CKA Simulator Preview Kubernetes 1.20

https://killer.sh

This is a preview of the full CKA Simulator course content.

The full course contains 25 scenarios from all the CKA areas. The course also provides a browser terminal which i replica of the original one. This is great to get used and comfortable before the real exam. After the test session (you stop it early, you'll get access to all questions and their detailed solutions. You'll have 36 hours cluster access means even after the session, once you have the solutions, you can still play around.

The following preview will give you an idea of what the full course will provide. These preview questions are in ad the full course. But the preview questions are part of the same CKA simulation environment which we setup for y to the full course you can solve these too.

The answers provided here assume that you did run the initial terminal setup suggestions as provided in the tips especially:

```
alias k=kubectl

export do="-o yaml --dry-run=client"
```

These questions can be solved in the test environment provided through the CKA Simulator

Preview Question 1

Use context: `kubectl config use-context k8s-c2-AC`

The cluster admin asked you to find out the following information about etcd running on cluster2-master1:

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

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

Finally you're asked to save an etcd snapshot at `/etc/etcd-snapshot.db` on cluster2-master1 and display its sta

Answer:

Find out etcd information

Let's check the nodes:

```
➜ k get node
NAME               STATUS   ROLES    AGE    VERSION
cluster2-master1   Ready    master   89m    v1.20.1
cluster2-worker1   Ready    <none>   87m    v1.20.1

➜ ssh cluster2-master1
```

First we check how etcd is setup in this cluster:

design

faq

```
➜ root@cluster2-master1:~# kubectl -n kube-system get pod
NAME                                       READY   STATUS    RESTARTS   AGE
coredns-66bff467f8-k8f48                   1/1     Running   0          26h
coredns-66bff467f8-rn8tr                   1/1     Running   0          26h
etcd-cluster2-master1                      1/1     Running   0          26h
kube-apiserver-cluster2-master1            1/1     Running   0          26h
kube-controller-manager-cluster2-master1   1/1     Running   0          26h
kube-proxy-qthfg                           1/1     Running   0          25h
kube-proxy-z55lp                           1/1     Running   0          26h
kube-scheduler-cluster2-master1            1/1     Running   1          26h
weave-net-cqdvt                            2/2     Running   0          26h
weave-net-dxzgh                            2/2     Running   1          25h
```

We see its running as a *Pod*, more specific a static *Pod*. So we check for the default kubelet directory for static ma

```
➜ root@cluster2-master1:~# find /etc/kubernetes/manifests/
/etc/kubernetes/manifests/
/etc/kubernetes/manifests/kube-controller-manager.yaml
/etc/kubernetes/manifests/kube-apiserver.yaml
/etc/kubernetes/manifests/etcd.yaml
/etc/kubernetes/manifests/kube-scheduler.yaml

➜ root@cluster2-master1:~# vim /etc/kubernetes/manifests/etcd.yaml
```

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

```
# /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://192.168.102.11:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt              # server certificate
    - --client-cert-auth=true                                      # enabled
    - --data-dir=/var/lib/etcd
    - --initial-advertise-peer-urls=https://192.168.102.11:2380
    - --initial-cluster=cluster2-master1=https://192.168.102.11:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key               # server private key
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.102.11:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://192.168.102.11:2380
    - --name=cluster2-master1
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
...
```

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

```
➜ root@cluster2-master1:~# openssl x509  -noout -text -in /etc/kubernetes/pki/etcd/server.crt |
-A2
        Validity
            Not Before: Sep  4 15:28:39 2020 GMT
            Not After : Sep  4 15:28:39 2021 GMT
```

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

```
# /opt/course/p1/etcd-info.txt
Server private key location: /etc/kubernetes/pki/etcd/server.key
Server certificate expiration date: Sep  4 15:28:39 2021 GMT
Is client certificate authentication enabled: yes
```

Create etcd snapshot

First we try:

```
ETCDCTL_API=3 etcdctl snapshot save /etc/etcd-snapshot.db
```

We get the endpoint also from the yaml. But we need to specify more parameters, all of which we can find the ya above:

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

This worked. Now we can output the status of the backup file:

```
➜ root@cluster2-master1:~# ETCDCTL_API=3 etcdctl snapshot status /etc/etcd-snapshot.db
4d4e953, 7213, 1291, 2.7 MB
```

The status shows:

- Hash: 4d4e953
- Revision: 7213
- Total Keys: 1291
- Total Size: 2.7 MB

Preview Question 2

Use context: `kubectl config use-context k8s-c1-H`

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

Create a new *Pod* named `p2-pod` with two containers, one of image `nginx:1.17-alpine` and one of image `busy` sure the busybox container keeps running for some time.

Create a new *Service* named `p2-service` which exposes that *Pod* internally in the cluster on port 3000->80.

Confirm that kube-proxy is running on all nodes cluster1-master1, cluster1-worker1 and cluster1-worker2 and th iptables.

Write the iptables rules of all nodes belonging the created *Service* `p2-service` into file `/opt/course/p2/iptable`

Finally delete the *Service* and confirm that the iptables rules are gone from all nodes.

Answer:

Create the *Pod*

First we create the *Pod*:

```
# check out export statement on top which allows us to use $do
k run p2-pod --image=nginx:1.17-alpine $do > p2.yaml

vim p2.yaml
```

Next we add the requested second container:

```
# p2.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: p2-pod
  name: p2-pod
  namespace: project-hamster              # add
spec:
  containers:
  - image: nginx:1.17-alpine
    name: p2-pod
  - image: busybox:1.31                   # add
    name: c2                              # add
    command: ["sh", "-c", "sleep 1d"]     # add
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

And we create the *Pod*:

```
k -f p2.yaml create
```

Create the *Service*

Next we create the *Service*:

```
k -n project-hamster expose pod p2-pod --name p2-service --port 3000 --target-port 80
```

This will create a yaml like:

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2020-04-30T20:58:14Z"
  labels:
    run: p2-pod
```

```
  managedFields:
...
    operation: Update
    time: "2020-04-30T20:58:14Z"
  name: p2-service
  namespace: project-hamster
  resourceVersion: "11071"
  selfLink: /api/v1/namespaces/project-hamster/services/p2-service
  uid: 2a1c0842-7fb6-4e94-8cdb-1602a3b1e7d2
spec:
  clusterIP: 10.97.45.18
  ports:
  - port: 3000
    protocol: TCP
    targetPort: 80
  selector:
    run: p2-pod
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

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

```
k -n project-hamster get pod,svc,ep
```

Confirm kube-proxy is running and is using iptables

First we get nodes in the cluster:

```
➜ k get node
NAME               STATUS   ROLES    AGE   VERSION
cluster1-master1   Ready    master   98m   v1.20.1
cluster1-worker1   Ready    <none>   96m   v1.20.1
cluster1-worker2   Ready    <none>   95m   v1.20.1
```

The idea here is to log into every node, find the kube-proxy docker container and check its logs:

```
➜ ssh cluster1-master1

➜ root@cluster1-master1$ docker ps | grep kube-proxy
3b02eb4daf9d        ...    "/usr/local/bin/kube…"   ...         k8s_kube-proxy_kube-proxy...
599c87b891cd        ...    "/pause"                 ...         k8s_POD_kube-proxy-p4jwv...

➜ root@cluster1-master1~# docker logs 3b02eb4daf9d
...
I0429 18:39:58.252984       1 server_others.go:186] Using iptables Proxier.
...
```

This should be repeated on every node and result in the same output `Using iptables Proxier`.

Check kube-proxy is creating iptables rules

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

```
➜ ssh cluster1-master1 iptables-save | grep p2-service
-A KUBE-SEP-6U447UXLLQIKP7BB -s 10.44.0.20/32 -m comment --comment "project-hamster/p2-service:
MASQ
```

```
-A KUBE-SEP-6U447UXLLQIKP7BB -p tcp -m comment --comment "project-hamster/p2-service:" -m tcp -
destination 10.44.0.20:80
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.97.45.18/32 -p tcp -m comment --comment "project-hams
cluster IP" -m tcp --dport 3000 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.97.45.18/32 -p tcp -m comment --comment "project-hamster/p2-service: clu
--dport 3000 -j KUBE-SVC-2A6FNMCK6FDH7PJH
-A KUBE-SVC-2A6FNMCK6FDH7PJH -m comment --comment "project-hamster/p2-service:" -j KUBE-SEP-6U4

➜ ssh cluster1-worker1 iptables-save | grep p2-service
-A KUBE-SEP-6U447UXLLQIKP7BB -s 10.44.0.20/32 -m comment --comment "project-hamster/p2-service:
MASQ
-A KUBE-SEP-6U447UXLLQIKP7BB -p tcp -m comment --comment "project-hamster/p2-service:" -m tcp -
destination 10.44.0.20:80
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.97.45.18/32 -p tcp -m comment --comment "project-hams
cluster IP" -m tcp --dport 3000 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.97.45.18/32 -p tcp -m comment --comment "project-hamster/p2-service: clu
--dport 3000 -j KUBE-SVC-2A6FNMCK6FDH7PJH
-A KUBE-SVC-2A6FNMCK6FDH7PJH -m comment --comment "project-hamster/p2-service:" -j KUBE-SEP-6U4

➜ ssh cluster1-worker2 iptables-save | grep p2-service
-A KUBE-SEP-6U447UXLLQIKP7BB -s 10.44.0.20/32 -m comment --comment "project-hamster/p2-service:
MASQ
-A KUBE-SEP-6U447UXLLQIKP7BB -p tcp -m comment --comment "project-hamster/p2-service:" -m tcp -
destination 10.44.0.20:80
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.97.45.18/32 -p tcp -m comment --comment "project-hams
cluster IP" -m tcp --dport 3000 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.97.45.18/32 -p tcp -m comment --comment "project-hamster/p2-service: clu
--dport 3000 -j KUBE-SVC-2A6FNMCK6FDH7PJH
-A KUBE-SVC-2A6FNMCK6FDH7PJH -m comment --comment "project-hamster/p2-service:" -j KUBE-SEP-6U4
```

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

```
➜ ssh cluster1-master1 iptables-save | grep p2-service >> /opt/course/p2/iptables.txt
➜ ssh cluster1-worker1 iptables-save | grep p2-service >> /opt/course/p2/iptables.txt
➜ ssh cluster1-worker2 iptables-save | grep p2-service >> /opt/course/p2/iptables.txt
```

Delete the *Service* and confirm iptables rules are gone

Delete the *Service*:

```
k -n project-hamster delete svc p2-service
```

And confirm the iptables rules are gone:

```
➜ ssh cluster1-master1 iptables-save | grep p2-service
➜ ssh cluster1-worker1 iptables-save | grep p2-service
➜ ssh cluster1-worker2 iptables-save | grep p2-service
```

Done.

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

Preview Question 3

Use context: `kubectl config use-context k8s-c1-H`

There should be two schedulers on cluster1-master1, but only one is is reported to be running. Write all schedule their status into `/opt/course/p3/schedulers.txt`.

There is an existing *Pod* named `special` in *Namespace* `default` which should be scheduled by the second schec pending state.

Fix the second scheduler. Confirm it's working by checking that *Pod* `special` is scheduled on a node and running

Answer:

Write the scheduler info into file:

```
k -n kube-system get pod --show-labels # find labels
k -n kube-system get pod -l component=kube-scheduler > /opt/course/p3/schedulers.txt
```

The file could look like:

```
# /opt/course/p3/schedulers.txt
NAME                                    READY   STATUS             RESTARTS   AGE
kube-scheduler-cluster1-master1         1/1     Running            0          26h
kube-scheduler-special-cluster1-master1   0/1     CrashLoopBackOff   20         26h
```

Check that *Pod*:

```
➜ k get pod special -o wide
NAME      READY   STATUS    ...     NODE     NOMINATED NODE   READINESS GATES
special   0/1     Pending   ...     <none>   <none>           <none>

➜ k get pod special -o jsonpath="{.spec.schedulerName}{'\n'}"
kube-scheduler-special
```

Seems it has no node assigned because of the scheduler not working.

Fix the Scheduler

First we get the available schedulers:

```
➜ k -n kube-system get pod | grep scheduler
kube-scheduler-cluster1-master1            1/1     Running            0          26h
kube-scheduler-special-cluster1-master1    0/1     CrashLoopBackOff   20         26h
```

It seems both are running as static *Pods* due to their name suffixes. First we check the logs:

```
➜ k -n kube-system logs kube-scheduler-special-cluster1-master1 | grep -i error
Error: unknown flag: --this-is-no-parameter
      --alsologtostderr                  log to standard error as well as files
      --logtostderr                      log to standard error instead of files (default true)
```

Well, it seems there is a unknown parameter set. So we connect into the master node, and check the manifests fi

```
➜ ssh cluster1-master1

➜ root@cluster1-master1:~# vim /etc/kubernetes/manifests/kube-scheduler-special.yaml
```

The kubelet could also have a different manifests directory specified via parameter `--pod-manifest-path` which via `ps aux | grep kubelet` and checking the kubelet systemd config. But in our case it's the default one.

Let's check the schedulers yaml:

```
# /etc/kubernetes/manifests/kube-scheduler-special.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-scheduler-special
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-scheduler
    - --authentication-kubeconfig=/etc/kubernetes/scheduler.conf
    - --authorization-kubeconfig=/etc/kubernetes/scheduler.conf
    - --bind-address=127.0.0.1
    - --port=7776
    - --secure-port=7777
    - --kubeconfig=/etc/kubernetes/kube-scheduler.conf
    - --leader-elect=false
    - --scheduler-name=kube-scheduler-special
    #- --this-is-no-parameter=what-the-hell                # remove this obvious error
    image: k8s.gcr.io/kube-scheduler:v1.20.1
...
```

Changes on static *Pods* are recognised automatically by the kubelet, so we wait shortly and check again (you migh few seconds):

```
➜ root@cluster1-master1:~# kubectl -n kube-system get pod | grep scheduler
kube-scheduler-cluster1-master1            1/1     Running   0          26h
kube-scheduler-special-cluster1-master1    0/1     Error     0          9s
```

Also: we can get a Running state shortly, but it can turn into Error. Check a few times by repeating the command. error, let's check the logs again:

```
➜ root@cluster1-master1:~# kubectl -n kube-system logs kube-scheduler-special-cluster1-master1
...
stat /etc/kubernetes/kube-scheduler.conf: no such file or directory
```

Well, it seems there is a file missing or a wrong path specified for that scheduler. So we check the manifests file a

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-scheduler-special
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-scheduler
    - --authentication-kubeconfig=/etc/kubernetes/scheduler.conf
```

```
    - --authorization-kubeconfig=/etc/kubernetes/scheduler.conf
    - --bind-address=127.0.0.1
    - --port=7776
    - --secure-port=7777
    #- --kubeconfig=/etc/kubernetes/kube-scheduler.conf          # wrong path
    - --kubeconfig=/etc/kubernetes/scheduler.conf               # correct path
    - --leader-elect=false
    - --scheduler-name=kube-scheduler-special
    #- --this-is-no-parameter=what-the-hell
    image: k8s.gcr.io/kube-scheduler:v1.20.1
...
```

Save and check the logs again:

```
➜ root@cluster1-master1:~# kubectl -n kube-system logs kube-scheduler-special-cluster1-master1
...
I0430 21:09:56.783926       1 configmap_cafile_content.go:205] Starting client-ca::kube-system:
apiserver-authentication::client-ca-file
I0430 21:09:56.783963       1 shared_informer.go:197] Waiting for caches to sync for client-ca:
system::extension-apiserver-authentication::client-ca-file
I0430 21:09:56.787005       1 secure_serving.go:178] Serving securely on 127.0.0.1:7777
I0430 21:09:56.787315       1 tlsconfig.go:219] Starting DynamicServingCertificateController
I0430 21:09:56.890232       1 shared_informer.go:204] Caches are synced for client-ca::kube-sys
apiserver-authentication::client-ca-file
I0430 21:09:56.890658       1 shared_informer.go:204] Caches are synced for client-ca::kube-sys
apiserver-authentication::requestheader-client-ca-file
```

Looking better, and the status:

```
➜ root@cluster1-master1:~# kubectl -n kube-system get pod | grep scheduler
kube-scheduler-cluster1-master1            1/1     Running   0          26h
kube-scheduler-special-cluster1-master1    1/1     Running   0          32s
```

Well, I call this beautifully fixed!

Check the *Pod* again

Finally, is the *Pod* running and scheduled on a node?

```
➜ k get pod special -o wide
NAME      READY   STATUS    RESTARTS   ...  NODE               NOMINATED NODE
special   1/1     Running   0          ...  cluster1-worker2   <none>
```

Yes, we did it!

If you have to troubleshoot Kubernetes services in the CKA exam you should first check the logs. Then check its c
parameters for obvious misconfigurations. A good starting point is checking if all paths (to config files or certificat