f

kubernetes.io/config.seen: 2025-02-12T13:32:02.867060432Z

kubernetes.io/config.source: file

Status: Running

SeccompProfile: RuntimeDefault

IP: 172.30.1.2

IPs:

IP: 172.30.1.2

Controlled By: Node/controlplane

Containers:

kube-controller-manager:

Container ID: containerd://a5ff61b46d26608c4f4c301161cde11aaf1eb615abfcb57fcd441a6c3ead7b35

Image: registry.k8s.io/kube-controller-manager:v1.31.0

Image ID: registry.k8s.io/kube-controller-manager@sha256:f6f3c33dda209e8434b83dacf5244c03b59b0018d93325ff21296a142b68497d

Port: <none>

Host Port: <none>

Command:

kube-controller-manegaar

--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**=**192.168.0.0/16

--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

--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**=**10.96.0.0/12

--use-service-account-credentials**=**true

State: Waiting

Reason: CrashLoopBackOff

Last State: Terminated

Reason: StartError

Message: failed to create containerd task: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "kube-controller-manegaar": executable file not found **in** $PATH: unknown

Exit Code: 128

Started: Thu, 01 Jan 1970 00:00:00 +0000

Finished: Wed, 12 Feb 2025 13:43:09 +0000

Ready: False

Restart Count: 7

Requests:

cpu: 25m

Liveness: http-get https://127.0.0.1:10257/healthz delay**=**10s timeout**=**15s period**=**10s *#success=1 #failure=8*

Startup: http-get https://127.0.0.1:10257/healthz delay**=**10s timeout**=**15s period**=**10s *#success=1 #failure=24*

Environment: <none>

Mounts:

/etc/ca-certificates from etc-ca-certificates **(**ro**)**

/etc/kubernetes/controller-manager.conf from kubeconfig **(**ro**)**

/etc/kubernetes/pki from k8s-certs **(**ro**)**

/etc/ssl/certs from ca-certs **(**ro**)**

/usr/libexec/kubernetes/kubelet-plugins/volume/exec from flexvolume-dir **(**rw**)**

/usr/local/share/ca-certificates from usr-local-share-ca-certificates **(**ro**)**

/usr/share/ca-certificates from usr-share-ca-certificates **(**ro**)**

Conditions:

Type Status

PodReadyToStartContainers True

Initialized True

Ready False

ContainersReady False

PodScheduled True

Volumes:

ca-certs:

Type: HostPath **(**bare host directory volume**)**

Path: /etc/ssl/certs

HostPathType: DirectoryOrCreate

etc-ca-certificates:

Type: HostPath **(**bare host directory volume**)**

Path: /etc/ca-certificates

HostPathType: DirectoryOrCreate

flexvolume-dir:

Type: HostPath **(**bare host directory volume**)**

Path: /usr/libexec/kubernetes/kubelet-plugins/volume/exec

HostPathType: DirectoryOrCreate

k8s-certs:

Type: HostPath **(**bare host directory volume**)**

Path: /etc/kubernetes/pki

HostPathType: DirectoryOrCreate

kubeconfig:

Type: HostPath **(**bare host directory volume**)**

Path: /etc/kubernetes/controller-manager.conf

HostPathType: FileOrCreate

usr-local-share-ca-certificates:

Type: HostPath **(**bare host directory volume**)**

Path: /usr/local/share/ca-certificates

HostPathType: DirectoryOrCreate

usr-share-ca-certificates:

Type: HostPath **(**bare host directory volume**)**

Path: /usr/share/ca-certificates

HostPathType: DirectoryOrCreate

QoS Class: Burstable

Node-Selectors: <none>

Tolerations: :NoExecute op**=**Exists

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal Pulled 12m **(**x4 over 13m**)** kubelet Container image "registry.k8s.io/kube-controller-manager:v1.31.0" already present on machine

Normal Created 12m **(**x4 over 13m**)** kubelet Created container kube-controller-manager

Warning Failed 12m **(**x4 over 13m**)** kubelet Error: failed to create containerd task: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "kube-controller-manegaar": executable file not found **in** $PATH: unknown

Warning BackOff 3m36s **(**x56 over 13m**)** kubelet Back-off restarting failed container kube-controller-manager **in** pod kube-controller-manager-controlplane\_kube-system**(**b0c098fd6896ecf1d8a30f03b739da5f**)**

*# 从这里可以看出：大概率是由于命令写错了。*

$ cat /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-manegaar

- --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**=**192.168.0.0/16

- --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

- --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**=**10.96.0.0/12

- --use-service-account-credentials**=**true

image: registry.k8s.io/kube-controller-manager:v1.31.0

imagePullPolicy: IfNotPresent

livenessProbe:

failureThreshold: 8

httpGet:

host: 127.0.0.1

path: /healthz

port: 10257

scheme: HTTPS

initialDelaySeconds: 10

periodSeconds: 10

timeoutSeconds: 15

name: kube-controller-manager

resources:

requests:

cpu: 25m

startupProbe:

failureThreshold: 24

httpGet:

host: 127.0.0.1

path: /healthz

port: 10257

scheme: HTTPS

initialDelaySeconds: 10

periodSeconds: 10

timeoutSeconds: 15

volumeMounts:

- mountPath: /etc/ssl/certs

name: ca-certs

readOnly: true

- mountPath: /etc/ca-certificates

name: etc-ca-certificates

readOnly: true

- mountPath: /usr/libexec/kubernetes/kubelet-plugins/volume/exec

name: flexvolume-dir

- mountPath: /etc/kubernetes/pki

name: k8s-certs

readOnly: true

- mountPath: /etc/kubernetes/controller-manager.conf

name: kubeconfig

readOnly: true

- mountPath: /usr/local/share/ca-certificates

name: usr-local-share-ca-certificates

readOnly: true

- mountPath: /usr/share/ca-certificates

name: usr-share-ca-certificates

readOnly: true

hostNetwork: true

priority: 2000001000

priorityClassName: system-node-critical

securityContext:

seccompProfile:

type: RuntimeDefault

volumes:

- hostPath:

path: /etc/ssl/certs

type: DirectoryOrCreate

name: ca-certs

- hostPath:

path: /etc/ca-certificates

type: DirectoryOrCreate

name: etc-ca-certificates

- hostPath:

path: /usr/libexec/kubernetes/kubelet-plugins/volume/exec

type: DirectoryOrCreate

name: flexvolume-dir

- hostPath:

path: /etc/kubernetes/pki

type: DirectoryOrCreate

name: k8s-certs

- hostPath:

path: /etc/kubernetes/controller-manager.conf

type: FileOrCreate

name: kubeconfig

- hostPath:

path: /usr/local/share/ca-certificates

type: DirectoryOrCreate

name: usr-local-share-ca-certificates

- hostPath:

path: /usr/share/ca-certificates

type: DirectoryOrCreate

name: usr-share-ca-certificates

status: **{}**

$ vim /etc/kubernetes/manifests/kube-controller-manager.yaml

*# kube-controller-manegaar kube-controller-manager*

$ kubectl -n kube-system get pod kube-controller-manager-controlplane

NAME READY STATUS RESTARTS AGE

kube-controller-manager-controlplane 1/1 Running 0 83s

$ kubectl get deployments.apps video-app

NAME READY UP-TO-DATE AVAILABLE AGE

video-app 2/2 2 2 22m

$ kubectl get pod

NAME READY STATUS RESTARTS AGE

video-app-7f4f8696cd-rl7ph 1/1 Running 0 2m18s

video-app-7f4f8696cd-vdfqw 1/1 Running 0 2m18s

$ kubectl get pods -l app**=**video-app

NAME READY STATUS RESTARTS AGE

video-app-7f4f8696cd-rl7ph 1/1 Running 0 2m39s

video-app-7f4f8696cd-vdfqw 1/1 Running 0 2m39s