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

- Score
- Questions and Answers
- Exam Tips

CKA Simulator B Kubernetes 1.33

https://killer.sh

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 | DNS / FQDN / Headless Service

Solve this question on: `ssh cka6016`

The *Deployment* `controller` in *Namespace* `lima-control` communicates with various cluster internal endpoints by using their DNS FQDN values.

Update the *ConfigMap* used by the *Deployment* with the correct FQDN values for:

1. `DNS_1`: *Service* `kubernetes` in *Namespace* `default`
2. `DNS_2`: Headless *Service* `department` in *Namespace* `lima-workload`
3. `DNS_3`: *Pod* `section100` in *Namespace* `lima-workload`. It should work even if the *Pod* IP changes
4. `DNS_4`: A *Pod* with IP `1.2.3.4` in *Namespace* `kube-system`

Ensure the *Deployment* works with the updated values.

ℹ️ You can use `nslookup` inside a *Pod* of the `controller` *Deployment*

Answer:

For this question we need to understand how cluster internal DNS works in Kubernetes. The most common use is `SERVICE.NAMESPACE.svc.cluster.local` which will resolve to the IP address of the Kubernetes *Service*. Note that we're asked to specify the FQDNs here so short values like `SERVICE.NAMESPACE` are not possible even if they would work.

Let's exec into the *Pod* for testing:

```
➜ ssh cka6016

➜ candidate@cka6016:~$ k -n lima-control get pod
NAME                          READY   STATUS    RESTARTS   AGE
controller-586d6657-gdmch   1/1     Running   0          11m
controller-586d6657-lvdtd   1/1     Running   0          11m

➜ candidate@cka6016:~$ k -n lima-control exec -it controller-586d6657-gdmch -- sh

➜ / # nslookup google.com
Server:         10.96.0.10
Address:        10.96.0.10:53

Non-authoritative answer:
Name:   google.com
Address: 142.250.185.238

Non-authoritative answer:
Name:   google.com
Address: 2a00:1450:4001:82f::200e

➜ / # nslookup non-exist.some.google.com
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find non-exist.some.google.com: NXDOMAIN

** server can't find non-exist.some.google.com: NXDOMAIN
```

We can perform DNS queries using `nslookup` and see if they resolve into an IP address.

Step 1

By default there is the `kubernetes` *Service* in `default` *Namespace* which can be used to access the K8s Api:

```
➜ / # 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
```

And we already have the value for `DNS_1` which is `kubernetes.default.svc.cluster.local`, that was the easy one.

Step 2

The next one is similar:

```
➜ / # nslookup department.lima-workload.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   department.lima-workload.svc.cluster.local
Address: 10.32.0.2
Name:   department.lima-workload.svc.cluster.local
Address: 10.32.0.9
```

The value for `DNS_2` is `department.lima-workload.svc.cluster.local`. It is the same structure as before but what's interesting here is that we get two IP addresses. These are the IP addresses of the *Pods* behind that *Service*.

This is the case because the *Service* is headless and doesn't have its own IP address, but it still has *Endpoints* and points properly to *Pods*:

```
➜ candidate@cka6016:~$ k -n lima-workload get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
department   ClusterIP   None           <none>        80/TCP    2m19s
section      ClusterIP   10.99.121.17   <none>        80/TCP    2m18s

➜ candidate@cka6016:~$ k -n lima-workload get ep
NAME         ENDPOINTS                     AGE
department   10.32.0.2:80,10.32.0.9:80     2m21s
section      10.32.0.10:80,10.32.0.3:80    2m21s
```

This means the decision which *Pod* IP to contact is now in the hands of the application which performed the DNS query of the headless *Service*.

Step 3

Now things start to get spicy, because we can do this:

```
➜ / # nslookup section100.section.lima-workload.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   section100.section.lima-workload.svc.cluster.local
Address: 10.32.0.10

➜ / # nslookup section200.section.lima-workload.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   section200.section.lima-workload.svc.cluster.local
Address: 10.32.0.3
```

Hence the value for `DNS_3` is `section100.section.lima-workload.svc.cluster.local`.

But this is **only possible** because the *Pods* behind the *Service* specify hostname and subdomain like this:

```
# kubectl -n lima-workload edit pod section100
apiVersion: v1
kind: Pod
metadata:
  name: section100
  namespace: lima-workload
  labels:
```

```
    name: section
spec:
  hostname: section100  # set hostname
  subdomain: section    # set subdomain to same name as service
  containers:
    - image: httpd:2-alpine
      name: pod
...
```

Step 4

It's possible to resolve a FQDN like `IP.NAMESPACE.pod.cluster.local` into an IP address:

```
→ / # nslookup 1-2-3-4.kube-system.pod.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   1-2-3-4.kube-system.pod.cluster.local
Address: 1.2.3.4
```

This is possible even without a *Pod* having to exist with that IP address in that *Namespace*.

We set `DNS_4` to `1-2-3-4.kube-system.pod.cluster.local`.

Solution

We should update the *ConfigMap*:

```
→ candidate@cka6016:~$ $ k -n lima-control get cm
NAME             DATA   AGE
control-config   4      10m

→ candidate@cka6016:~$ k -n lima-control edit cm control-config
```

```
apiVersion: v1
data:
  DNS_1: kubernetes.default.svc.cluster.local                 # UPDATE
  DNS_2: department.lima-workload.svc.cluster.local           # UPDATE
  DNS_3: section100.section.lima-workload.svc.cluster.local   # UPDATE
  DNS_4: 1-2-3-4.kube-system.pod.cluster.local                # UPDATE
kind: ConfigMap
metadata:
  name: control-config
  namespace: lima-control
...
```

And restart the *Deployment*:

```
→ candidate@cka6016:~$ kubectl -n lima-control rollout restart deploy controller
deployment.apps/controller restarted
```

And the *Pod* logs also look happy now:

```
→ candidate@cka6016:~$ k -n lima-control logs -f controller-54b5b69d7d-mgng2

+ 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


+ nslookup department.lima-workload.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   department.lima-workload.svc.cluster.local
Address: 10.32.0.2
Name:   department.lima-workload.svc.cluster.local
Address: 10.32.0.9


+ nslookup section100.section.lima-workload.svc.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   section100.section.lima-workload.svc.cluster.local
Address: 10.32.0.10


+ nslookup 1-2-3-4.kube-system.pod.cluster.local
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   1-2-3-4.kube-system.pod.cluster.local
Address: 1.2.3.4
```

Question 2 | Create a Static Pod and Service

Solve this question on: `ssh cka2560`

Create a `Static Pod` named `my-static-pod` in *Namespace* `default` on the controlplane node. It should be of image `nginx:1-alpine` and have resource requests for `10m` CPU and `20Mi` memory.

Create a NodePort *Service* named `static-pod-service` which exposes that static *Pod* on port `80`.

> ℹ️ For verification check if the new *Service* has one *Endpoint*. It should also be possible to access the *Pod* via the `cka2560` internal IP address, like using `curl 192.168.100.31:NODE_PORT`

Answer:

```
➜ ssh cka2560

➜ candidate@cka2560:~$ sudo -i

➜ root@cka2560:~# cd /etc/kubernetes/manifests/

➜ root@cka2560:~# k run my-static-pod --image=nginx:1-alpine -o yaml --dry-run=client > my-static-pod.yaml
```

Then edit the `my-static-pod.yaml` to add the requested resource requests:

```
# cka2560:/etc/kubernetes/manifests/my-static-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: my-static-pod
  name: my-static-pod
spec:
  containers:
  - image: nginx:1-alpine
    name: my-static-pod
    resources:
      requests:
        cpu: 10m
        memory: 20Mi
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

And make sure it's running:

```
➜ root@cka2560:~# k get pod -A | grep my-static
default       my-static-pod-cka2560              1/1     Running   0             20s
```

Now we expose that static *Pod*:

```
➜ root@cka2560:~# k expose pod my-static-pod-cka2560 --name static-pod-service --type=NodePort --port 80
```

This will generate a *Service* yaml like:

```
# kubectl expose pod my-static-pod-cka2560 --name static-pod-service --type=NodePort --port 80
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: my-static-pod
  name: static-pod-service
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: my-static-pod
  type: NodePort
status:
```

```
  loadBalancer: {}
```

Then we check the *Service* and *Endpoints*:

```
➜ root@cka2560:~# k get svc,ep -l run=my-static-pod
NAME                           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/static-pod-service   NodePort   10.98.249.240   <none>        80:32699/TCP   34s

NAME                             ENDPOINTS      AGE
endpoints/static-pod-service   10.32.0.4:80   34s
```

Also we should be able to access that Nginx container, your NodePort might be different than the one used here:

```
➜ root@cka2560:~# k get node -owide
NAME            STATUS   ROLES           AGE   VERSION   INTERNAL-IP      ...
cka2560         Ready    control-plane   8d    v1.33.1   192.168.100.31   ...

➜ root@cka2560:~# curl 192.168.100.31:32699
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Question 3 | Kubelet client/server cert info

Solve this question on: `ssh cka5248`

Node `cka5248-node1` has been added to the cluster using `kubeadm` and TLS bootstrapping.

Find the `Issuer` and `Extended Key Usage` values on `cka5248-node1` for:

1. Kubelet Client Certificate, the one used for outgoing connections to the kube-apiserver
2. Kubelet Server Certificate, the one used for incoming connections from the kube-apiserver

Write the information into file `/opt/course/3/certificate-info.txt`.

ℹ️ You can connect to the worker node using `ssh cka5248-node1` from `cka5248`

Answer:

First we check the kubelet client certificate:

```
➜ ssh cka5248

➜ candidate@cka5248:~$ ssh cka5248-node1

➜ candidate@cka5248-node1:~$ sudo -i

➜ root@cka5248-node1:~# find /var/lib/kubelet/pki
/var/lib/kubelet/pki
/var/lib/kubelet/pki/kubelet-client-2024-10-29-14-24-14.pem
/var/lib/kubelet/pki/kubelet.crt
/var/lib/kubelet/pki/kubelet.key
/var/lib/kubelet/pki/kubelet-client-current.pem

➜ root@cka5248-node1:~# openssl x509 -noout -text -in /var/lib/kubelet/pki/kubelet-client-current.pem | grep
Issuer
        Issuer: CN = kubernetes

➜ root@cka5248-node1:~# openssl x509 -noout -text -in /var/lib/kubelet/pki/kubelet-client-current.pem | grep
"Extended Key Usage" -A1
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
```

Next we check the kubelet server certificate:

```
➜ root@cka5248-node1:~# openssl x509 -noout -text -in /var/lib/kubelet/pki/kubelet.crt | grep Issuer
        Issuer: CN = cka5248-node1-ca@1730211854

➜ root@cka5248-node1:~# openssl x509 -noout -text -in /var/lib/kubelet/pki/kubelet.crt | grep "Extended Key
Usage" -A1
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
```

We see that the server certificate was generated on the worker node itself and the client certificate was issued by the Kubernetes api. The `Extended Key Usage` also shows if it's for client or server authentication.

The solution file should look something like this:

```
# cka5248:/opt/course/3/certificate-info.txt
Issuer: CN = kubernetes
X509v3 Extended Key Usage: TLS Web Client Authentication
Issuer: CN = cka5248-node1-ca@1730211854
X509v3 Extended Key Usage: TLS Web Server Authentication
```

Question 4 | Pod Ready if Service is reachable

Solve this question on: `ssh cka3200`

Do the following in *Namespace* `default`:

- Create a *Pod* named `ready-if-service-ready` of image `nginx:1-alpine`
- Configure a LivenessProbe which simply executes command `true`
- Configure a ReadinessProbe which does check if the url `http://service-am-i-ready:80` is reachable, you can use `wget -T2 -O- http://service-am-i-ready:80` for this
- Start the *Pod* and confirm it isn't ready because of the ReadinessProbe.

Then:

- Create a second *Pod* named `am-i-ready` of image `nginx:1-alpine` with label `id: cross-server-ready`
- The already existing *Service* `service-am-i-ready` should now have that second *Pod* as endpoint
- Now the first *Pod* should be in ready state, check that

Answer:

It's a bit of an anti-pattern for one *Pod* to check another *Pod* for being ready using probes, hence the normally available `readinessProbe.httpGet` doesn't work for absolute remote urls. Still the workaround requested in this task should show how probes and *Pod*<->*Service* communication works.

First we create the first *Pod*:

```
➜ ssh cka3200

➜ candidate@cka3200:~$ k run ready-if-service-ready --image=nginx:1-alpine --dry-run=client -o yaml > 4_pod1.yaml
```

Next perform the necessary additions manually:

```
# cka3200:/home/candidate/4_pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: ready-if-service-ready
  name: ready-if-service-ready
spec:
  containers:
  - image: nginx:1-alpine
    name: ready-if-service-ready
    resources: {}
    livenessProbe:                                  # add from here
      exec:
        command:
        - 'true'
    readinessProbe:
      exec:
        command:
```

```
        - sh
        - -c
        - 'wget -T2 -O- http://service-am-i-ready:80'   # to here
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Then create the *Pod* and confirm it's in a non-ready state:

```
➜ candidate@cka3200:~$ candidate@cka3200:~$ k -f 4_pod1.yaml create
pod/ready-if-service-ready created

➜ candidate@cka3200:~$ k get pod ready-if-service-ready
NAME                    READY   STATUS    RESTARTS   AGE
ready-if-service-ready   0/1     Running   0          8s
```

We can also check the reason for this using describe:

```
➜ candidate@cka3200:~$ k describe pod ready-if-service-ready
...
  Warning  Unhealthy  7s (x4 over 23s)  kubelet            Readiness probe failed: command timed out: "sh -c
wget -T2 -O- http://service-am-i-ready:80" timed out after 1s
```

Now we create the second *Pod*:

```
➜ candidate@cka3200:~$ k run am-i-ready --image=nginx:1-alpine --labels="id=cross-server-ready"
pod/am-i-ready created
```

The already existing *Service* `service-am-i-ready` should now have an *Endpoint*:

```
➜ candidate@cka3200:~$ k describe svc service-am-i-ready
Name:                     service-am-i-ready
Namespace:                default
Labels:                   id=cross-server-ready
Annotations:              <none>
Selector:                 id=cross-server-ready
Type:                     ClusterIP
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.108.19.168
IPs:                      10.108.19.168
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
Endpoints:                10.44.0.30:80
Session Affinity:         None
Internal Traffic Policy:  Cluster
Events:                   <none>

➜ candidate@cka3200:~$ k get ep service-am-i-ready
NAME                ENDPOINTS       AGE
service-am-i-ready   10.44.0.30:80   6d19h
```

Which will result in our first *Pod* being ready, just give it a minute for the Readiness probe to check again:

```
➜ candidate@cka3200:~$ k get pod ready-if-service-ready
NAME                    READY   STATUS    RESTARTS   AGE
ready-if-service-ready   1/1     Running   0          2m10s
```

Look at these *Pods* working together!

Question 5 | Kubectl sorting

Solve this question on: `ssh cka8448`

Create two bash script files which use `kubectl` sorting to:

1. Write a command into `/opt/course/5/find_pods.sh` which lists all *Pods* in all *Namespaces* sorted by their AGE (`metadata.creationTimestamp`)
2. Write a command into `/opt/course/5/find_pods_uid.sh` which lists all *Pods* in all *Namespaces* sorted by field `metadata.uid`

Answer:

A good resources here (and for many other things) is the kubectl-cheat-sheet. You can reach it fast when searching for "cheat sheet" in the Kubernetes docs.

Step 1

```
➜ ssh cka8448

➜ candidate@cka8448:~$ vim /opt/course/5/find_pods.sh
```

```
# cka8448:/opt/course/5/find_pods.sh
kubectl get pod -A --sort-by=.metadata.creationTimestamp
```

We should be able to execute it and see sorting by AGE:

```
➜ sh /opt/course/5/find_pods.sh
NAMESPACE     NAME                                    READY   ...
kube-system   kube-proxy-dvv7m                        1/1     ...
kube-system   weave-net-gjrxh                         2/2     ...
kube-system   etcd-cka8448                            1/1     ...
kube-system   kube-apiserver-cka8448                  1/1     ...
kube-system   kube-scheduler-cka8448                  1/1     ...
kube-system   kube-controller-manager-cka8448         1/1     ...
default       berlin-external-monitor-6c8fd896dd-66tvw   1/1     ...
default       berlin-external-proxy-98bccbc68-59gjg   1/1     ...
default       berlin-external-proxy-98bccbc68-phpvt   1/1     ...
kube-system   coredns-6f8b9d9f4b-8z7rb                1/1     ...
kube-system   coredns-6f8b9d9f4b-fg7bt                1/1     ...
```

Step 2

For the second command we create file:

```
# cka8448:/opt/course/5/find_pods_uid.sh
kubectl get pod -A --sort-by=.metadata.uid
```

When we execute we should see a different sorting order:

```
➜ sh /opt/course/5/find_pods_uid.sh
NAMESPACE     NAME                                       READY   ...
kube-system   kube-proxy-dvv7m                           1/1     ...
kube-system   coredns-6f8b9d9f4b-8z7rb                   1/1     ...
default       berlin-external-monitor-6c8fd896dd-66tvw   1/1     ...
default       berlin-external-proxy-98bccbc68-59gjg      1/1     ...
default       berlin-external-proxy-98bccbc68-phpvt      1/1     ...
kube-system   kube-controller-manager-cka8448            1/1     ...
kube-system   kube-scheduler-cka8448                     1/1     ...
kube-system   kube-apiserver-cka8448                     1/1     ...
kube-system   etcd-cka8448                               1/1     ...
kube-system   coredns-6f8b9d9f4b-fg7bt                   1/1     ...
kube-system   weave-net-gjrxh                            2/2     ...
```

Question 6 | Fix Kubelet

Solve this question on: `ssh cka1024`

There seems to be an issue with the kubelet on controlplane node `cka1024`, it's not running.

Fix the kubelet and confirm that the node is available in Ready state.

Create a *Pod* called `success` in `default` *Namespace* of image `nginx:1-alpine`.

ℹ️ The node has no taints and can schedule *Pods* without additional tolerations

Answer:

The procedure on scenarios like these is to first check if the kubelet is running. If it isn't start it, check its logs and fix issues if there are some. It could also be helpful to look at the config files of other clusters and diff/compare.

Investigate

Check node status:

```
➜ ssh cka1024

➜ candidate@cka1024:~$ k get node
E0423 12:27:08.326639   12871 memcache.go:265] "Unhandled Error" err="couldn't get current server API group
list: Get \"https://192.168.100.41:6443/api?timeout=32s\": dial tcp 192.168.100.41:6443: connect: connection
refused"
E0423 12:27:08.329430   12871 memcache.go:265] "Unhandled Error" err="couldn't get current server API group
list: Get \"https://192.168.100.41:6443/api?timeout=32s\": dial tcp 192.168.100.41:6443: connect: connection
refused"
E0423 12:27:08.332448   12871 memcache.go:265] "Unhandled Error" err="couldn't get current server API group
list: Get \"https://192.168.100.41:6443/api?timeout=32s\": dial tcp 192.168.100.41:6443: connect: connection
refused"
E0423 12:27:08.335352   12871 memcache.go:265] "Unhandled Error" err="couldn't get current server API group
list: Get \"https://192.168.100.41:6443/api?timeout=32s\": dial tcp 192.168.100.41:6443: connect: connection
refused"
E0423 12:27:08.342153   12871 memcache.go:265] "Unhandled Error" err="couldn't get current server API group
list: Get \"https://192.168.100.41:6443/api?timeout=32s\": dial tcp 192.168.100.41:6443: connect: connection
refused"
The connection to the server 192.168.100.41:6443 was refused - did you specify the right host or port?
```

Okay, this looks very wrong. First we check if the kubelet is running:

```
➜ candidate@cka1024:~$ sudo -i

➜ root@cka1024:~# ps aux | grep kubelet
root       12892  0.0  0.1   7076  ...  0:00 grep --color=auto kubelet
```

No kubectl process running, just the grep command itself is displayed. We check if the kubelet is configured as service, which is default for a kubeadm installation:

```
➜ root@cka1024:~# service kubelet status
○ kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: inactive (dead) since Sun 2025-03-23 08:16:52 UTC; 1 month 0 days ago
   Duration: 2min 46.830s
       Docs: https://kubernetes.io/docs/
   Main PID: 7346 (code=exited, status=0/SUCCESS)
        CPU: 5.956s
...
```

We can see it's not running (inactive) in this line:

```
Active: inactive (dead) since Sun 2025-03-23 08:16:52 UTC; 1 month 0 days ago
```

But the kubelet is configured as a service with config at `/usr/lib/systemd/system/kubelet.service`, let's try to start it:

```
➜ root@cka1024:~# service kubelet start

➜ root@cka1024:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: activating (auto-restart) (Result: exit-code) since Wed 2025-04-23 12:31:07 UTC; 2s ago
       Docs: https://kubernetes.io/docs/
```

```
    Process: 13014 ExecStart=/usr/local/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS
$KUBELET_KUBEADM_ARGS $KUBELET_EX>
   Main PID: 13014 (code=exited, status=203/EXEC)
        CPU: 10ms

Apr 23 12:31:07 cka1024 systemd[1]: kubelet.service: Failed with result 'exit-code'.
```

Above we see it's trying to execute `/usr/local/bin/kubelet` in this line:

```
Process: 13014 ExecStart=/usr/local/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS
$KUBELET_KUBEADM_ARGS $KUBELET_EX>
```

It does so with some arguments defined in its service config file. A good way to find errors and get more info is to run the command manually:

```
➜ root@cka1024:~# /usr/local/bin/kubelet
-bash: /usr/local/bin/kubelet: No such file or directory

➜ root@cka1024:~# whereis kubelet
kubelet: /usr/bin/kubelet
```

That's the issue: wrong path to the kubelet binary.

Read Logs

Usually we need to dig a bit deeper and check logs using `journalctl -u kubelet` or `cat /var/log/syslog | grep kubelet`:

```
➜  root@cka1024:~# cat /var/log/syslog | grep kubelet
2025-03-23T08:13:26.775366+00:00 ubuntu systemd[1]: Started kubelet.service - kubelet: The Kubernetes Node
Agent.
2025-03-23T08:13:26.782571+00:00 ubuntu (kubelet)[6826]: kubelet.service: Referenced but unset environment
variable evaluates to an empty string: KUBELET_KUBEADM_ARGS
...
2025-04-23T12:31:48.264234+00:00 ubuntu systemd[1]: kubelet.service: Scheduled restart job, restart counter
is at 5.
2025-04-23T12:31:48.272108+00:00 ubuntu systemd[1]: Started kubelet.service - kubelet: The Kubernetes Node
Agent.
2025-04-23T12:31:48.284966+00:00 ubuntu systemd[1]: kubelet.service: Main process exited, code=exited,
status=203/EXEC
2025-04-23T12:31:48.285487+00:00 ubuntu systemd[1]: kubelet.service: Failed with result 'exit-code'.
```

If we check logs we should always look at the time, we probably only want the latest ones. Here we see:

```
kubelet.service: Main process exited, code=exited, status=203/EXEC
```

The logs don't show any error messages from the kubelet itself. Usually if the kubelet is started and exits because of an error, like an unknown argument passed, there will be error logs. But because there is nothing more here it could be a good idea to try to execute the kubelet binary manually.

We already did this above before checking the logs and it showed us that a wrong binary path was used in the service config file.

Fix the Kubelet

We go ahead and correct the path in file `/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf`:

```
➜ root@cka1024:~# vim /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
```

```
# cka1024:/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf

# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the
KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the
user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS
should be sourced from this file.
EnvironmentFile=-/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS
$KUBELET_EXTRA_ARGS
```

In the very last line we updated the binary path to `/usr/bin/kubelet`.

Now we reload the service:

```
➜ root@cka1024:~# systemctl daemon-reload

➜ root@cka1024:~# service kubelet restart

➜ root@cka1024:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Wed 2025-04-23 12:33:25 UTC; 5s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 13124 (kubelet)
      Tasks: 9 (limit: 1317)
     Memory: 88.3M (peak: 88.6M)
        CPU: 1.093s
     CGroup: /system.slice/kubelet.service
             └─13124 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/ku>
...

➜ root@cka1024:~# ps aux | grep kubelet
root       13124  9.2  7.1 1896084 82432 ?       Ssl  12:33   0:01 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --container-runtime-endpoint=unix:///var/run/containerd/containerd.sock
--pod-infra-container-image=registry.k8s.io/pause:3.10
...
```

That looks much better. We can wait for the containers to appear, which can take a minute:

```
➜ root@cka1024:~# watch crictl ps
CONTAINER       ...  CREATED           STATE       NAME                      ...
ccfbd17742b05   ...  25 seconds ago    Running     kube-controller-manager   ...
ff3910e3c8c6c   ...  25 seconds ago    Running     kube-scheduler            ...
9b49473786774   ...  25 seconds ago    Running     kube-apiserver            ...
f5de1f6e11d5c   ...  26 seconds ago    Running     etcd                      ...
```

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.

Also the node should be available, give it a bit of time though:

```
➜ root@cka1024:~# k get node
NAME       STATUS   ROLES           AGE   VERSION
cka1024    Ready    control-plane   31d   v1.33.1
```

ℹ️ It might take some time till `k get node` doesn't throw any errors after fixing the issue

Finally we create the requested *Pod*:

```
➜ root@cka1024:~# k run success --image nginx:1-alpine
pod/success created

➜ root@cka1024:~# k get pod success -o wide
NAME      READY   STATUS    ...   NODE      NOMINATED NODE   READINESS GATES
success   1/1     Running   ...   cka1024   <none>           <none>
```

Question 7 | Etcd Operations

Solve this question on: `ssh cka2560`

You have been tasked to perform the following etcd operations:

1. Run `etcd --version` and store the output at `/opt/course/7/etcd-version`
2. Make a snapshot of etcd and save it at `/opt/course/7/etcd-snapshot.db`

Answer:

Step 1: Etcd Version

Here we simply need to execute a command, shouldn't be that hard:

```
➜ ssh cka2560

➜ candidate@cka2560:~$ sudo -i

➜ root@cka2560:~# etcd --version
Command 'etcd' not found, but can be installed with:
apt install etcd-server
```

Well, etcd is not installed directly on the controlplane but it runs as a *Pod* instead. So we do:

```
root@cka2560:~# k -n kube-system get pod
NAME                              READY   STATUS    RESTARTS      AGE
coredns-78c4c75bb8-fgkfv          1/1     Running   0             15d
coredns-78c4c75bb8-l7mmh          1/1     Running   0             15d
etcd-cka2560                      1/1     Running   0             13m
kube-apiserver-cka2560            1/1     Running   0             15d
kube-controller-manager-cka2560   1/1     Running   0             15d
kube-proxy-f56td                  1/1     Running   0             15d
kube-scheduler-cka2560            1/1     Running   0             15d
weave-net-44k9c                   2/2     Running   1 (15d ago)   15d

root@cka2560:~# k -n kube-system exec etcd-cka2560 -- etcd --version
etcd Version: 3.5.21
Git SHA: a17edfd
Go Version: go1.23.7
Go OS/Arch: linux/amd64

root@cka2560:~# k -n kube-system exec etcd-cka2560 -- etcd --version > /opt/course/7/etcd-version
```

Step 2: Etcd Snapshot

First we log into the controlplane and try to create a snapshot of etcd:

```
➜ ssh cka2560

➜ candidate@cka2560:~$ sudo -i

➜ root@cka2560:~# ETCDCTL_API=3 etcdctl snapshot save /opt/course/7/etcd-snapshot.db
{"level":"info","ts":"2024-11-07T14:02:17.746254Z","caller":"snapshot/v3_snapshot.go:65","msg":"created
temporary db file","path":"/opt/course/7/etcd-snapshot.db.part"}
^C
```

But it fails or hangs because we need to authenticate ourselves. For the necessary information we can check the etc manifest:

```
➜ root@cka2560:~# vim /etc/kubernetes/manifests/etcd.yaml
```

We only check the `etcd.yaml` for necessary information we don't change it.

```yaml
# cka2560:/etc/kubernetes/manifests/etcd.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
```

```
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.100.31:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt                              # use
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd
    - --initial-advertise-peer-urls=https://192.168.100.31:2380
    - --initial-cluster=cka2560=https://192.168.100.31:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key                               # use
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.100.31:2379   # use
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://192.168.100.31:2380
    - --name=cka2560
    - --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                       # use
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: k8s.gcr.io/etcd:3.3.15-0
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 127.0.0.1
        path: /health
        port: 2381
        scheme: HTTP
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: etcd
    resources: {}
    volumeMounts:
    - mountPath: /var/lib/etcd
      name: etcd-data
    - mountPath: /etc/kubernetes/pki/etcd
      name: etcd-certs
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
    name: etcd-certs
  - hostPath:
      path: /var/lib/etcd                                                      # important
      type: DirectoryOrCreate
    name: etcd-data
status: {}
```

But we also know that the api-server is connecting to etcd, so we can check how its manifest is configured:

```
➜ root@cka2560:~# cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep etcd
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

We use the authentication information and pass it to etcdctl:

```
ETCDCTL_API=3 etcdctl snapshot save /opt/course/7/etcd-snapshot.db \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/server.crt \
--key /etc/kubernetes/pki/etcd/server.key
```

Which should provide successful output:

```
➜ root@cka2560:~# ETCDCTL_API=3 etcdctl snapshot save /opt/course/7/etcd-snapshot.db \
--cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/server.crt \
--key /etc/kubernetes/pki/etcd/server.key
{"level":"info","ts":"2025-03-02T13:35:48.806437Z","caller":"snapshot/v3_snapshot.go:65","msg":"created
temporary db file","path":"/opt/course/7/etcd-snapshot.db.part"}
{"level":"info","ts":"2025-03-
02T13:35:48.929550Z","logger":"client","caller":"v3@v3.5.16/maintenance.go:212","msg":"opened snapshot
stream; downloading"}
{"level":"info","ts":"2025-03-02T13:35:48.929975Z","caller":"snapshot/v3_snapshot.go:73","msg":"fetching
snapshot","endpoint":"127.0.0.1:2379"}
{"level":"info","ts":"2025-03-
02T13:35:49.110620Z","logger":"client","caller":"v3@v3.5.16/maintenance.go:220","msg":"completed snapshot
read; closing"}
{"level":"info","ts":"2025-03-02T13:35:49.155626Z","caller":"snapshot/v3_snapshot.go:88","msg":"fetched
snapshot","endpoint":"127.0.0.1:2379","size":"2.4 MB","took":"now"}
{"level":"info","ts":"2025-03-
02T13:35:49.155886Z","caller":"snapshot/v3_snapshot.go:97","msg":"saved","path":"/opt/course/7/etcd-
snapshot.db"}
Snapshot saved at /opt/course/7/etcd-snapshot.db
```

ℹ️ Don't use `snapshot status` because it can alter the snapshot file and render it invalid in certain etcd versions

(Optional) Etcd Restore

ℹ️ Doing this wrong can leave this cluster broken and will affect this question and also others

We create a *Pod* in the cluster and wait for it to be running:

```
➜ root@cka2560:~# kubectl run test --image=nginx
pod/test created

➜ root@cka2560:~# kubectl get pod -l run=test
NAME   READY   STATUS    RESTARTS   AGE
test   1/1     Running   0          17s
```

Next we stop all controlplane components:

```
➜ root@cka2560:~# cd /etc/kubernetes/manifests/

➜ root@cka2560:/etc/kubernetes/manifests# mv * ..

➜ root@cka2560:/etc/kubernetes/manifests# watch crictl ps
```

It's **very important** to wait for all K8s controlplane containers to be removed before continuing. This can take a minute!

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.

Now we restore the snapshot into a specific directory:

```
➜ root@cka2560:~# TCDCTL_API=3 etcdctl snapshot restore /opt/course/7/etcd-snapshot.db --data-dir
/var/lib/etcd-snapshot --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt -
-key /etc/kubernetes/pki/etcd/server.key
Deprecated: Use `etcdutl snapshot restore` instead.

2025-03-02T13:38:07Z    info    snapshot/v3_snapshot.go:265     restoring snapshot      {"path":
"/opt/course/7/etcd-snapshot.db", "wal-dir": "/var/lib/etcd-snapshot/member/wal", "data-dir":
"/var/lib/etcd-snapshot", "snap-dir": "/var/lib/etcd-snapshot/member/snap", "initial-memory-map-size": 0}
2025-03-02T13:38:07Z    info    membership/store.go:141 Trimming membership information from the backend...
2025-03-02T13:38:07Z    info    membership/cluster.go:421       added member    {"cluster-id":
"cdf818194e3a8c32", "local-member-id": "0", "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls":
["http://localhost:2380"]}
2025-03-02T13:38:08Z    info    snapshot/v3_snapshot.go:293     restored snapshot       {"path":
"/opt/course/7/etcd-snapshot.db", "wal-dir": "/var/lib/etcd-snapshot/member/wal", "data-dir":
"/var/lib/etcd-snapshot", "snap-dir": "/var/lib/etcd-snapshot/member/snap", "initial-memory-map-size": 0}
```

We could specify another host to make the backup from by using `etcdctl --endpoints http://IP`, but here we just use the default value which is: `http://127.0.0.1:2379,http://127.0.0.1:4001`.

The restored files are located at the new folder `/var/lib/etcd-snapshot`, now we have to tell etcd to use that directory:

```
➜ root@cka2560:~# vim /etc/kubernetes/etcd.yaml
```

```
# /etc/kubernetes/etcd.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
...
    - mountPath: /etc/kubernetes/pki/etcd
      name: etcd-certs
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
    name: etcd-certs
  - hostPath:
      path: /var/lib/etcd-snapshot                # change
      type: DirectoryOrCreate
    name: etcd-data
status: {}
```

Now we move all controlplane yaml again into the manifest directory. Give it some time (up to several minutes) for etcd to restart and for the api-server to be reachable again:

```
➜ root@cka2560:/etc/kubernetes/manifests# mv ../*.yaml .

➜ root@cka2560:/etc/kubernetes/manifests# watch crictl ps
```

Then we check again for the *Pod*:

```
➜ root@cka2560:~# kubectl get pod -l run=test
No resources found in default namespace.
```

Awesome, snapshot and restore worked as our *Pod* is gone.

Question 8 | Get Controlplane Information

Solve this question on: `ssh cka8448`

Check how the controlplane components kubelet, kube-apiserver, kube-scheduler, kube-controller-manager and etcd are started/installed on the controlplane node.

Also find out the name of the DNS application and how it's started/installed in the cluster.

Write your findings into file `/opt/course/8/controlplane-components.txt`. The file should be structured like:

```
# /opt/course/8/controlplane-components.txt
kubelet: [TYPE]
kube-apiserver: [TYPE]
kube-scheduler: [TYPE]
kube-controller-manager: [TYPE]
etcd: [TYPE]
dns: [TYPE] [NAME]
```

Choices of `[TYPE]` are: `not-installed`, `process`, `static-pod`, `pod`

Answer:

We could start by finding processes of the requested components, especially the kubelet at first:

```
➜ ssh cka8448

➜ candidate@cka8448:~$ sudo -i

➜ root@cka8448:~# ps aux | grep kubelet
```

We can see which components are controlled via systemd looking at `/usr/lib/systemd` directory:

```
➜ root@cka8448:~# find /usr/lib/systemd | grep kube
/usr/lib/systemd/user/podman-kube@.service
/usr/lib/systemd/system/kubelet.service.d
/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
/usr/lib/systemd/system/kubelet.service
/usr/lib/systemd/system/podman-kube@.service
```

```
➜ root@cka8448:~# service kubelet status
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Sun 2024-12-08 16:10:53 UTC; 1h 6min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 7355 (kubelet)
      Tasks: 11 (limit: 1317)
     Memory: 69.0M (peak: 75.9M)
        CPU: 1min 58.582s
     CGroup: /system.slice/kubelet.service
             └─7355 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet>
...

➜ root@cka8448:~# find /usr/lib/systemd | grep etcd
```

This shows kubelet is controlled via systemd, but no other service named kube nor etcd. It seems that this cluster has been setup using kubeadm, so we check in the default manifests directory:

```
➜ root@cka8448:~# find /etc/kubernetes/manifests/
/etc/kubernetes/manifests/
/etc/kubernetes/manifests/kube-controller-manager.yaml
/etc/kubernetes/manifests/etcd.yaml
/etc/kubernetes/manifests/kube-apiserver.yaml
/etc/kubernetes/manifests/kube-scheduler.yaml
```

The kubelet could also have a different manifests directory specified via a `KubeletConfiguration`, but the one above is the default one.

This means the main 4 controlplane services are setup as static *Pods*. Actually, let's check all *Pods* running on in the `kube-system` *Namespace*:

```
➜ root@cka8448:~# k -n kube-system get pod -o wide
NAME                              ...   NODE
coredns-6f8b9d9f4b-8z7rb          ...   cka8448
coredns-6f8b9d9f4b-fg7bt          ...   cka8448
etcd-cka8448                      ...   cka8448
kube-apiserver-cka8448            ...   cka8448
kube-controller-manager-cka8448   ...   cka8448
kube-proxy-dvv7m                  ...   cka8448
kube-scheduler-cka8448            ...   cka8448
weave-net-gjrxh                   ...   cka8448
```

Above we see the 4 static pods, with `-cka8448` as suffix.

We also see that the dns application seems to be coredns, but how is it controlled?

```
➜ root@cka8448$ kubectl -n kube-system get ds
NAME         DESIRED   ...   NODE SELECTOR            AGE
kube-proxy   1         ...   kubernetes.io/os=linux   67m
weave-net    1         ...   <none>                   67m

➜ root@cka8448$ k -n kube-system get deploy
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
coredns   2/2     2            2           68m
```

Seems like coredns is controlled via a *Deployment*. We combine our findings in the requested file:

```
# /opt/course/8/controlplane-components.txt
kubelet: process
kube-apiserver: static-pod
kube-scheduler: static-pod
kube-controller-manager: static-pod
etcd: static-pod
dns: pod coredns
```

You should be comfortable investigating a running cluster, know different methods on how a cluster and its services can be setup and be able to troubleshoot and find error sources.

Question 9 | Kill Scheduler, Manual Scheduling

Solve this question on: `ssh cka5248`

Temporarily stop the kube-scheduler, this means in a way that you can start it again afterwards.

Create a single *Pod* named `manual-schedule` of image `httpd:2-alpine`, confirm it's created but not scheduled on any node.

Now you're the scheduler and have all its power, manually schedule that *Pod* on node `cka5248`. Make sure it's running.

Start the kube-scheduler again and confirm it's running correctly by creating a second *Pod* named `manual-schedule2` of image `httpd:2-alpine` and check if it's running on `cka5248-node1`.

Answer:

Stop the Scheduler

First we find the controlplane node:

```
➜ ssh cka5248

➜ candidate@cka5248:~$ k get node
NAME            STATUS   ROLES           AGE     VERSION
cka5248         Ready    control-plane   6d22h   v1.33.1
cka5248-node1   Ready    <none>          6d22h   v1.33.1
```

Then we connect and check if the scheduler is running:

```
➜ candidate@cka5248:~$ sudo -i

➜ root@cka5248:~# kubectl -n kube-system get pod | grep schedule
kube-scheduler-cka5248            1/1     Running   0              6d22h
```

Kill the Scheduler (temporarily):

```
➜ root@cka5248:~# cd /etc/kubernetes/manifests/

➜ root@cka5248:~# mv kube-scheduler.yaml ..
```

And it should be stopped, we can wait for the container to be removed with `watch crictl ps`:

```
➜ root@cka5248:/etc/kubernetes/manifests# watch crictl ps

➜ root@cka5248:/etc/kubernetes/manifests# kubectl -n kube-system get pod | grep schedule

➜ root@cka5248:/etc/kubernetes/manifests#
```

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.

Create a *Pod*

Now we create the *Pod*:

```
➜ root@cka5248:~# k run manual-schedule --image=httpd:2-alpine
pod/manual-schedule created
```

And confirm it has no node assigned:

```
➜ root@cka5248:~# k get pod manual-schedule -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP       NODE     ...
manual-schedule   0/1     Pending   0          14s   <none>   <none>   ...
```

Manually schedule the *Pod*

Let's play the scheduler now:

```
➜ root@cka5248:~# k get pod manual-schedule -o yaml > 9.yaml
```

```yaml
# cka5248:/root/9.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-09-04T15:51:02Z"
  labels:
    run: manual-schedule
  managedFields:
...
    manager: kubectl-run
    operation: Update
    time: "2020-09-04T15:51:02Z"
  name: manual-schedule
  namespace: default
  resourceVersion: "3515"
  selfLink: /api/v1/namespaces/default/pods/manual-schedule
  uid: 8e9d2532-4779-4e63-b5af-feb82c74a935
spec:
  nodeName: cka5248       # ADD the controlplane node name
  containers:
```

```
  - image: httpd:2-alpine
    imagePullPolicy: IfNotPresent
    name: manual-schedule
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-nxnc7
      readOnly: true
  dnsPolicy: ClusterFirst
...
```

The scheduler sets the nodeName for a *Pod* declaration. How it finds the correct node to schedule on, that's a very much complicated matter and takes many variables into account.

As we cannot `kubectl apply` or `kubectl edit` , in this case we need to delete and create or replace:

```
➜ root@cka5248:~# k -f 9.yaml replace --force
```

How does it look?

```
➜ root@cka5248:~# k get pod manual-schedule -o wide
NAME              READY   STATUS    ...   NODE
manual-schedule   1/1     Running   ...   cka5248
```

It looks like our *Pod* is running on the controlplane now as requested, although no tolerations were specified. Only the scheduler takes taints/tolerations/affinity into account when finding the correct node name. That's why it's still possible to assign *Pods* manually directly to a controlplane node and skip the scheduler.

Start the scheduler again

```
➜ root@cka5248:~# cd /etc/kubernetes/manifests/

➜ root@cka5248:/etc/kubernetes/manifests# mv ../kube-scheduler.yaml .
```

Checks it's running:

```
➜ root@cka5248:~# kubectl -n kube-system get pod | grep schedule
kube-scheduler-cka5248            1/1     Running   0              13s
```

Schedule a second test *Pod*:

```
➜ root@cka5248:~# k run manual-schedule2 --image=httpd:2-alpine

➜ root@cka5248:~# k get pod -o wide | grep schedule
manual-schedule    1/1     Running   0          95s   10.32.0.2   cka5248
manual-schedule2   1/1     Running   0          9s    10.44.0.3   cka5248-node1
```

Back to normal.

Question 10 | PV PVC Dynamic Provisioning

Solve this question on: `ssh cka6016`

There is a backup *Job* which needs to be adjusted to use a *PVC* to store backups.

Create a *StorageClass* named `local-backup` which uses `provisioner: rancher.io/local-path` and `volumeBindingMode: WaitForFirstConsumer`. To prevent possible data loss the *StorageClass* should keep a *PV* retained even if a bound *PVC* is deleted.

Adjust the *Job* at `/opt/course/10/backup.yaml` to use a *PVC* which request `50Mi` storage and uses the new *StorageClass*.

Deploy your changes, verify the *Job* completed once and the *PVC* was bound to a newly created *PV*.

> ℹ️ To re-run a *Job*, delete it and create it again

> ℹ️ The abbreviation *PV* stands for *PersistentVolume* and *PVC* for *PersistentVolumeClaim*

Answer:

The *StorageClass* should use provider `rancher.io/local-path`, which is of the project Local Path Provisioner. This project works with Dynamic Volume Provisioning, but instead of creating actual volumes it uses local storage on the node where the *Pod* runs, by default at path `/opt/local-path-provisioner`.

Cloud companies like AWS or GCP provide their own *StorageClasses* and providers, which if used for *PVCs* create *PVs* backed by actual volumes in the cloud account.

Create *StorageClass*

First we can have a look at existing ones:

```
➜ ssh cka6016

➜ candidate@cka6016:~$ k get sc
NAME         PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ...
local-path   rancher.io/local-path   Delete          WaitForFirstConsumer   ...
```

The `local-path` is the default one available if the Local Path Provisioner is installed. But we can see it has a reclaimPolicy of Delete. Still we could use this one as template for the one we need to create:

```
➜ candidate@cka6016:~$ vim sc.yaml
```

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-backup
provisioner: rancher.io/local-path
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

We need to use `reclaimPolicy: Retain` because this will cause the *PV* to not get deleted even after the associated *PVC* is deleted. It's very easy to delete resources in Kubernetes which can lead to quick data loss. Especially in this case where important data, like from a backup, is in play.

```
➜ candidate@cka6016:~$ k -f sc.yaml apply
storageclass.storage.k8s.io/local-backup created

➜ candidate@cka6016:~$ k get sc
NAME           PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ...
local-backup   rancher.io/local-path   Retain          WaitForFirstConsumer   ...
local-path     rancher.io/local-path   Delete          WaitForFirstConsumer   ...
```

This looks like what we want. Now we have the choice between two *StorageClasses*.

Check existing *Job*

Let's have a look at the existing *Job*:

```
# cka6016:/opt/course/10/backup.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: backup
  namespace: project-bern
spec:
  backoffLimit: 0
  template:
    spec:
      volumes:
        - name: backup
          emptyDir: {}
      containers:
        - name: bash
          image: bash:5
          command:
            - bash
            - -c
            - |
              set -x
              touch /backup/backup-$(date +%Y-%m-%d-%H-%M-%S).tar.gz
              sleep 15
          volumeMounts:
            - name: backup
              mountPath: /backup
      restartPolicy: Never
```

Currently it uses an emptyDir volume which means in only stores data in the temporary filesystem of the *Pod*. This means once the *Pod* is deleted the data is deleted as well.

We could go ahead and create it now to see if everything else works:

```
➜ candidate@cka6016:~$ k -f /opt/course/10/backup.yaml apply
job.batch/backup created

➜ candidate@cka6016:~$ k -n project-bern get job,pod
NAME               STATUS     COMPLETIONS   DURATION   AGE
job.batch/backup   Complete   1/1           5s         11s

NAME               READY   STATUS      RESTARTS   AGE
pod/backup-pll27   0/1     Completed   0          21s
```

Looks like it completed without errors.

Adjust *Job* template

For this we first need to create a *PVC* and then use in the *Job* template:

```
➜ candidate@cka6016:~$ cd /opt/course/10

➜ candidate@cka6016:/opt/course/10$ cp backup.yaml backup.yaml_ori

➜ candidate@cka6016:/opt/course/10$ vim backup.yaml
```

```yaml
# cka6016:/opt/course/10/backup.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: backup-pvc
  namespace: project-bern            # use same Namespace
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Mi                  # request the required size
  storageClassName: local-backup     # use the new StorageClass
---
apiVersion: batch/v1
kind: Job
metadata:
  name: backup
  namespace: project-bern
spec:
  backoffLimit: 0
  template:
    spec:
      volumes:
        - name: backup
          persistentVolumeClaim:     # CHANGE
            claimName: backup-pvc    # CHANGE
      containers:
        - name: bash
          image: bash:5
          command:
            - bash
            - -c
            - |
              set -x
              touch /backup/backup-$(date +%Y-%m-%d-%H-%M-%S).tar.gz
              sleep 15
          volumeMounts:
            - name: backup
```

```
            mountPath: /backup
      restartPolicy: Never
```

We first made a backup of the provided file, which is always a good idea. Then we added the new *PVC* and referenced the *PVC* in the *Pod* `volumes:` section.

Deploy changes and verify

First we delete the existing *Job* because we did create it once before without any changes. And then we deploy:

```
➜ candidate@cka6016:/opt/course/10$ k delete -f backup.yaml
job.batch "backup" deleted

➜ candidate@cka6016:/opt/course/10$ k apply -f backup.yaml
persistentvolumeclaim/backup-pvc created
job.batch/backup created
```

Then we should see the *Job* execution created a *Pod* which used the *PVC* which created a *PV*:

```
➜ candidate@cka6016:/opt/course/10$ k -n project-bern get job,pod,pvc,pv
NAME               STATUS    COMPLETIONS   DURATION   AGE
job.batch/backup   Running   0/1           13s        13s

NAME               READY   STATUS    RESTARTS   AGE
pod/backup-q7dgx   1/1     Running   0          13s

NAME          STATUS   VOLUME                                     CAPACITY   ...
backup-pvc    Bound    pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a   50Mi       ...

NAME          CAPACITY   ...  RECLAIM POLICY  STATUS  CLAIM                     ...
pvc-dbcce...  50Mi       ...  Retain          Bound   project-bern/backup-pvc   ...
```

Optional investigation

Because the Local Path Provisioner is used we can actually see the volume represented on the filesystem. And because this cluster only has one node, and we're already on it, we can simply do:

```
➜ candidate@cka6016:~$ find /opt/local-path-provisioner
/opt/local-path-provisioner/
/opt/local-path-provisioner/pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a_project-bern_backup-pvc
/opt/local-path-provisioner/pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a_project-bern_backup-pvc/backup-2024-12-30-17-27-51.tar.gz
```

If we run the *Job* again we should see another backup file:

```
➜ candidate@cka6016:~$ k -n project-bern delete job backup
job.batch "backup" deleted

➜ candidate@cka6016:~$ k apply -f backup.yaml
persistentvolumeclaim/backup-pvc unchanged
job.batch/backup created

➜ candidate@cka6016:~$ k -n project-bern get job,pod,pvc,pv
NAME               STATUS     COMPLETIONS   DURATION   AGE
job.batch/backup   Complete   1/1           18s        20s
```

```
NAME                READY   STATUS      RESTARTS   AGE
pod/backup-jpq2t    0/1     Completed   0          20s

➜ candidate@cka6016:~$ find /opt/local-path-provisioner
/opt/local-path-provisioner/
/opt/local-path-provisioner/pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a_project-bern_backup-pvc
/opt/local-path-provisioner/pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a_project-bern_backup-pvc/backup-2024-12-30-17-27-51.tar.gz
/opt/local-path-provisioner/pvc-dbccec94-cc31-4e30-b5fe-7cb42a85fe7a_project-bern_backup-pvc/backup-2024-12-30-17-34-26.tar.gz
```

And if we delete the *PVC* we should still see the *PV* and the files in the volume (filesystem in this case):

ℹ️ Removing the *PVC* and *Job* might affect your scoring for this question, so best create them again after testing deletion

```
➜ candidate@cka6016:~$ k -n project-bern delete pvc backup-pvc
persistentvolumeclaim "backup-pvc" deleted

➜ candidate@cka6016:~$ k get pv,pvc -A
NAME          CAPACITY   ...  RECLAIM POLICY   STATUS     CLAIM                     ...
pvc-dbcce...  50Mi       ...  Retain           Released   project-bern/backup-pvc   ...
```

We can no longer see the *PVC*, but the *PV* is in status `Released`. This is because we set the `reclaimPolicy: Retain` in the *StorageClass*. Now we could manually export/rescue the data in the volume and afterwards delete the *PV* manually.

Question 11 | Create Secret and mount into Pod

Solve this question on: `ssh cka2560`

Create *Namespace* `secret` and implement the following in it:

- Create *Pod* `secret-pod` with image `busybox:1`. It should be kept running by executing `sleep 1d` or something similar
- Create the existing *Secret* `/opt/course/11/secret1.yaml` and mount it readonly into the *Pod* at `/tmp/secret1`
- Create a new *Secret* called `secret2` which should contain `user=user1` and `pass=1234`. These entries should be available inside the *Pod's* container as environment variables `APP_USER` and `APP_PASS`

Answer

First we create the *Namespace*:

```
➜ ssh cka2560

➜ candidate@cka2560:~$ k create ns secret
namespace/secret created
```

Secret 1

To create the existing *Secret* we need to adjust the *Namespace* for it:

```
➜ candidate@cka2560:~$ cp /opt/course/11/secret1.yaml 11_secret1.yaml
```

```
# cka2560:/home/candidate/11_secret1.yaml
apiVersion: v1
data:
  halt: IyEgL2Jpbi9zaAo...
kind: Secret
metadata:
  creationTimestamp: null
  name: secret1
  namespace: secret           # UPDATE
```

```
➜ candidate@cka2560:~$ k -f 11_secret1.yaml create
secret/secret1 created
```

Secret 2

Next we create the second *Secret*:

```
➜ candidate@cka2560:~$ k -n secret create secret generic secret2 --from-literal=user=user1 --from-
literal=pass=1234
secret/secret2 created
```

Pod

Now we create the *Pod* template:

```
➜ candidate@cka2560:~$ k -n secret run secret-pod --image=busybox:1 --dry-run=client -o yaml -- sh -c "sleep
1d" > 11.yaml
```

Then make the necessary changes:

```
# cka2560:/home/candidate/11.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: secret-pod
  name: secret-pod
  namespace: secret                       # important if not automatically added
spec:
  containers:
  - args:
    - sh
    - -c
    - sleep 1d
    image: busybox:1
    name: secret-pod
    resources: {}
    env:                                  # add
    - name: APP_USER                      # add
      valueFrom:                          # add
```

```
        secretKeyRef:                  # add
          name: secret2                # add
          key: user                    # add
    - name: APP_PASS                   # add
      valueFrom:                       # add
        secretKeyRef:                  # add
          name: secret2                # add
          key: pass                    # add
    volumeMounts:                      # add
    - name: secret1                    # add
      mountPath: /tmp/secret1          # add
      readOnly: true                   # add
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:                             # add
  - name: secret1                      # add
    secret:                            # add
      secretName: secret1              # add
status: {}
```

And execute:

```
➜ candidate@cka2560:~$ k -f 11.yaml create
pod/secret-pod created
```

Finally we verify:

```
➜ candidate@cka2560:~$ k -n secret exec secret-pod -- env | grep APP
APP_PASS=1234
APP_USER=user1

➜ candidate@cka2560:~$ k -n secret exec secret-pod -- find /tmp/secret1
/tmp/secret1
/tmp/secret1/..data
/tmp/secret1/halt
/tmp/secret1/..2019_12_08_12_15_39.463036797
/tmp/secret1/..2019_12_08_12_15_39.463036797/halt

➜ candidate@cka2560:~$ k -n secret exec secret-pod -- cat /tmp/secret1/halt
#! /bin/sh
### BEGIN INIT INFO
# Provides:          halt
# Required-Start:
# Required-Stop:
# Default-Start:
# Default-Stop:      0
# Short-Description: Execute the halt command.
# Description:
...
```

Question 12 | Schedule Pod on Controlplane Nodes

Solve this question on: `ssh cka5248`

Create a *Pod* of image `httpd:2-alpine` in *Namespace* `default`.

The *Pod* should be named `pod1` and the container should be named `pod1-container`.

This *Pod* should **only** be scheduled on controlplane nodes.

Do **not** add new labels to any nodes.

Answer:

First we find the controlplane node(s) and their taints:

```
➜ ssh cka5248

➜ candidate@cka5248:~$ k get node
NAME            STATUS   ROLES           AGE   VERSION
cka5248         Ready    control-plane   90m   v1.33.1
cka5248-node1   Ready    <none>          85m   v1.33.1

➜ candidate@cka5248:~$ k describe node cka5248 | grep Taint -A1
Taints:             node-role.kubernetes.io/control-plane:NoSchedule
Unschedulable:      false

➜ candidate@cka5248:~$ k get node cka5248 --show-labels
NAME      STATUS   ROLES           AGE   VERSION   LABELS
cka5248   Ready    control-plane   91m   v1.33.1
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=ck
a5248,kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node.kubernetes.io/exclude-from-
external-load-balancers=
```

Next we create the *Pod* yaml:

```
➜ candidate@cka5248:~$ k run pod1 --image=httpd:2-alpine --dry-run=client -o yaml > 12.yaml
```

Solution using NodeSelector

Use the K8s docs and search for tolerations and nodeSelector to find examples, then update:

```yaml
# cka5248:/home/candidate/12.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: httpd:2-alpine
    name: pod1-container                    # change
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  tolerations:                              # add
  - effect: NoSchedule                      # add
    key: node-role.kubernetes.io/control-plane # add
  nodeSelector:                             # add
    node-role.kubernetes.io/control-plane: ""  # add
```

```
status: {}
```

ℹ️ The nodeSelector specifies `node-role.kubernetes.io/control-plane` with no value because this is a key-only label and we want to match regardless of the value

Important here to add the toleration for running on controlplane nodes, but also the nodeSelector to make sure it **only** runs on controlplane nodes. If we just specify a toleration the *Pod* can be scheduled on controlplane or worker nodes.

Solution using NodeAffinity

We could also use nodeAffinity instead of nodeSelector, although in this case it is more complex and not really suggested:

```
# cka5248:/home/candidate/12.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: httpd:2-alpine
    name: pod1-container                        # change
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  tolerations:                                  # add
  - effect: NoSchedule                          # add
    key: node-role.kubernetes.io/control-plane # add
  affinity:                                             # add
    nodeAffinity:                                       # add
      requiredDuringSchedulingIgnoredDuringExecution:   # add
        nodeSelectorTerms:                              # add
        - matchExpressions:                             # add
          - key: node-role.kubernetes.io/control-plane # add
            operator: Exists                            # add
status: {}
```

Using nodeAffinity still requires the toleration.

Verify

Now we create the *Pod* and and check if is scheduled:

```
➜ candidate@cka5248:~$ k -f 12.yaml create
pod/pod1 created

➜ candidate@cka5248:~$ k get pod pod1 -o wide
NAME   READY   STATUS    ...   NODE      NOMINATED NODE   READINESS GATES
pod1   1/1     Running   ...   cka5248   <none>           <none>
```

We can see the *Pod* is scheduled on the controlplane node.

Question 13 | Multi Containers and Pod shared Volume

Solve this question on: `ssh cka3200`

Create a *Pod* with multiple containers named `multi-container-playground` in *Namespace* `default`:

- It should have a volume attached and mounted into each container. The volume shouldn't be persisted or shared with other *Pods*
- Container `c1` with image `nginx:1-alpine` should have the name of the node where its *Pod* is running on available as environment variable `MY_NODE_NAME`
- Container `c2` with image `busybox:1` should write the output of the `date` command every second in the shared volume into file `date.log`. You can use `while true; do date >> /your/vol/path/date.log; sleep 1; done` for this.
- Container `c3` with image `busybox:1` should constantly write the content of file `date.log` from the shared volume to stdout. You can use `tail -f /your/vol/path/date.log` for this.

ℹ️ Check the logs of container `c3` to confirm correct setup

Answer:

First we create the *Pod* template:

```
➜ ssh cka3200

➜ candidate@cka3200:~$ k run multi-container-playground --image=nginx:1-alpine --dry-run=client -o yaml > 13.yaml
```

And add the other containers and the commands they should execute:

```
# cka3200:/home/candidate/13.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: multi-container-playground
  name: multi-container-playground
spec:
  containers:
  - image: nginx:1-alpine
    name: c1                                                                    # change
    resources: {}
    env:                                                                        # add
    - name: MY_NODE_NAME                                                        # add
      valueFrom:                                                                # add
        fieldRef:                                                               # add
          fieldPath: spec.nodeName                                              # add
    volumeMounts:                                                               # add
    - name: vol                                                                 # add
      mountPath: /vol                                                           # add
  - image: busybox:1                                                            # add
    name: c2                                                                    # add
    command: ["sh", "-c", "while true; do date >> /vol/date.log; sleep 1; done"]  # add
    volumeMounts:                                                               # add
```

```
    - name: vol                                                    # add
      mountPath: /vol                                              # add
  - image: busybox:1                                               # add
    name: c3                                                       # add
    command: ["sh", "-c", "tail -f /vol/date.log"]                 # add
    volumeMounts:                                                  # add
    - name: vol                                                    # add
      mountPath: /vol                                              # add
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  volumes:                                                         # add
    - name: vol                                                    # add
      emptyDir: {}                                                 # add
status: {}
```

Well, there was a lot requested here! We check if everything is good with the *Pod*:

```
➜ candidate@cka3200:~$ k -f 13.yaml create
pod/multi-container-playground created

➜ candidate@cka3200:~$ k get pod multi-container-playground
NAME                         READY   STATUS    RESTARTS   AGE
multi-container-playground   3/3     Running   0          47s
```

Not a bad start. Now we check if container `c1` has the requested node name as env variable:

```
➜ candidate@cka3200:~$ k exec multi-container-playground -c c1 -- env | grep MY
MY_NODE_NAME=cka3200
```

And finally we check the logging, which means that `c2` correctly writes and `c3` correctly reads and outputs to stdout:

```
➜ candidate@cka3200:~$ k logs multi-container-playground -c c3
Tue Nov  5 13:41:33 UTC 2024
Tue Nov  5 13:41:34 UTC 2024
Tue Nov  5 13:41:35 UTC 2024
Tue Nov  5 13:41:36 UTC 2024
Tue Nov  5 13:41:37 UTC 2024
Tue Nov  5 13:41:38 UTC 2024
```

Question 14 | Find out Cluster Information

Solve this question on: `ssh cka8448`

You're ask to find out following information about the cluster:

1. How many controlplane nodes are available?
2. How many worker nodes (non controlplane nodes) are available?
3. What is the Service CIDR?
4. Which Networking (or CNI Plugin) is configured and where is its config file?
5. Which suffix will static pods have that run on `cka8448`?

Write your answers into file `/opt/course/14/cluster-info`, structured like this:

```
# /opt/course/14/cluster-info
1: [ANSWER]
2: [ANSWER]
3: [ANSWER]
4: [ANSWER]
5: [ANSWER]
```

Answer:

How many controlplane and worker nodes are available?

```
➜ ssh cka8448

➜ candidate@cka8448:~$ k get node
NAME       STATUS   ROLES           AGE   VERSION
cka8448    Ready    control-plane   71m   v1.33.1
```

We see one controlplane and no worker nodes.

What is the Service CIDR?

```
➜ candidate@cka8448:~$ sudo -i

➜ root@cka8448:~# cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep range
    - --service-cluster-ip-range=10.96.0.0/12
```

Which Networking (or CNI Plugin) is configured and where is its config file?

```
➜ root@cka8448:~# find /etc/cni/net.d/
/etc/cni/net.d/
/etc/cni/net.d/.kubernetes-cni-keep
/etc/cni/net.d/10-weave.conflist
/etc/cni/net.d/87-podman-bridge.conflist

➜ root@cka8448:~# cat /etc/cni/net.d/10-weave.conflist
{
    "cniVersion": "0.3.0",
    "name": "weave",
    "plugins": [
        {
            "name": "weave",
            "type": "weave-net",
            "hairpinMode": true
        },
        {
            "type": "portmap",
            "capabilities": {"portMappings": true},
            "snat": true
        }
    ]
}
```

By default the kubelet looks into `/etc/cni/net.d` to discover the CNI plugins. This will be the same on every controlplane and worker nodes.

Which suffix will static pods have that run on cka8448?

The suffix is the node hostname with a leading hyphen.

Result

The resulting `/opt/course/14/cluster-info` could look like:

```
# /opt/course/14/cluster-info

# How many controlplane nodes are available?
1: 1

# How many worker nodes (non controlplane nodes) are available?
2: 0

# What is the Service CIDR?
3: 10.96.0.0/12

# Which Networking (or CNI Plugin) is configured and where is its config file?
4: Weave, /etc/cni/net.d/10-weave.conflist

# Which suffix will static pods have that run on cka8448?
5: -cka8448
```

Question 15 | Cluster Event Logging

Solve this question on: `ssh cka6016`

1. Write a `kubectl` command into `/opt/course/15/cluster_events.sh` which shows the latest events in the whole cluster, ordered by time (`metadata.creationTimestamp`)
2. Delete the kube-proxy *Pod* and write the events this caused into `/opt/course/15/pod_kill.log` on `cka6016`
3. Manually kill the containerd container of the kube-proxy *Pod* and write the events into `/opt/course/15/container_kill.log`

Answer:

Step 1

```
➜ ssh cka6016

➜ candidate@cka6016:~$ vim /opt/course/15/cluster_events.sh
```

```
# cka6016:/opt/course/15/cluster_events.sh
kubectl get events -A --sort-by=.metadata.creationTimestamp
```

And we can execute it which should show recent events:

```
➜ candidate@cka6016:~$ sh /opt/course/15/cluster_events.sh
NAMESPACE     LAST SEEN   TYPE     REASON           OBJECT                         MESSAGE
...
32.0.2:8181: connect: connection refused
default       19m         Normal    Pulled              pod/team-york-board-7d74f8f86c-fvzw5    Successfully
pulled image "httpd:2-alpine" in 4.574s (4.575s including waiting). Image size: 22038396 bytes.
default       19m         Normal    Created             pod/team-york-board-7d74f8f86c-fvzw5    Created
container httpd
default       19m         Normal    Pulled              pod/team-york-board-7d74f8f86c-9fg47    Successfully
pulled image "httpd:2-alpine" in 425ms (4.976s including waiting). Image size: 22038396 bytes.
default       19m         Normal    Started             pod/team-york-board-7d74f8f86c-fvzw5    Started
container httpd
default       19m         Normal    Pulled              pod/team-york-board-7d74f8f86c-xnprt    Successfully
pulled image "httpd:2-alpine" in 711ms (5.685s including waiting). Image size: 22038396 bytes.
default       19m         Normal    Created             pod/team-york-board-7d74f8f86c-xnprt    Created
container httpd
default       19m         Normal    Created             pod/team-york-board-7d74f8f86c-9fg47    Created
container httpd
default       19m         Normal    Started             pod/team-york-board-7d74f8f86c-9fg47    Started
container httpd
default       19m         Normal    Started             pod/team-york-board-7d74f8f86c-xnprt    Started
container httpd
...
```

Step 2

We delete the kube-proxy *Pod*:

```
➜ candidate@cka6016:~$ k -n kube-system get pod -l k8s-app=kube-proxy -owide
NAME               READY   ...     NODE      NOMINATED NODE   READINESS GATES
kube-proxy-lf2fs   1/1     ...     cka6016   <none>           <none>

➜ candidate@cka6016:~$ k -n kube-system delete pod kube-proxy-lf2fs
pod "kube-proxy-lf2fs" deleted
```

Now we can check the events, for example by using the command that we created before:

```
➜ candidate@cka6016:~$ sh /opt/course/15/cluster_events.sh
```

Write the events caused by the deletion into `/opt/course/15/pod_kill.log` on `cka6016`:

```
# cka6016:/opt/course/15/pod_kill.log
kube-system   12s         Normal    Killing            pod/kube-proxy-lf2fs                    Stopping
container kube-proxy
kube-system   12s         Normal    SuccessfulCreate   daemonset/kube-proxy                    Created pod:
kube-proxy-wb4tb
kube-system   11s         Normal    Scheduled          pod/kube-proxy-wb4tb                    Successfully
assigned kube-system/kube-proxy-wb4tb to cka6016
kube-system   11s         Normal    Pulled             pod/kube-proxy-wb4tb                    Container
image "registry.k8s.io/kube-proxy:v1.33.1" already present on machine
kube-system   11s         Normal    Created            pod/kube-proxy-wb4tb                    Created
container kube-proxy
kube-system   11s         Normal    Started            pod/kube-proxy-wb4tb                    Started
container kube-proxy
default       10s         Normal    Starting           node/cka6016
```

Step 3

ℹ️ Node `cka6016` is already the controlplane and the only node of the cluster. Otherwise we might have to ssh onto the correct worker node where the *Pod* is running instead

Finally we will try to provoke events by killing the container belonging to the container of a kube-proxy *Pod*:

```
➜ candidate@cka6016:~$ sudo -i

➜ root@cka6016:~# crictl ps | grep kube-proxy
2fd052f1fcf78       505d571f5fd56       57 seconds ago      Running             kube-proxy                0
                  3455856e0970c       kube-proxy-wb4tb

➜ root@cka6016:~# crictl rm --force 2fd052f1fcf78
2fd052f1fcf78
2fd052f1fcf78

➜ root@cka6016:~# crictl ps | grep kube-proxy
6bee4f36f8410       505d571f5fd56       5 seconds ago       Running             kube-proxy                0
                  3455856e0970c       kube-proxy-wb4tb
```

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.

We killed the container (2fd052f1fcf78), but also noticed that a new container (6bee4f36f8410) was directly created again. Thanks Kubernetes!

Now we see if this caused events again and we write those into the second file:

```
➜ candidate@cka6016:~$ sh /opt/course/15/cluster_events.sh
```

Write the events caused by the killing into `/opt/course/15/container_kill.log` on `cka6016`:

```
# /opt/course/15/container_kill.log
kube-system   21s         Normal    Created            pod/kube-proxy-wb4tb                  Created
container kube-proxy
kube-system   21s         Normal    Started            pod/kube-proxy-wb4tb                  Started
container kube-proxy
default       90s         Normal    Starting           node/cka6016
default       20s         Normal    Starting           node/cka6016
```

Comparing the events we see that when we deleted the whole *Pod* there were more things to be done, hence more events. For example was the *DaemonSet* in the game to re-create the missing *Pod*. Where when we manually killed the main container of the *Pod*, the *Pod* still exists but only its container needed to be re-created, hence less events.

Question 16 | Namespaces and Api Resources

Solve this question on: `ssh cka3200`

Write the names of all namespaced Kubernetes resources (like *Pod*, *Secret*, *ConfigMap*...) into `/opt/course/16/resources.txt`.

Find the `project-*` *Namespace* with the highest number of `Roles` defined in it and write its name and amount of *Roles* into `/opt/course/16/crowded-namespace.txt`.

Answer:

Namespace and Namespaces Resources

We can get a list of all resources:

```
k api-resources      # shows all

k api-resources -h   # a bit of help is always good
```

So we write them into the requested location:

```
➜ ssh cka3200

➜ candidate@cka3200:~$ k api-resources --namespaced -o name > /opt/course/16/resources.txt
```

Which results in the file:

```
# cka3200:/opt/course/16/resources.txt
bindings
configmaps
endpoints
events
limitranges
persistentvolumeclaims
pods
podtemplates
replicationcontrollers
resourcequotas
```

```
secrets
serviceaccounts
services
controllerrevisions.apps
daemonsets.apps
deployments.apps
replicasets.apps
statefulsets.apps
localsubjectaccessreviews.authorization.k8s.io
horizontalpodautoscalers.autoscaling
cronjobs.batch
jobs.batch
leases.coordination.k8s.io
endpointslices.discovery.k8s.io
events.events.k8s.io
ingresses.networking.k8s.io
networkpolicies.networking.k8s.io
poddisruptionbudgets.policy
rolebindings.rbac.authorization.k8s.io
roles.rbac.authorization.k8s.io
csistoragecapacities.storage.k8s.io
```

Namespace with most Roles

```
➜ candidate@cka3200:~$ k -n project-jinan get role --no-headers | wc -l
No resources found in project-jinan namespace.
0

➜ candidate@cka3200:~$ k -n project-miami get role --no-headers | wc -l
300

➜ candidate@cka3200:~$ k -n project-melbourne get role --no-headers | wc -l
2

➜ candidate@cka3200:~$ k -n project-seoul get role --no-headers | wc -l
10

➜ candidate@cka3200:~$ k -n project-toronto get role --no-headers | wc -l
No resources found in project-toronto namespace.
0
```

Finally we write the name and amount into the file:

```
# cka3200:/opt/course/16/crowded-namespace.txt
project-miami with 300 roles
```

Question 17 | Operator, CRDs, RBAC, Kustomize

Solve this question on: `ssh cka6016`

There is Kustomize config available at `/opt/course/17/operator`. It installs an operator which works with different *CRDs*. It has been deployed like this:

```
kubectl kustomize /opt/course/17/operator/prod | kubectl apply -f -
```

Perform the following changes in the Kustomize base config:

1. The operator needs to `list` certain *CRDs*. Check the logs to find out which ones and adjust the permissions for *Role* `operator-role`
2. Add a new *Student* resource called `student4` with any name and description

Deploy your Kustomize config changes to prod.

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 done here for prod:

```
➜ ssh cka6016

➜ candidate@cka6016:~$ cd /opt/course/17/operator

➜ candidate@cka6016:/opt/course/17/operator$ ls
base  prod
```

Investigate Base

Let's investigate the base first for better understanding:

```
➜ candidate@cka6016:/opt/course/17/operator$ k kustomize base
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: classes.education.killer.sh
spec:
  group: education.killer.sh
...
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: students.education.killer.sh
spec:
  group: education.killer.sh
...
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: operator
  namespace: NAMESPACE_REPLACE
...
```

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 Prod

```
➜ candidate@cka6016:/opt/course/17/operator$ k kustomize prod
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: classes.education.killer.sh
spec:
  group: education.killer.sh
...
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: students.education.killer.sh
spec:
  group: education.killer.sh
...
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: operator
  namespace: operator-prod
...
```

We can see that all resources now have `namespace: operator-prod`. Also prod adds the additional label `project_id: prod_7768e94e-88da-4744-9135-f1e7fbb96daf` to the *Deployment*. The rest is taken from base.

Locate Issue

The instructions tell us to check the logs:

```
➜ candidate@cka6016:/opt/course/17/operator$ k -n operator-prod get pod
NAME                        READY   STATUS    RESTARTS   AGE
operator-7f4f58d4d9-v6ftw   1/1     Running   0          6m9s

➜ candidate@cka6016:/opt/course/17/operator$ k -n operator-prod logs operator-7f4f58d4d9-v6ftw
+ true
+ kubectl get students
Error from server (Forbidden): students.education.killer.sh is forbidden: User
"system:serviceaccount:operator-prod:operator" cannot list resource "students" in API group
"education.killer.sh" in the namespace "operator-prod"
+ kubectl get classes
Error from server (Forbidden): classes.education.killer.sh is forbidden: User
"system:serviceaccount:operator-prod:operator" cannot list resource "classes" in API group
"education.killer.sh" in the namespace "operator-prod"
+ sleep 10
+ true
```

We can see that the operator tries to list resources `students` and `classes`. If we look at the *Deployment* we can see that it simply runs `kubectl` commands in a loop:

```
# kubectl -n operator-prod edit deploy operator
apiVersion: apps/v1
kind: Deployment
metadata:
...
  name: operator
  namespace: operator-prod
spec:
...
  template:
...
    spec:
      containers:
      - command:
        - /bin/sh
        - -c
        - |
          set -x
          while true; do
            kubectl get students
            kubectl get classes
            sleep 10
          done
```

Adjust RBAC

Now we need to adjust the existing *Role* `operator-role`. In the Kustomize config directory we find file `rbac.yaml` which we need to edit. Instead of manually editing the Yaml we could also generate it via command line:

```
➜ candidate@cka6016:/opt/course/17/operator$ k -n operator-prod create role operator-role --verb list --
resource student --resource class -oyaml --dry-run=client
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: operator-role
  namespace: operator-prod
rules:
```

```
- apiGroups:
  - education.killer.sh
  resources:
  - students
  - classes
  verbs:
  - list
```

Now we copy&paste it into `rbac.yaml`:

```
➜ candidate@cka6016:/opt/course/17/operator$ vim base/rbac.yaml
```

```
# cka6016:/opt/course/17/operator/base/rbac.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: operator-role
  namespace: default
rules:
- apiGroups:
  - education.killer.sh
  resources:
  - students
  - classes
  verbs:
  - list
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: operator-rolebinding
  namespace: default
subjects:
  - kind: ServiceAccount
    name: operator
    namespace: default
roleRef:
  kind: Role
  name: operator-role
  apiGroup: rbac.authorization.k8s.io
```

And we deploy:

```
➜ candidate@cka6016:/opt/course/17/operator$ kubectl kustomize /opt/course/17/operator/prod | kubectl apply
-f -
customresourcedefinition.apiextensions.k8s.io/classes.education.killer.sh unchanged
customresourcedefinition.apiextensions.k8s.io/students.education.killer.sh unchanged
serviceaccount/operator unchanged
role.rbac.authorization.k8s.io/operator-role configured
rolebinding.rbac.authorization.k8s.io/operator-rolebinding unchanged
deployment.apps/operator unchanged
class.education.killer.sh/advanced unchanged
student.education.killer.sh/student1 unchanged
student.education.killer.sh/student2 unchanged
student.education.killer.sh/student3 unchanged
```

We can see that only the *Role* was configured, which is what we want. And the logs are not throwing errors any more:

```
➜ candidate@cka6016:/opt/course/17/operator$ k -n operator-prod logs operator-7f4f58d4d9-v6ftw
+ kubectl get students
NAME       AGE
student1   22m
student2   22m
student3   22m
+ kubectl get classes
NAME       AGE
advanced   20m
```

Create new Student resource

Finally we need to create a new *Student* resource. Here we can simply copy an existing one in `students.yaml`:

```
➜ candidate@cka6016:/opt/course/17/operator$ vim base/students.yaml
```

```
# cka6016:/opt/course/17/operator/base/students.yaml
...
apiVersion: education.killer.sh/v1
kind: Student
metadata:
  name: student3
spec:
  name: Carol Williams
  description: A student excelling in container orchestration and management
---
apiVersion: education.killer.sh/v1
kind: Student
metadata:
  name: student4
spec:
  name: Some Name
  description: Some Description
```

And we deploy:

```
➜ candidate@cka6016:/opt/course/17/operator$ kubectl kustomize /opt/course/17/operator/prod | kubectl apply
-f -
customresourcedefinition.apiextensions.k8s.io/classes.education.killer.sh unchanged
customresourcedefinition.apiextensions.k8s.io/students.education.killer.sh unchanged
serviceaccount/operator unchanged
role.rbac.authorization.k8s.io/operator-role unchanged
rolebinding.rbac.authorization.k8s.io/operator-rolebinding unchanged
deployment.apps/operator unchanged
class.education.killer.sh/advanced unchanged
student.education.killer.sh/student1 unchanged
student.education.killer.sh/student2 unchanged
student.education.killer.sh/student3 unchanged
student.education.killer.sh/student4 created

➜ candidate@cka6016:/opt/course/17/operator$ k -n operator-prod get student
NAME       AGE
student1   28m
student2   28m
student3   27m
student4   43s
```

Only *Student* `student4` got created, everything else stayed the same.

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.

Privacy Policy / Datenschutz

Terms / AGB

CONTENT

CKS

CKA

CKAD

LFCS

LINKS

Killercoda

Kim Wuestkamp