CKAD Simulator Questions and Answers

Kubernetes 1.34

> **Note:** 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.

Question 1 | Namespaces

Solve this question on instance: `ssh ckad5601`

The DevOps team would like to get the list of all Namespaces in the cluster.

The list can contain other columns like STATUS or AGE.

Save the list to `/opt/course/1/namespaces` on ckad5601.

Answer:

```
k get ns > /opt/course/1/namespaces
```

The content should then look like:

```
# /opt/course/1/namespaces
NAME              STATUS   AGE
default           Active   136m
earth             Active   105m
jupiter           Active   105m
kube-node-lease   Active   136m
kube-public       Active   136m
kube-system       Active   136m
mars              Active   105m
shell-intern      Active   105m
```

Question 2 | Pods

Solve this question on instance: `ssh ckad5601`

Create a single Pod of image `httpd:2.4.41-alpine` in Namespace `default`. The Pod should be named `pod1` and the container should be named `pod1-container`.

Your manager would like to run a command manually on occasion to output the status of that exact Pod. Please write a command that does this into `/opt/course/2/pod1-status-command.sh` on ckad5601. The command should use kubectl.

Answer:

```
k run # help

k run pod1 --image=httpd:2.4.41-alpine --dry-run=client -oyaml > 2.yaml

vim 2.yaml
```

Change the container name in 2.yaml to pod1-container:

```
# 2.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: httpd:2.4.41-alpine
    name: pod1-container # change
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Then run:

```
➜ k create -f 2.yaml
pod/pod1 created

➜ k get pod
NAME   READY   STATUS              RESTARTS   AGE
pod1   0/1     ContainerCreating   0          6s

➜ k get pod
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          30s
```

Next create the requested command:

```
vim /opt/course/2/pod1-status-command.sh
```

The content of the command file could look like:

```
# /opt/course/2/pod1-status-command.sh
kubectl -n default describe pod pod1 | grep -i status:
```

Another solution would be using jsonpath:

```
# /opt/course/2/pod1-status-command.sh
kubectl -n default get pod pod1 -o jsonpath="{.status.phase}"
```

To test the command:

```
➜ sh /opt/course/2/pod1-status-command.sh
Running
```

Question 3 | Job

Solve this question on instance: `ssh ckad7326`

Team Neptune needs a Job template located at `/opt/course/3/job.yaml`. This Job should run image `busybox:1.31.0` and execute `sleep 2 && echo done`. It should be in namespace `neptune`, run a total of 3 times and should execute 2 runs in parallel.

Start the Job and check its history. Each pod created by the Job should have the label `id: awesome-job`. The job should be named `neb-new-job` and the container `neb-new-job-container`.

Answer:

```
k -n neptune create job -h

k -n neptune create job neb-new-job --image=busybox:1.31.0 --dry-
run=client -oyaml -- sh -c "sleep 2 && echo done" > /opt/course/3/job.yaml

vim /opt/course/3/job.yaml
```

Make the required changes in the yaml:

```
# /opt/course/3/job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  creationTimestamp: null
  name: neb-new-job
  namespace: neptune
spec:
  completions: 3          # add
  parallelism: 2          # add
  template:
    metadata:
      creationTimestamp: null
      labels:             # add
        id: awesome-job   # add
    spec:
      containers:
      - command:
        - sh
        - -c
        - sleep 2 && echo done
        image: busybox:1.31.0
        name: neb-new-job-container # update
        resources: {}
      restartPolicy: Never
status: {}
```

Then to create it:

```
k -f /opt/course/3/job.yaml create # namespace already set in yaml hence
not needed
```

Check Job and Pods, you should see two running parallel at most but three in total:

```
➜ k -n neptune get pod,job | grep neb-new-job
pod/neb-new-job-jhq2g              0/1     ContainerCreating   0
4s
pod/neb-new-job-vf6ts              0/1     ContainerCreating   0
4s
job.batch/neb-new-job   0/3           4s         5s

➜ k -n neptune get pod,job | grep neb-new-job
pod/neb-new-job-gm8sz              0/1     Completed           0
12s
pod/neb-new-job-jhq2g              0/1     Completed           0
22s
pod/neb-new-job-vf6ts              0/1     Completed           0
22s
job.batch/neb-new-job   3/3           21s        23s
```

Check history:

```
➜ k -n neptune describe job neb-new-job
...
Events:
  Type    Reason            Age    From            Message
  ----    ------            ----   ----            -------
  Normal  SuccessfulCreate  2m52s  job-controller  Created pod: neb-new-
job-jhq2g
  Normal  SuccessfulCreate  2m52s  job-controller  Created pod: neb-new-
job-vf6ts
  Normal  SuccessfulCreate  2m42s  job-controller  Created pod: neb-new-
job-gm8sz
```

At the age column we can see that two pods run parallel and the third one after that. Just as it was required in the task.

Question 4 | Helm Management

Solve this question on instance: `ssh ckad7326`

Team Mercury asked you to perform some operations using Helm, all in Namespace `mercury`:

1. Delete release `internal-issue-report-apiv1`
2. Upgrade release `internal-issue-report-apiv2` to any newer version of chart `killershell/nginx` available
3. Install a new release `internal-issue-report-apache` of chart `killershell/apache`. The Deployment should have two replicas, set these via Helm-values during install
4. There seems to be a broken release, stuck in `pending-install` state. Find it and delete it

Answer:

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

Helm Release: Installed instance of a Chart

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

Step 1

First we should delete the required release:

```
➜ helm -n mercury ls
NAME                            NAMESPACE    ...   STATUS      CHART
internal-issue-report-apiv1     mercury      ...   deployed    nginx-
18.1.14
internal-issue-report-apiv2     mercury      ...   deployed    nginx-
18.1.14
internal-issue-report-app       mercury      ...   deployed    nginx-
18.1.14

➜ helm -n mercury uninstall internal-issue-report-apiv1
release "internal-issue-report-apiv1" uninstalled
```

Step 2

Next we need to upgrade a release:

```
➜ helm repo list
NAME            URL
killershell     http://localhost:6000

➜ helm repo update
...Successfully got an update from the "killershell" chart repository

➜ helm search repo nginx --versions
NAME                CHART VERSION    DESCRIPTION
killershell/nginx   18.2.0       NGINX Open Source is a...
killershell/nginx   18.1.15      NGINX Open Source is a...
killershell/nginx   18.1.14      NGINX Open Source is a...
killershell/nginx   13.0.0       NGINX Open Source is a...

➜ helm -n mercury upgrade internal-issue-report-apiv2 killershell/nginx
Release "internal-issue-report-apiv2" has been upgraded. Happy Helming!
```

> **INFO:** Also check out `helm rollback` for undoing a helm rollout/upgrade

Step 3

Now we're asked to install a new release, with a customised values setting:

```
➜ helm show values killershell/apache
...
replicaCount: 1
...

➜ helm -n mercury install internal-issue-report-apache killershell/apache
--set replicaCount=2
NAME: internal-issue-report-apache
...
STATUS: deployed

➜ k -n mercury get deploy internal-issue-report-apache
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
internal-issue-report-apache   2/2     2            2           64s
```

Step 4

By default releases in pending-upgrade state aren't listed, but we can show all:

```
➜ helm -n mercury ls -a
NAME                          NAMESPACE   ...  STATUS           CHART
internal-issue-report-apache   mercury     ...  deployed         apache-
11.2.20
internal-issue-report-apiv2    mercury     ...  deployed         nginx-
18.2.0
internal-issue-report-app      mercury     ...  deployed         nginx-
18.1.14
internal-issue-report-daniel   mercury     ...  pending-install  nginx-
18.1.14

➜ helm -n mercury uninstall internal-issue-report-daniel
release "internal-issue-report-daniel" uninstalled
```

Question 5 | ServiceAccount, Secret

Solve this question on instance: `ssh ckad7326`

Team Neptune has its own ServiceAccount named `neptune-sa-v2` in Namespace `neptune`. A coworker needs the token from the Secret that belongs to that ServiceAccount. Write the base64 decoded token to file `/opt/course/5/token` on ckad7326.

Answer:

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

```
k -n neptune get sa # get overview
k -n neptune get secrets # shows all secrets of namespace
k -n neptune get secrets -oyaml | grep annotations -A 1 # shows secrets
with first annotation
```

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

```
➜ k -n neptune describe secret neptune-secret-1
...
Data
====
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6Im5aZFdqZDJ2aGNvQ3BqWHZOR1g1b3pIcm5JZ0hHNWxTZk
wzQnFaaTFad2MifQ...
ca.crt:     1066 bytes
namespace:  7 bytes
```

Copy the token (part under token:) and paste it using vim:

```
vim /opt/course/5/token
```

File `/opt/course/5/token` should contain the token:

```
# /opt/course/5/token
eyJhbGciOiJSUzI1NiIsImtpZCI6Im5aZFdqZDJ2aGNvQ3BqWHZOR1g1b3pIcm5JZ0hHNWxTZk
wzQnFaaTFad2MifQ...
```

Question 6 | ReadinessProbe

Solve this question on instance: `ssh ckad5601`

Create a single Pod named `pod6` in Namespace `default` of image `busybox:1.31.0`. The Pod should have a readiness-probe executing `cat /tmp/ready`. It should initially wait 5 and periodically wait 10 seconds. This will set the container ready only if the file `/tmp/ready` exists.

The Pod should run the command `touch /tmp/ready && sleep 1d`, which will create the necessary file to be ready and then idles. Create the Pod and confirm it starts.

Answer:

```
k run pod6 --image=busybox:1.31.0 --dry-run=client -oyaml --command -- sh
-c "touch /tmp/ready && sleep 1d" > 6.yaml

vim 6.yaml
```

Search for a readiness-probe example on https://kubernetes.io/docs, then copy and alter the relevant section:

```
# 6.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod6
  name: pod6
spec:
  containers:
  - command:
    - sh
    - -c
    - touch /tmp/ready && sleep 1d
    image: busybox:1.31.0
    name: pod6
    resources: {}
    readinessProbe:                                 # add
      exec:                                         # add
        command:                                    # add
        - sh                                        # add
        - -c                                        # add
        - cat /tmp/ready                            # add
      initialDelaySeconds: 5                        # add
      periodSeconds: 10                             # add
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Then:

```
k -f 6.yaml create
```

Running `k get pod6` we should see the job being created and completed:

```
➜ k get pod pod6
NAME   READY   STATUS    RESTARTS   AGE
pod6   1/1     Running   0          15s
```

Question 7 | Pods, Namespaces

Solve this question on instance: `ssh ckad7326`

The board of Team Neptune decided to take over control of one e-commerce webserver from Team Saturn. The administrator who once setup this webserver is not part of the organisation any longer. All information you could get was that the e-commerce system is called `my-happy-shop`.

Search for the correct Pod in Namespace `saturn` and move it to Namespace `neptune`. It doesn't matter if you shut it down and spin it up again, it probably hasn't any customers anyways.

Answer:

Let's see all those Pods:

```
➜ k -n saturn get pod
NAME                READY   STATUS    RESTARTS   AGE
webserver-sat-001   1/1     Running   0          111m
webserver-sat-002   1/1     Running   0          111m
webserver-sat-003   1/1     Running   0          111m
...
```

The Pod names don't reveal any information. We assume the Pod we are searching has a label or annotation with the name `my-happy-shop`, so we search for it:

```
k -n saturn describe pod # describe all pods, then manually look for it

# or do some filtering like this
k -n saturn get pod -o yaml | grep my-happy-shop -A10
```

We see the webserver we're looking for is `webserver-sat-003`:

```
k -n saturn get pod webserver-sat-003 -o yaml > 7_webserver-sat-003.yaml # export
vim 7_webserver-sat-003.yaml
```

Change the Namespace to neptune, also remove the status: section, the token volume, the token volumeMount and the nodeName:

```
# 7_webserver-sat-003.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    description: this is the server for the E-Commerce System my-happy-
shop
  labels:
    id: webserver-sat-003
  name: webserver-sat-003
  namespace: neptune # new namespace here
spec:
  containers:
  - image: nginx:1.16.1-alpine
    imagePullPolicy: IfNotPresent
    name: webserver-sat
  restartPolicy: Always
```

Then we execute:

```
k -n neptune create -f 7_webserver-sat-003.yaml

➜ k -n neptune get pod | grep webserver
webserver-sat-003               1/1     Running            0          22s
```

It seems the server is running in Namespace neptune, so we can do:

```
k -n saturn delete pod webserver-sat-003 --force --grace-period=0
```

Question 8 | Deployment, Rollouts

Solve this question on instance: `ssh ckad7326`

There is an existing Deployment named `api-new-c32` in Namespace `neptune`. A developer did make an update to the Deployment but the updated version never came online. Check the Deployment history and find a revision that works, then rollback to it. Could you tell Team Neptune what the error was so it doesn't happen again?

Answer:

```
k -n neptune get deploy # overview
k -n neptune rollout -h
k -n neptune rollout history -h

➜ k -n neptune rollout history deploy api-new-c32
deployment.extensions/api-new-c32
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl edit deployment api-new-c32 --namespace=neptune
3         kubectl edit deployment api-new-c32 --namespace=neptune
4         kubectl edit deployment api-new-c32 --namespace=neptune
5         kubectl edit deployment api-new-c32 --namespace=neptune
```

Let's check Pod and Deployment status:

```
➜ k -n neptune get deploy,pod | grep api-new-c32
deployment.extensions/api-new-c32    3/3     1            3           141m

pod/api-new-c32-65d998785d-jtmqq    1/1     Running            0
141m
pod/api-new-c32-686d6f6b65-mj2fp    1/1     Running            0
141m
pod/api-new-c32-6dd45bdb68-2p462    1/1     Running            0
141m
pod/api-new-c32-7d64747c87-zh648    0/1     ImagePullBackOff   0
141m
```

Let's check the pod for errors:

```
➜ k -n neptune describe pod api-new-c32-7d64747c87-zh648 | grep -i image
    Image:          ngnix:1-alpine
```

Someone seems to have added a new image with a spelling mistake in the name `ngnix:1-alpine`, that's the reason we can tell Team Neptune!

Now let's revert to the previous version:

```
k -n neptune rollout undo deploy api-new-c32

➜ k -n neptune get deploy api-new-c32
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
api-new-c32   3/3     3            3           146m
```

Question 9 | Pod -> Deployment

Solve this question on instance: `ssh ckad9043`

In Namespace `pluto` there is single Pod named `holy-api`. It has been working okay for a while now but Team Pluto needs it to be more reliable.

Convert the Pod into a Deployment named `holy-api` with 3 replicas and delete the single Pod once done. The raw Pod template file is available at `/opt/course/9/holy-api-pod.yaml`.

In addition, the new Deployment should set `allowPrivilegeEscalation: false` and `privileged: false` for the security context on container level.

Please create the Deployment and save its yaml under `/opt/course/9/holy-api-deployment.yaml` on ckad9043.

Answer:

```
cp /opt/course/9/holy-api-pod.yaml /opt/course/9/holy-api-deployment.yaml
# make a copy!

vim /opt/course/9/holy-api-deployment.yaml
```

Now copy/use a Deployment example yaml and put the Pod's metadata: and spec: into the Deployment's template: section:

```
# /opt/course/9/holy-api-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: holy-api        # name stays the same
  namespace: pluto      # important
spec:
  replicas: 3           # 3 replicas
  selector:
    matchLabels:
      id: holy-api      # set the correct selector
  template:
    metadata:
      labels:
        id: holy-api
      name: holy-api
    spec:
      containers:
      - env:
        - name: CACHE_KEY_1
          value: b&MTCi0=[T66RXm!jO@
        - name: CACHE_KEY_2
          value: PCAILGej5Ld@Q%{Q1=#
        - name: CACHE_KEY_3
          value: 2qz-]2OJlWDSTn_;RFQ
        image: nginx:1.17.3-alpine
        name: holy-api-container
        securityContext:                    # add
          allowPrivilegeEscalation: false   # add
          privileged: false                 # add
        volumeMounts:
        - mountPath: /cache1
          name: cache-volume1
        - mountPath: /cache2
          name: cache-volume2
        - mountPath: /cache3
          name: cache-volume3
      volumes:
      - emptyDir: {}
        name: cache-volume1
      - emptyDir: {}
        name: cache-volume2
      - emptyDir: {}
        name: cache-volume3
```

Next create the new Deployment:

```
k -f /opt/course/9/holy-api-deployment.yaml create
```

Finally delete the single Pod:

```
k -n pluto delete pod holy-api --force --grace-period=0
```

Question 10 | Service, Logs

Solve this question on instance: `ssh ckad9043`

Team Pluto needs a new cluster internal Service. Create a ClusterIP Service named `project-plt-6cc-svc` in Namespace `pluto`. This Service should expose a single Pod named `project-plt-6cc-api` of image `nginx:1.17.3-alpine`, create that Pod as well. The Pod should be identified by label `project: plt-6cc-api`. The Service should use tcp port redirection of `3333:80`.

Finally use for example curl from a temporary nginx:alpine Pod to get the response from the Service. Write the response into `/opt/course/10/service_test.html` on ckad9043. Also check if the logs of Pod `project-plt-6cc-api` show the request and write those into `/opt/course/10/service_test.log` on ckad9043.

Answer:

```
k -n pluto run project-plt-6cc-api --image=nginx:1.17.3-alpine --labels
project=plt-6cc-api
```

Next we create the service:

```
k -n pluto expose pod -h # help

k -n pluto expose pod project-plt-6cc-api --name project-plt-6cc-svc --
port 3333 --target-port 80
```

Check the Service is running:

```
➜ k -n pluto get pod,svc | grep 6cc
pod/project-plt-6cc-api         1/1     Running   0          9m42s
service/project-plt-6cc-svc   ClusterIP   10.31.241.234   <none>
3333/TCP   2m24s
```

Finally we check the connection using a temporary Pod:

```
➜ k run tmp --restart=Never --rm --image=nginx:alpine -i -- curl
http://project-plt-6cc-svc.pluto:3333
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Copy or pipe the html content into `/opt/course/10/service_test.html`.

Also the requested logs:

```
k -n pluto logs project-plt-6cc-api > /opt/course/10/service_test.log
```

Question 11 | Working with Containers

Solve this question on instance: `ssh ckad9043`

There are files to build a container image located at `/opt/course/11/image` on ckad9043. The container will run a Golang application which outputs information to stdout. You're asked to perform the following tasks:

> ℹ️ Run all Docker and Podman commands as user root. Use `sudo docker` and `sudo podman` or become root with `sudo -i`

1. Change the Dockerfile: set ENV variable `SUN_CIPHER_ID` to hardcoded value `5b9c1065-e39d-4a43-a04a-e59bcea3e03f`
2. Build the image using sudo docker, tag it `registry.killer.sh:5000/sun-cipher:v1-docker` and push it to the registry
3. Build the image using sudo podman, tag it `registry.killer.sh:5000/sun-cipher:v1-podman` and push it to the registry
4. Run a container using sudo podman, which keeps running detached in the background, named `sun-cipher` using image `registry.killer.sh:5000/sun-cipher:v1-podman`
5. Write the logs your container `sun-cipher` produces into `/opt/course/11/logs` on ckad9043

Answer:

Dockerfile: list of commands from which an Image can be build **Image:** binary file which includes all data/requirements to be run as a Container **Container:** running instance of an Image **Registry:** place where we can push/pull Images to/from

Step 1

Change the `/opt/course/11/image/Dockerfile` to:

```
# build container stage 1
FROM docker.io/library/golang:1.15.15-alpine3.14
WORKDIR /src
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o bin/app .

# app container stage 2
FROM docker.io/library/alpine:3.12.4
COPY --from=0 /src/bin/app app
# CHANGE NEXT LINE
ENV SUN_CIPHER_ID=5b9c1065-e39d-4a43-a04a-e59bcea3e03f
CMD ["./app"]
```

Step 2

Build and push using Docker:

```
➜ cd /opt/course/11/image
➜ sudo docker build -t registry.killer.sh:5000/sun-cipher:v1-docker .
➜ sudo docker push registry.killer.sh:5000/sun-cipher:v1-docker
```

Step 3

Build and push using Podman:

```
➜ cd /opt/course/11/image
➜ sudo podman build -t registry.killer.sh:5000/sun-cipher:v1-podman .
➜ sudo podman push registry.killer.sh:5000/sun-cipher:v1-podman
```

Step 4

Run container using Podman:

```
➜ sudo podman run -d --name sun-cipher registry.killer.sh:5000/sun-
cipher:v1-podman
```

Step 5

Collect logs:

```
➜ sudo podman logs sun-cipher > /opt/course/11/logs
```

Question 12 | Storage, PV, PVC, Pod volume

Solve this question on instance: `ssh ckad5601`

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

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

Finally create a new Deployment `project-earthflower` in Namespace `earth` which mounts that volume at `/tmp/project-data`. The Pods of that Deployment should be of image `httpd:2.4.41-alpine`.

Answer:

Create the PV:

```
# 12_pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
 name: earth-project-earthflower-pv
spec:
 capacity:
  storage: 2Gi
 accessModes:
  - ReadWriteOnce
 hostPath:
  path: "/Volumes/Data"
```

Create the PVC:

```
# 12_pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: earth-project-earthflower-pvc
  namespace: earth
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
     storage: 2Gi
```

Check that both have the status Bound:

```
➜ k -n earth get pv,pvc
NAME                                  CAPACITY   ACCESS MODES   ...  STATUS
CLAIM
persistentvolume/...earthflower-pv    2Gi        RWO            ...  Bound
...er-pvc
```

Create the Deployment:

```yaml
# 12_dep.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: project-earthflower
  namespace: earth
spec:
  replicas: 1
  selector:
    matchLabels:
      app: project-earthflower
  template:
    metadata:
      labels:
        app: project-earthflower
    spec:
      volumes:                                      # add
      - name: data                                  # add
        persistentVolumeClaim:                      # add
          claimName: earth-project-earthflower-pvc  # add
      containers:
      - image: httpd:2.4.41-alpine
        name: container
        volumeMounts:                               # add
        - name: data                                # add
          mountPath: /tmp/project-data              # add
```

Question 13 | Storage, StorageClass, PVC

Solve this question on instance: `ssh ckad9043`

Team Moonpie, which has the Namespace `moon`, needs more storage. Create a new PersistentVolumeClaim named `moon-pvc-126` in that namespace. This claim should use a new StorageClass `moon-retain` with the provisioner set to `moon-retainer` and the reclaimPolicy set to `Retain`. The claim should request storage of 3Gi, an accessMode of ReadWriteOnce and should use the new StorageClass.

The provisioner `moon-retainer` will be created by another team, so it's expected that the PVC will not boot yet. Confirm this by writing the event message from the PVC into file `/opt/course/13/pvc-126-reason` on ckad9043.

Answer:

Create the StorageClass:

```
# 13_sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: moon-retain
provisioner: moon-retainer
reclaimPolicy: Retain
```

Create the PVC:

```
# 13_pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: moon-pvc-126            # name as requested
  namespace: moon               # important
spec:
  accessModes:
    - ReadWriteOnce             # RWO
  resources:
    requests:
      storage: 3Gi              # size
  storageClassName: moon-retain # uses our new storage class
```

Check the status and get the reason:

```
➜ k -n moon describe pvc moon-pvc-126
...
Events:
Normal  ExternalProvisioning  ...  Waiting for a volume to be created
either by the external provisioner 'moon-retainer' or manually by the
system administrator...
```

Write to file:

```
# /opt/course/13/pvc-126-reason
Waiting for a volume to be created either by the external provisioner
'moon-retainer' or manually by the system administrator. If volume
creation is delayed, please verify that the provisioner is running and
correctly registered.
```

Question 14 | Secret, Secret-Volume, Secret-Env

Solve this question on instance: `ssh ckad9043`

You need to make changes on an existing Pod in Namespace `moon` called `secret-handler`. Create a new Secret `secret1` which contains `user=test` and `pass=pwd`. The Secret's content should be available in Pod secret-handler as environment variables `SECRET1_USER` and `SECRET1_PASS`. The yaml for Pod secret-handler is available at `/opt/course/14/secret-handler.yaml`.

There is existing yaml for another Secret at `/opt/course/14/secret2.yaml`, create this Secret and mount it inside the same Pod at `/tmp/secret2`. Your changes should be saved under `/opt/course/14/secret-handler-new.yaml` on ckad9043. Both Secrets should only be available in Namespace moon.

Answer:

Create secret1:

```
k -n moon create secret generic secret1 --from-literal user=test --from-
literal pass=pwd
```

Create secret2:

```
k -n moon -f /opt/course/14/secret2.yaml create
```

Edit the Pod yaml:

```
# /opt/course/14/secret-handler-new.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    id: secret-handler
  name: secret-handler
  namespace: moon
spec:
  volumes:
  - name: cache-volume1
    emptyDir: {}
  - name: secret2-volume              # add
    secret:                           # add
      secretName: secret2             # add
  containers:
  - name: secret-handler
    image: bash:5.0.11
    args: ['bash', '-c', 'sleep 2d']
    volumeMounts:
    - mountPath: /cache1
      name: cache-volume1
    - name: secret2-volume            # add
      mountPath: /tmp/secret2         # add
    env:
    - name: SECRET_KEY_1
      value: ">8$kH#kj..i8}HImQd{"
    - name: SECRET1_USER              # add
      valueFrom:                      # add
        secretKeyRef:                 # add
          name: secret1               # add
          key: user                   # add
    - name: SECRET1_PASS              # add
      valueFrom:                      # add
        secretKeyRef:                 # add
          name: secret1               # add
          key: pass                   # add
```

Apply:

```
k -f /opt/course/14/secret-handler-new.yaml replace --force --grace-
period=0
```

Verify:

```
➜ k -n moon exec secret-handler -- env | grep SECRET1
SECRET1_USER=test
SECRET1_PASS=pwd
```

Question 15 | ConfigMap, Configmap-Volume

Solve this question on instance: `ssh ckad9043`

Team Moonpie has a nginx server Deployment called `web-moon` in Namespace `moon`. Someone started configuring it but it was never completed. To complete please create a ConfigMap called `configmap-web-moon-html` containing the content of file `/opt/course/15/web-moon.html` under the data key-name `index.html`.

The Deployment web-moon is already configured to work with this ConfigMap and serve its content. Test the nginx configuration for example using curl from a temporary nginx:alpine Pod.

Answer:

```
k -n moon create configmap configmap-web-moon-html --from-
file=index.html=/opt/course/15/web-moon.html
```

After waiting a bit or restarting the Pods:

```
k -n moon rollout restart deploy web-moon
```

Test:

```
➜ k run tmp --restart=Never --rm -i --image=nginx:alpine -- curl
10.44.0.78
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Moon Webpage</title>
</head>
<body>
This is some great content.
</body>
```

Question 16 | Logging sidecar

Solve this question on instance: `ssh ckad7326`

The Tech Lead of Mercury2D decided it's time for more logging. There is an existing container named `cleaner-con` in Deployment `cleaner` in Namespace `mercury`. This container mounts a volume and writes logs into a file called `cleaner.log`.

The yaml for the existing Deployment is available at `/opt/course/16/cleaner.yaml`. Persist your changes at `/opt/course/16/cleaner-new.yaml` on ckad7326 but also make sure the Deployment is running.

Create a sidecar container named `logger-con`, image `busybox:1.31.0`, which mounts the same volume and writes the content of `cleaner.log` to stdout, you can use the `tail -f` command for this.

Answer:

Sidecar containers in K8s are initContainers with `restartPolicy: Always`.

```
# /opt/course/16/cleaner-new.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cleaner
  namespace: mercury
spec:
  replicas: 2
  selector:
    matchLabels:
      id: cleaner
  template:
    metadata:
      labels:
        id: cleaner
    spec:
      volumes:
      - name: logs
        emptyDir: {}
      initContainers:
      - name: init
        image: bash:5.0.11
        command: ['bash', '-c', 'echo init >
/var/log/cleaner/cleaner.log']
        volumeMounts:
        - name: logs
          mountPath: /var/log/cleaner
      - name: logger-con                                          #
add
        image: busybox:1.31.0                                     #
add
        restartPolicy: Always                                     #
add
        command: ["sh", "-c", "tail -f /var/log/cleaner/cleaner.log"]   #
add
        volumeMounts:                                             #
add
        - name: logs                                              #
add
          mountPath: /var/log/cleaner                             #
add
      containers:
      - name: cleaner-con
        image: bash:5.0.11
        args: ['bash', '-c', 'while true; do echo `date`: "remove random
file" >> /var/log/cleaner/cleaner.log; sleep 1; done']
        volumeMounts:
        - name: logs
          mountPath: /var/log/cleaner
```

Check the logs:

```
➜ k -n mercury logs cleaner-576967576c-cqtgx -c logger-con
init
Wed Sep 11 10:45:44 UTC 2099: remove random file
...
```

Question 17 | InitContainer

Solve this question on instance: `ssh ckad5601`

There is a Deployment yaml at `/opt/course/17/test-init-container.yaml`. This Deployment spins up a single Pod of image `nginx:1.17.3-alpine` and serves files from a mounted volume, which is empty right now.

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

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

Answer:

```
# 17_test-init-container.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-init-container
  namespace: mars
spec:
  replicas: 1
  selector:
    matchLabels:
      id: test-init-container
  template:
    metadata:
      labels:
        id: test-init-container
    spec:
      volumes:
      - name: web-content
        emptyDir: {}
      initContainers:                # initContainer start
      - name: init-con
        image: busybox:1.31.0
        command: ['sh', '-c', 'echo "check this out!" > /tmp/web-
content/index.html']
        volumeMounts:
        - name: web-content
          mountPath: /tmp/web-content # initContainer end
      containers:
      - image: nginx:1.17.3-alpine
        name: nginx
        volumeMounts:
        - name: web-content
          mountPath: /usr/share/nginx/html
        ports:
        - containerPort: 80
```

Test:

```
➜ k run tmp --restart=Never --rm -i --image=nginx:alpine -- curl 10.0.0.67
check this out!
```

Question 18 | Service misconfiguration

Solve this question on instance: `ssh ckad5601`

There seems to be an issue in Namespace `mars` where the ClusterIP service `manager-api-svc` should make the Pods of Deployment `manager-api-deployment` available inside the cluster.

You can test this with `curl manager-api-svc.mars:4444` from a temporary nginx:alpine Pod. Check for the misconfiguration and apply a fix.

Answer:

Check endpoints:

```
➜ k -n mars describe service manager-api-svc
Name:              manager-api-svc
...
Endpoints:         <none>
```

No endpoints. Check the Service yaml:

```
k -n mars edit service manager-api-svc
```

The selector was pointing to the Deployment instead of the Pod:

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

After fixing:

```
➜ k -n mars run tmp --restart=Never --rm -i --image=nginx:alpine -- curl -
m 5 manager-api-svc:4444
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Question 19 | Service ClusterIP->NodePort

Solve this question on instance: `ssh ckad5601`

In Namespace `jupiter` you'll find an apache Deployment (with one replica) named `jupiter-crew-deploy` and a ClusterIP Service called `jupiter-crew-svc` which exposes it. Change this service to a NodePort one to make it available on all nodes on port 30100.

Test the NodePort Service using the internal IP of all available nodes and the port 30100 using curl.

Answer:

```
k -n jupiter edit service jupiter-crew-svc
```

```
spec:
  ports:
  - name: 8080-80
    port: 8080
    protocol: TCP
    targetPort: 80
    nodePort: 30100 # add the nodePort
  type: NodePort    # change type
```

Get node IPs and test:

```
➜ k get nodes -o wide
NAME       STATUS   ROLES           AGE   VERSION   INTERNAL-IP      ...
ckad5601   Ready    control-plane   18h   v1.34.1   192.168.100.11   ...

➜ curl 192.168.100.11:30100
<html><body><h1>It works!</h1></body></html>
```

Question 20 | NetworkPolicy

Solve this question on instance: `ssh ckad7326`

In Namespace `venus` you'll find two Deployments named `api` and `frontend`. Both Deployments are exposed inside the cluster using Services. Create a NetworkPolicy named `np1` which restricts outgoing tcp connections from Deployment frontend and only allows those going to Deployment api. Make sure the NetworkPolicy still allows outgoing traffic on UDP/TCP ports 53 for DNS resolution.

Test using: `wget www.google.com` and `wget api:2222` from a Pod of Deployment frontend.

Answer:

> **INFO:** For learning NetworkPolicies check out https://editor.cilium.io

```
# 20_np1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: np1
  namespace: venus
spec:
  podSelector:
    matchLabels:
      id: frontend          # label of the pods this policy should be
applied on
  policyTypes:
  - Egress                  # we only want to control egress
  egress:
  - to:                     # 1st egress rule
    - podSelector:            # allow egress only to pods with api label
        matchLabels:
          id: api
  - ports:                  # 2nd egress rule
    - port: 53                # allow DNS UDP
      protocol: UDP
    - port: 53                # allow DNS TCP
      protocol: TCP
```

Notice that we specify two egress rules. If we specify multiple egress rules then these are connected using a logical OR:

```
allow outgoing traffic if
  (destination pod has label id:api) OR ((port is 53 UDP) OR (port is 53
TCP))
```

Test:

```
➜ k -n venus exec frontend-789cbdc677-c9v8h -- wget -O- -T 5
www.google.de:80
wget: download timed out  # External blocked ✓

➜ k -n venus exec frontend-789cbdc677-c9v8h -- wget -O- api:2222
<html><body><h1>It works!</h1></body></html>  # Internal works ✓
```

Question 21 | Requests and Limits, ServiceAccount

Solve this question on instance: `ssh ckad7326`

Team Neptune needs 3 Pods of image `httpd:2.4-alpine`, create a Deployment named `neptune-10ab` for this. The containers should be named `neptune-pod-10ab`. Each container should have a memory

request of 20Mi and a memory limit of 50Mi.

Team Neptune has its own ServiceAccount `neptune-sa-v2` under which the Pods should run. The Deployment should be in Namespace `neptune`.

Answer:

```
# 21.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: neptune-10ab
  namespace: neptune
spec:
  replicas: 3
  selector:
    matchLabels:
      app: neptune-10ab
  template:
    metadata:
      labels:
        app: neptune-10ab
    spec:
      serviceAccountName: neptune-sa-v2 # add
      containers:
      - image: httpd:2.4-alpine
        name: neptune-pod-10ab  # change
        resources:              # add
          limits:               # add
            memory: 50Mi        # add
          requests:             # add
            memory: 20Mi        # add
```

Verify:

```
➜ k -n neptune get pod | grep neptune-10ab
neptune-10ab-7d4b8d45b-4nzj5   1/1     Running   0          57s
neptune-10ab-7d4b8d45b-lzwrf   1/1     Running   0          17s
neptune-10ab-7d4b8d45b-z5hcc   1/1     Running   0          17s
```

Question 22 | Labels, Annotations

Solve this question on instance: `ssh ckad9043`

Team Sunny needs to identify some of their Pods in namespace `sun`. They ask you to add a new label `protected: true` to all Pods with an existing label `type: worker` or `type: runner`. Also add an

annotation `protected: do not delete this pod` to all Pods having the new label `protected: true`.

Answer:

```
➜ k -n sun get pod --show-labels
NAME           READY   STATUS    RESTARTS   AGE   LABELS
0509649a       1/1     Running   0          25s
type=runner,type_old=messenger
0509649b       1/1     Running   0          24s   type=worker
...
```

Add labels:

```
k -n sun label pod -l type=runner protected=true
k -n sun label pod -l type=worker protected=true

# Or combined:
k -n sun label pod -l "type in (worker,runner)" protected=true
```

Add annotations:

```
k -n sun annotate pod -l protected=true protected="do not delete this pod"
```

Preview Questions

Preview Question 1 | Liveness Probe

Solve this question on instance: `ssh ckad9043`

In Namespace `pluto` there is a Deployment named `project-23-api`. It has been working okay for a while but Team Pluto needs it to be more reliable. Implement a liveness-probe which checks the container to be reachable on port 80. Initially the probe should wait 10, periodically 15 seconds.

The original Deployment yaml is available at `/opt/course/p1/project-23-api.yaml`. Save your changes at `/opt/course/p1/project-23-api-new.yaml` and apply the changes.

Answer:

```
# /opt/course/p1/project-23-api-new.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: project-23-api
  namespace: pluto
spec:
  replicas: 3
  selector:
    matchLabels:
      app: project-23-api
  template:
    metadata:
      labels:
        app: project-23-api
    spec:
      containers:
      - image: httpd:2.4-alpine
        name: httpd
        env:
        - name: APP_ENV
          value: "prod"
        livenessProbe:                  # add
          tcpSocket:                    # add
            port: 80                    # add
          initialDelaySeconds: 10       # add
          periodSeconds: 15             # add
```

Apply and verify:

```
k -f /opt/course/p1/project-23-api-new.yaml apply

➜ k -n pluto describe pod project-23-api-xxx | grep Liveness
    Liveness:   tcp-socket :80 delay=10s timeout=1s period=15s #success=1
#failure=3
```

Preview Question 2 | Deployment and Service

Solve this question on instance: `ssh ckad9043`

Team Sun needs a new Deployment named `sunny` with 4 replicas of image `nginx:1.17.3-alpine` in Namespace `sun`. The Deployment and its Pods should use the existing ServiceAccount `sa-sun-deploy`.

Expose the Deployment internally using a ClusterIP Service named `sun-srv` on port 9999. The nginx containers should run as default on port 80. The management of Team Sun would like to execute a command to check that all Pods are running on occasion. Write that command into file `/opt/course/p2/sunny_status_command.sh`. The command should use kubectl.

Answer:

```
# p2_sunny.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sunny
  namespace: sun
spec:
  replicas: 4                                # change
  selector:
    matchLabels:
      app: sunny
  template:
    metadata:
      labels:
        app: sunny
    spec:
      serviceAccountName: sa-sun-deploy      # add
      containers:
      - image: nginx:1.17.3-alpine
        name: nginx
```

Create Service:

```
k -n sun expose deployment sunny --name sun-srv --port 9999 --target-port
80
```

Status command:

```
# /opt/course/p2/sunny_status_command.sh
kubectl -n sun get deployment sunny
```

Preview Question 3 | Debugging Service

Solve this question on instance: `ssh ckad5601`

Management of EarthAG recorded that one of their Services stopped working. All the information they could give you is that it was located in Namespace `earth` and that it stopped working after the latest rollout.

Find the Service, fix any issues and confirm it's working again. Write the reason of the error into file `/opt/course/p3/ticket-654.txt`.

Answer:

Check resources:

```
➜ k -n earth get all
...
deployment.apps/earth-3cc-web            0/4     4            0
116m
```

Pods not ready. Check why:

```
k -n earth edit deploy earth-3cc-web
```

Found the issue - wrong port for readinessProbe:

```
readinessProbe:
  tcpSocket:
    port: 82   # this port doesn't seem to be right, should be 80
```

Change to port 80, save, and verify:

```
➜ k run tmp --restart=Never --rm -i --image=nginx:alpine -- curl -m 5
earth-3cc-web.earth:6363
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Write reason:

```
# /opt/course/p3/ticket-654.txt
Wrong port for readinessProbe defined!
```

CKAD Tips

Knowledge

- Study all topics as proposed in the curriculum until you feel comfortable with all
- Learn and Study the in-browser scenarios on https://killercoda.com/killer-shell-ckad
- Read this and do all examples: https://kubernetes.io/docs/concepts/cluster-administration/logging
- Understand Rolling Update Deployment including maxSurge and maxUnavailable
- Do 1 or 2 test sessions with this CKAD Simulator

- Be fast and breathe kubectl

CKAD Preparation

- 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

Allowed resources:

- https://kubernetes.io/docs
- https://kubernetes.io/blog
- https://helm.sh/docs

Terminal Handling

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, you can put a task in the background using `Ctrl+z` and pull it back into foreground running command `fg`
- Delete pods fast with: `k delete pod x --grace-period 0 --force`

Vim Settings

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

Vim Tips

- Toggle line numbers: `:set number` or `:set nonumber`
- Jump to line: `Esc :22 + Enter`
- Mark lines: `Esc+V` (then arrow keys)
- Copy marked lines: `y`
- Cut marked lines: `d`
- Paste lines: `p` or `P`
- Indent multiple lines: Mark with `Shift+v`, then `>` or `<`, repeat with `.`

Exam Tips

1. Use `kubectl run --dry-run=client -oyaml` to generate YAML
2. Use `kubectl explain <resource>` for field documentation
3. Set aliases: `alias k=kubectl`

4. Enable kubectl autocompletion
5. Practice imperative commands for speed
6. Always verify resources are running after creation