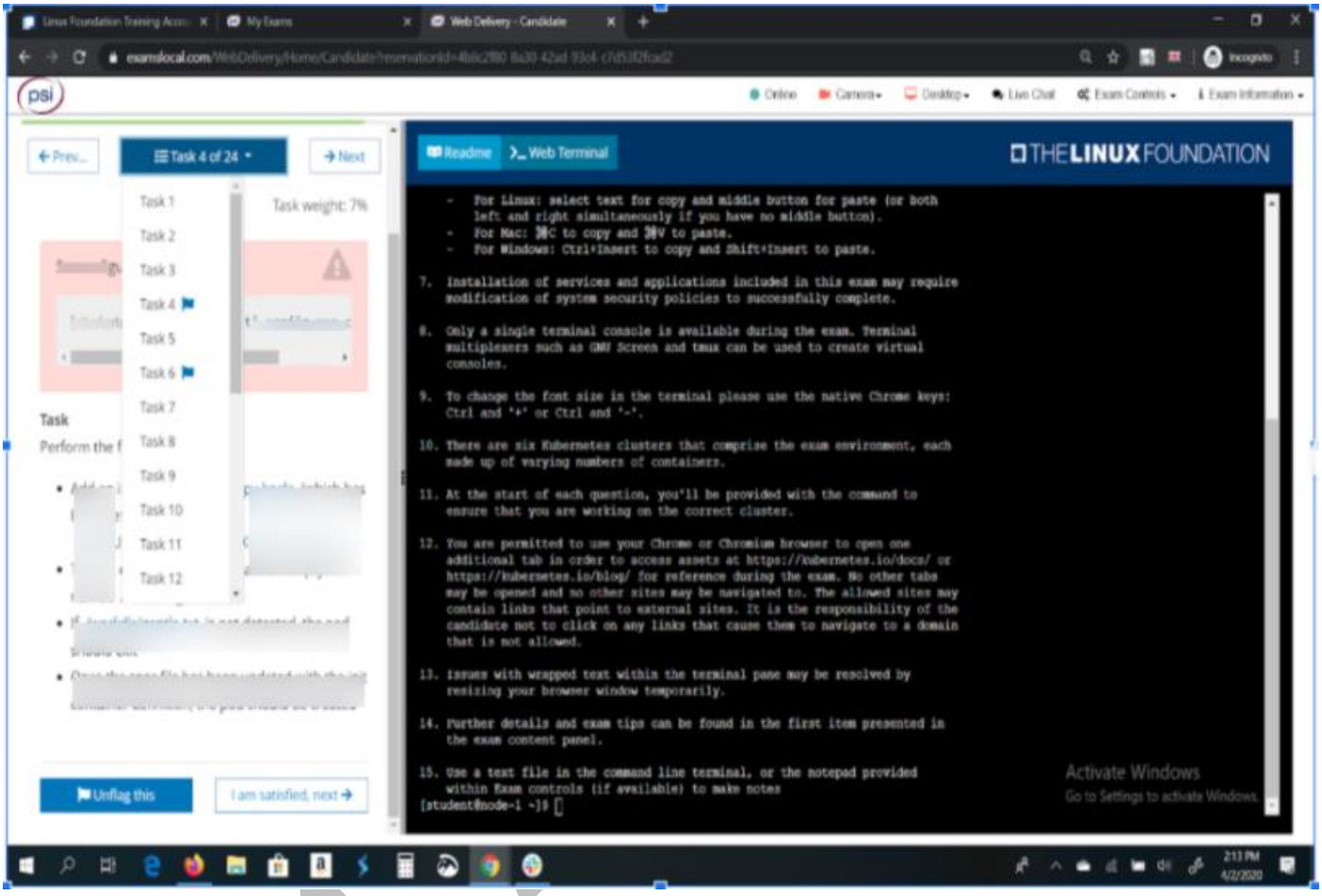
1 CKAD INSTRUCTION

Exam Portal UI:



Linux Foundation Certification Exams User Interface

https://docs.linuxfoundation.org/tc-docs/certification/lf-candidate-handbook/exam-user-interface#exam-console-functions-in-top-menu-bar

https://docs.linuxfoundation.org/tc-docs/certification/lf-candidate-handbook/exam-user-interface#content-panel

https://docs.linuxfoundation.org/tc-docs/certification/lf-candidate-handbook/exam-user-interface#linux-server-terminal

Each question will start with the context switching command like

kubectl config use-context k8s

Note: Use Kubernetes Official Documentation to Create Object Files. https://kubernetes.io/ In Exam.

Note: This exam was built to give you a real exam like feel in terms of your ability to read and interpret a given question, validate your own work, manage time to complete given tasks.

Having said that:

1. Please note that this exam is not a replica of the actual exam.
2. Please note that the questions in this exam are not the same as in the actual exam.
3. Please note that the difficulty level may not be the same as in the actual exam.

Some Important points:

1. Don't panic and try to keep your cool.
2. Always complete the easy questions first. If something is impossible then attempt it when you have completed all other questions and rechecked all your answers.
3. Make Sure you run kubectl config use-context <context> before every question so you can perfrom task on correct cluster.
4. Practice creating resources using imperative commands e.g. **kubectl create --dry-run -o yaml > file.yaml**, kubectl run, kubectl scaleand kubectl expose commands. This helped me to solve the easy questions much faster.
5. Take help from "kubectl create -h" command.
6. Try to use imperative commands as much as possible to save time.
7. Save your time for easy questions
8. Most of the Questions you have to run on Student-1 node. do not run commands on different clusters otherwise answers will not be registered.
9. You can bookmark below URLS for help:
   - https://kubernetes.io/docs/reference/kubectl/cheatsheet/#viewing-finding-resources
   - https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment
   - https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/#static-pod-creation
   - https://kubernetes.io/docs/concepts/services-networking/service/
   - https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#step-two-add-a-nodeselector-field-to-your-pod-configuration
   - https://kubernetes.io/docs/concepts/scheduling-eviction/

- https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
- https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/#upgrade-the-first-control-plane-node
- https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
- https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistent-volumes
- https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims
- https://kubernetes.io/docs/concepts/storage/volumes/
- https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#built-in-snapshot
- https://kubernetes.io/docs/concepts/cluster-administration/logging/#streaming-sidecar-container
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/#clusterrole-example
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/#clusterrolebinding-example
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/#kubectl-create-clusterrole
- https://kubernetes.io/docs/concepts/services-networking/network-policies/#networkpolicy-resource
- https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/
- https://kubernetes.io/docs/concepts/services-networking/ingress/#the-ingress-resource

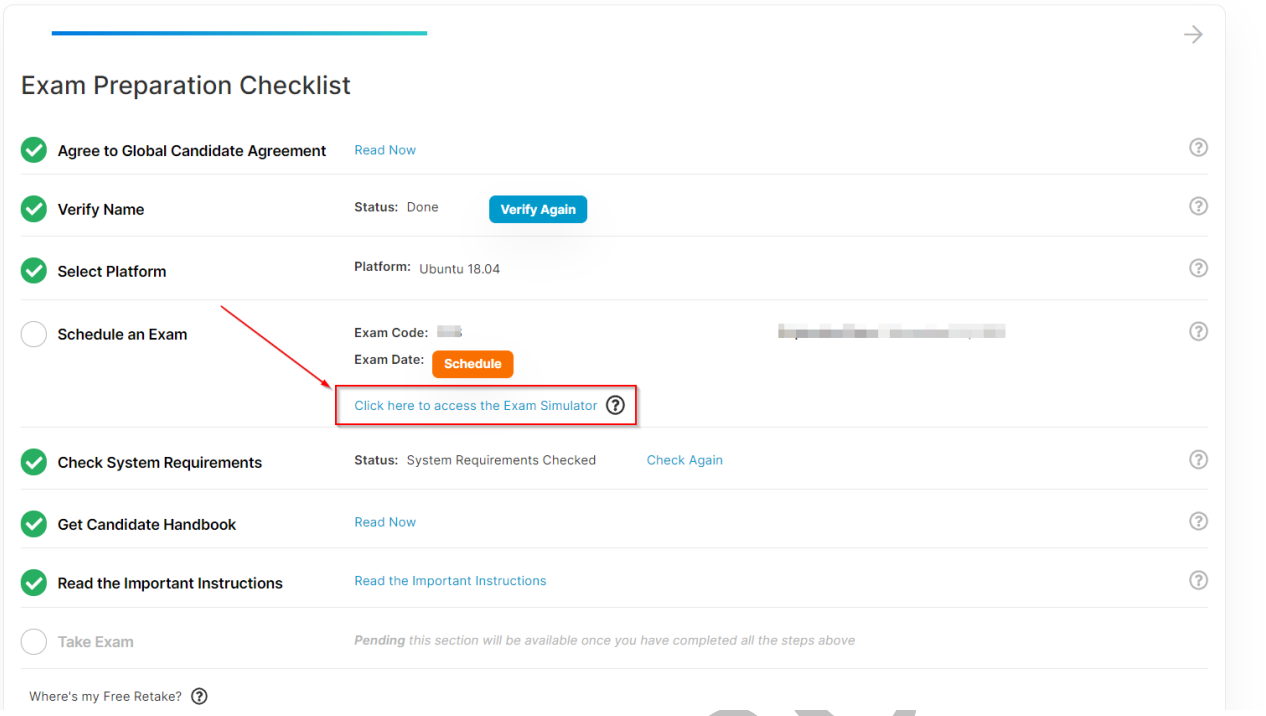
10. Check *Bonus_Prepare_&_Register_For_CKA_&_CKAD_Exam_ed1* PPT from the Portal.

CKAD More Practice Questions and Notes Links:

- ckad-prep-notes
  https://github.com/twajr/ckad-prep-notes
- CKAD Exercises
  https://github.com/dgkanatsios/CKAD-exercises
- Practice Enough With These 150 Questions for the CKAD Exam
  https://medium.com/bb-tutorials-and-thoughts/practice-enough-with-these-questions-for-the-ckad-exam-2f42d1228552

1. Time is very-very limited in exam so do not waste time in exam!!
2. Make Sure you go through all the videos and hands-on guides 2-3 times at least.
3. Make sure you perform all the practice questions available on the portal at least 4-5 times.
4. Make sure your desk and room are clean Proctor can ask you to move your camera multiple times if suspected any activity in the room.
5. Watch this Exam environment demo here: https://www.youtube.com/watch?v=9UqkWcdy140
6. Also Now you have 2 attempts for killer.sh after purchasing the CKA/CKAD/CKS certificate so make sure you use those attempts. Check here: https://www.cncf.io/announcements/2021/06/02/linux-foundation-kubernetes-certifications-now-include-exam-simulator/

Access from here


Exam Preparation Checklist
Agree to Global Candidate Agreement
Read Now
Verify Name
Status: Done
Verify Again
Select Platform
Platform: Ubuntu 18.04
Schedule an Exam
Exam Code:
Exam Date:
Schedule
Click here to access the Exam Simulator
Check System Requirements
Status: System Requirements Checked
Check Again
Get Candidate Handbook
Read Now
Read the Important Instructions
Read the Important Instructions
Take Exam
Pending this section will be available once you have completed all the steps above
Where's my Free Retake?

2 CKAD PRACTICE QUESTION

Q1) Create a secret and mount it as a volume.

Task:

- Create a secret mysecret with values user=myuser and password=mypassword
- Create an redis pod with volume redis-volume and mount secret as volume and put it on the path /etc/foo.

Ans:

https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets-as-environment-variables

Create a secret my-secret

```
$ kubectl create secret generic my-secret --from-literal=username=user --from-literal=password=mypassword
```

Verify the Secret

```
$ kubectl describe secret my-secret
```

Create the pod and use secret as volume

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: redis
    volumeMounts:
    - name: redis-volume
      mountPath: "/etc/cfg"
      readOnly: true
  volumes:
  - name: redis-volume
    secret:
      secretName: mysecret
```

```
$ kubectl create -f nginx-volume.yml
```

Q2) You are required to create a pod that requests a certain amount of CPU and memory, so it gets scheduled to-a node that has those resources available.

- Create a pod named nginx-resources in the pod-resources namespace that requests a minimum of 200m CPU and 1Gi memory for its container

- The pod should use the nginx image

Ans:
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-memory

```
kubectl create ns pod-resources
kubectl run nginx-resources image=nginx -n pod-resources  --dry-run=client -o yaml > nginx-resources.yaml
```

Now edit this file and add environment variable

```
$ vim nginx-resources.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-resources
  namespace: pod-resources
  labels:
    run: nginx-resources
spec:
  containers:
  - name: nginx-resources
    image: nginx
    resources:
      requests:
        valueFrom:
          cpu: 200m
          memory: “1Gi”
$ kubectl create -f nginx-resources.yaml
```

Q3) You are tasked to create a ConfigMap from a file.

Task

- Create an env file file.env with var1=val1
- Create a configmap envcfgmap from this env file and verify the configmap

Ans:

```
$ echo var1=val1 > file.env
$ cat file.env
$ kubectl create cm envcfgmap --from-env-file=file.env
$ kubectl get cm envcfgmap -o yaml –export
```

Q4) Your application's namespace requires a specific service account to be used.

Task

- Create a namespace production
- Create a service account build-robot in production namespace
- Create a deployment app-a in production namespace with nginx image with number of replica 3.
- Update the app-a deployment in the production namespace to run as the restrictedservice service account.

Ans:

```
$ kubectl create ns production
$ kubectl create serviceaccount build-robot -n production
$ kubectl get serviceaccount -n production
$ kubectl create deployment app-a --image=nginx --replicas=3 -n production
$ kubectl get deployment -n production
$ kubectl set serviceaccount deployment app-a restrictedservice -n production
```

Q5) A pod is running on the cluster but it is not responding.

Task

- The desired behavior is to have Kubemetes restart the pod when an endpoint returns an HTTP 500 on the /healthz endpoint. The service, probe-pod, should never send traffic to the pod while it is failing.

Please complete the following:

- The application has an endpoint, /started, that will indicate if it can accept traffic by returning an HTTP 200. If the endpoint returns an HTTP 500, the application has not yet finished initialization.
- The application has another endpoint /healthz that will indicate if the application is still working as expected by returning an HTTP 200. If the endpoint returns an HTTP 500 the application is no longer responsive.
- Configure the probe-pod pod provided to use these endpoints
- The probes should use port 8080

Ans:

Note: Below Yaml file is just to give you overview for that you need application to give you message 200 when application running successfully and 500 when you are getting error.

```
$  vim liveness.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: liveness
    ports:
      - containerPort: 8080
        protocol: TCP
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 5
    readinessProbe:
      httpGet:
        path: /started
        port: 8080
```

```
$ kubectl create -f liveness.yaml
```

Explanation: *when ever we have to restart the pod if health check fails then we implement liveness probe. check type is httpget. To perform a probe, the kubelet sends an HTTP GET request to the server that is running in the container and listening on port 8080. If the handler for the server's /healthz path returns a success code, the kubelet considers the container to be alive and healthy. If the handler returns a failure code, the kubelet kills the container and restarts it.*

Q6) Task

Please complete the following;

- Deploy the counter pod to the cluster using the https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/debug/counter-pod.yaml YAML spec file.
- Create a file /opt/log_output.txt
- Retrieve all currently available application logs from the running pod and store them in the file /opt/log_Output.txt.

Ans:

```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/debug/counter-pod.yaml
$ kubectl get pods
$ kubectl logs counter
$ touch /opt/log_output.txt
$ kubectl logs counter > /opt/log_output.txt
$ cat /opt/log_output.txt
```

Q7) Task

- From the pods running in namespace cpu-stress, write the name only of the pod that is consuming the most CPU to file /opt/file/pod.txt.

Ans:

```
$ kubectl top pods -n cpu-stress
$ echo “max-load-pod” > /opt/file/pod.txt
```

Q8) Task

Create a new deployment for running.nginx with the following parameters:

- Create namespace kdpd00201
- Run the deployment in the kdpd00201 namespace.
- Name the deployment frontend and configure with 4 replicas
- Configure the pod with a container image of lfccncf/nginx:1.13.7
- Set an environment variable of NGINX_PORT=8080 and also expose that port for the container above

Ans:

```
$ kubectl create ns kdpd0020
$ kubectl create deployement api --image=lfccncf/nginx: 1.13.7-alpine --replicas=4 -n kdpd00201 --dry-run=client -o yaml > nginx_deployemnet.yml
```

Now edit this file and add environment variable

```
$ vim nginx_deployment.yml
apiVersion: apps/v1
kind: Deployement
metadata:
```

```
  labels:
    app: api
  name: api
  namespace: kdpd00201
spec:
  replicas: 4
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
      - image: lfccncf/nginx:1.13.7-alpine
        name: nginx
        ports:
        - containerPort: 8080
        env:
        - name: NGINX_PORT
          value: “8080”
```

```
$ kubectl create -f nginx_deployment.yml
```

Q9) Task

Please complete the following:

- Create namespace kdpd00202.
- Create deployment app in namespace kdpd00202 with image lfccncf/nginx:1.12
- Update the app deployment in the kdpd00202 namespace with a maxSurge of 5% and a maxUnavailable of 2%
- Perform a rolling update of the app deployment, changing the lfccncf/nginx image version to 1.13
- Roll back the app deployment to the previous version

Ans:

Note: *Get deployment app then under strategy change maxBurge and maxUnavailable*

```
$ kubectl create namespace kdpd00202
$ kubectl create deployment app --image=lfccncf/nginx:1.12 --replicas=2 -n kdpd00202
$ kubectl edit deployement app -n kdpd00202
```

```
        uid: ldfa2527-5c61-46a9-8dd3-e24643d3ce14
        spec:
         progressDeadlineSeconds: 100
```

```
    replicas: 10
    revisionHistoryLimit: 10
    selector:
      matchLabels:
        app: nginx
      strategy:
       rollingUpdate:
        maxBurge: 5%
        maxUnavailable: 2%
      type: RollingUpdate
     template:
      metadata:
       creationTimestamp: null
       labels:
        app: nginx
      spec:
        containers:
        - image: lfccncf/nginx:1.13
          imageFullPolicy: IfNotPresent
          name: nginx
          ports:
          - containerPort: 80
            protocol: TCP
```

```
$ kubectl rollout status deployement app -n kdpd00202
$ kubectl rollout undo deployement app -n kdpd00202
Deployement.apps/app rolled back
$ kubectl rollout status deployement app -n kdpd00202
```

root@master: /home/ubuntu

```
revisionHistoryLimit: 10
selector:
  matchLabels:
    app: app
strategy:
  rollingUpdate:
    maxSurge: 5%
    maxUnavailable: 2%
  type: RollingUpdate
template:
  metadata:
    creationTimestamp: null
    labels:
      app: app
  spec:
    containers:
    - image: Ifccncf/nginx:1.12
      imagePullPolicy: IfNotPresent
      name: nginx
      resources: {}
```

Q10) Task

- Create a deployment kdsn00101-deployment in namespace kdsn00101 with image nginx.
- Add the func=webFrontEnd key/value label to the pod template metadata to identify the pod for the service definition
- Have 4 replicas

Next, create and deploy a service in namespace kdsn00101 that accomplishes the following:

- Exposes the service on TCP port 8080
- is mapped to the pods defined by the specification of kdsn00l01-deployment
- Is of type NodePort has a name of cherry

Ans:

```
$ kubectl create ns kdsn00101
$ kubectl create deployment kdsn00101-deployment --image=nginx -n kdsn00101
$ kubectl edit deployment kdsn00101-deployment -n kdsn00101
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2020-10-09T08:50:33Z"
  generation: 1
  labels:
    app: nginx
  name: kdsn00101-deployment
  namespace: kdsn00101
  resourceVersion: "4786"
  selfLink: /apis/apps/v1/namespaces/kdsn00101/deployments/kdsn00101-deployment
  uid: 8d3ace00-7761-4189-ba10-fbc676c311bf
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
        func: webFrontEnd
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
```

```
$ kubectl get deployment kdsn00101-deployement -n kdsn00101

$ kubectl expose deployment kdsn00101-deployment -n kdsn00101 --type NodePort --port 8080 --name cherry
```

Q11) Task

Follow the steps below to create a pod that will start at a pre-determined time and which runs to completion only once each time it is started:

- Create a YAML formatted Kubernetes manifest at periodic.yaml that runs the following shell command: **date in a single busybox** container.
- The command should run every minute and must complete within **22 seconds** or be terminated oy Kubernetes.

- The Cronjob name and container name should both be **hello**
- Create the resource in the above manifest and verify that the job executes successfully at least once

Ans:

https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/#cronjob

```
$ kubectl create cronjob hello --image=busybox --schedule " * * * * *" --dry-run=client -o yaml > periodic.yaml

$ vim periodic.yaml
```

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  startingDeadlineSeconds: 22
  concurrencyPolicy: Allow
  jobTemplate:
    metadata:
      name: hello
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            args:
            - /bin/sh
            - -c
            - date
          restartPolicy: Never
```

```
$ kubectl create -f periodic.yaml

$ kubectl get cronjob
```

Q12) A deployment is falling on the cluster due to an incorrect image being specified. Locate the deployment, and fix the problem.

Ans:

Note: *troubleshooting question*

Q13) A user has reported an application is unreachable due to a failing livenessProbe.

Task

Perform the following tasks:

- Find the broken pod and store its name and namespace to /opt/KDOB00401/broken.txt in the format: <namespace>/<pod>
  The output file has already been created.

- Store the associated error events to a file /opt/KDOB00401/error.txt, The output file has already been created.You will need to use the -o wide output specifier with your command

- Fix the issue.

Ans:

Note: *troubleshooting question*

For first task please write pod name and namespace in given file in <namespace>/<pod> format.

For second task get events by below command and save events in given file.

```
kubectl get events --all-namespaces  | grep -i $podname >  <file>
```

For third task Fix issue.

Q14) A project that you are working on has a requirement for persistent data to be available.

Task

To facilitate this, perform the following tasks:

- Create a file at /opt/KDSP00101/data/index.html with the content Acct=Finance

- Create a PersistentVolume named **task-pv-volume** using hostPath and allocate **1Gi** to it, specifying that the volume is at **/opt/KDSP00101/data** on the cluster's node. The configuration should specify the access mode of **ReadWriteOnce** . It should define the StorageClass name **exam** for the PersistentVolume , which will be used to bind PersistentVolumeClaim requests to this PersistenetVolume.

- Create a PefsissentVolumeClaim named **task-pv-claim** that requests a volume of at least **100Mi** and specifies an access mode of **ReadWriteOnce**

- Create a pod **myfrontend** that uses the PersistentVolmeClaim as a volume with a label **app: my-storage-app** mounting the resulting volume to a mountPath **/usr/share/nginx/html** inside the pod

Ans:

```
$ echo 'Acct=Finance' > /opt/KDP00101/data/index.html
$ vim pv.yml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: task-pv-volume
spec:
 capacity:
  storage: 1Gi
  accessModes:
  - ReadWriteOnce
  storageClassName: exam
  hostPath:
    path: /optKDsp00101/data
    type: Directory
```

```
$ vim pvc.yml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: task-pv-claim
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 100Mi
  storageClassName: exam
```

```
$ kubectl create -f pv.yml
$ kubectl create -f pvc.yml
$ kubectl get pv
$ kubectl get pvc
$ vim pod.yml
```

```
 apiVersion: v1
 kind: Pod
 metadata:
  name: myfrontend
  labels:
    app: my-storage-app
 spec:
   containers:
   - name: myfrontend
     image: nginx
```

```
  volumneMounts:
  - mountPath: "/user/share/nginx/html"
    name: mypod
volumes:
- name: mypod
  persistentVolumneClaim:
    claimName: task-pv-claim
```

```
$ kubectl create -f pod.yml
```

Q15) Given a container that writes a log file in format A and a container that converts log files from format A to format B, create a deployment that runs both containers such that the log files from the first container are converted by the second container, emitting logs in format B.

Task:

- Create a deployment named deployment-xyz in the default namespace, that:
- Includes a primary lfccncf/busybox:1 container, named logger-dev
- includes a sidecar lfccncf/fluentd:v0.12 container, named adapter-zen
- Mounts a shared volume /tmp/log on both containers, which does not persist when the pod is deleted
- Instructs the logger-dev container to run the command

  ```
  while true; do
  echo “I luv cncf”>> /
  tmp/log/input.log;
  sleep 10;
  done
  ```

  which should output logs to /tmp/log/input.log in plain text format, with example values:
  I luv cncf
  I luv cncf
  I luv cncf

- The adapter-zen sidecar container should read /tmp/log/input.log and output the data to /tmp/log/output.* in Fluentd JSON format.

Note: No knowledge of Fluentd is required to complete this task: all you will need to achieve this is to create the ConfigMap from the spec file provided at /opt/test/fluentd-configmap.yaml, and mount that ConfigMap to /fluentd/etc in the adapter-zen sidecar container.

Ans:

```
$ kubectl create deployement deployement-xyz --image=lfccncf/busybox:1 --dry-run=client -o yaml > deployement_xyz.yml

$ vim deployement_xyz.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: deployment-xyz
  name: deployment-xyz
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deployment-xyz
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: deployment-xyz
    spec:
      containers:
      - image: lfccncf/busybox:1
        name: busybox
        resources: {}
status: {}
```

after updating the deployment file:

```
apiVersion: apps/v1
kind: deployment
metadata:
    labels:
   app: deployement-xyz
  name: deployement-xyz
spec:
  replicas: 1
  selectore:
    matchLabels:
      app: deployement-xyz
template:
  metadata:
   labels:
     app: deployement-xyz
   spec:
    volumes:
    - name: myvol1
      emptyDir: {}
    - name: myvol2
      configMap:
        name: <config name>
    containers:
    - image: lfccncf/busybox:1
      name: logger-dev
```

```
      command: [“/bin/sh”, “-c”, “while true; do echo ‘I luv cncf’ >> /tmp/log/input.log;
sleep 10; done”]
      volumneMounts:
      - name: myvol1
        mountPath: /tmp/log
    - image: lfccncf/fluentd:v0.12
      name: adapter-zen
      command: [“bin/sh”, “-c”,”tail -f /tmp/log/input.log >> /tmp/log/output.log”]
      volumeMounts:
      -  name : myvol1
         mountPath: /tmp/log
      -  name: myvol2
         mountPath: /fluentd/etc
```

```
$ kubectl create -f deployement_xyz.yml
$ kubectl get deployement
```

Explanation: *In question it’s mentioned that no need to create a persistant volume so we have created a empty volume and mounted container on /tmp/log with name myvol1.then in side car container a config volume is mounted at location /fluentd/etc this config is responsible for the logs conversion.*

Q16) Create a new pod called **super-user-pod** with image **busybox:1.28**. Allow the pod to be able to set system_time. **The container should sleep for 4800 seconds**.

Check more on: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

Ans:

```
$ kubectl run super-user-pod --image=busybox:1.28 --dry-run=client -o yaml > super-user-pod.yaml
$ vim super-user-pod.yaml
$ kubectl create -f super-user-pod.yaml
```

```
root@master:~# vim super-user-pod.yaml
root@master:~# kubectl create -f super-user-pod.yaml
pod/super-user-pod created
root@master:~#
```

root@master: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: super-user-pod
spec:
  containers:
  - name: super-user-pod
    image: busybox:1.28
    command: ["sleep","4800"]
    securityContext:
      capabilities:
        add: ["SYS_TIME"]
~
~
~
~
~
~
```

Get a shell into the running Container:

```
kubectl exec -it super-user-pod -- sh
```

In your shell, view the capabilities for process 1:

```
cd /proc/1
cat status
```

Q17) Create a deployment as follows

- Name ngnix-app
- Using container ngnix with version 1.11.10-alpine
- The deployment should contain 3 replicas

Next, deploy the application with new version 1.13.0-alpine, by performing a rolling update, and record that update. Finally, rollback that update to the previous version 1.11.10-alpine

Ans:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 80
~
~
~
~
~
~
```

Vim nginx-deployment.yaml

kubectl apply -f nginx-deployment.yaml --record

kubectl get deployment

kubectl rollout history deployment nginx-deploy

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy   1/1     1            1           2m22s
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl rollout history deployment nginx-deploy
deployment.apps/nginx-deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx-deployment.yaml --record=true

root@kubeadm-master:/home/ubuntu/Kubernetes#
```

kubectl set image deployment/nginx-deploy nginx=1.17 --record

kubectl rollout history deployment nginx-deploy

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl set image deployment/nginx-deploy nginx=1.17 --record
deployment.apps/nginx-deploy image updated
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl rollout history deployment nginx-deploy
deployment.apps/nginx-deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx-deployment.yaml --record=true
2         kubectl set image deployment/nginx-deploy nginx=1.17 --record=true

root@kubeadm-master:/home/ubuntu/Kubernetes#
```

kubectl describe deployment nginx-deploy

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe deployment nginx-deploy
Name:                   nginx-deploy
Namespace:              default
CreationTimestamp:      Mon, 21 Sep 2020 05:34:39 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 2
                        kubernetes.io/change-cause: kubectl set image deployment/nginx-deploy nginx=1.17 --record=true
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 2 total | 1 available | 1 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        1.17
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  nginx-deploy-767cbb69b8 (1/1 replicas created)
NewReplicaSet:   nginx-deploy-649f54f665 (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  3m14s  deployment-controller  Scaled up replica set nginx-deploy-767cbb69b8 to 1
  Normal  ScalingReplicaSet  30s    deployment-controller  Scaled up replica set nginx-deploy-649f54f665 to 1
root@kubeadm-master:/home/ubuntu/Kubernetes#
```

Finally, Rollback:

kubectl rollout undo <deployment> //for rollback

Q18) Create a new service account with the name **pvviewer**. Grant this Service account access to **list** all PersistentVolumes in the cluster by creating an appropriate cluster role called **pvviewer-role** and ClusterRoleBinding called **pvviewer-role-binding**.

Next, create a pod called **pvviewer** with the **image: redis** and **serviceaccount: pvviewer** in the default namespace.

Reference:

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#clusterrole-example

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#kubectl-create-clusterrole

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#clusterrolebinding-example

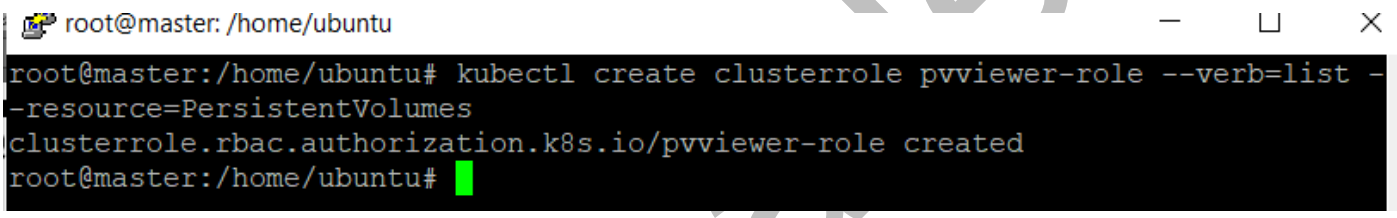
Ans:

Create Service account

```
$ kubectl create serviceaccount pvviewer
```

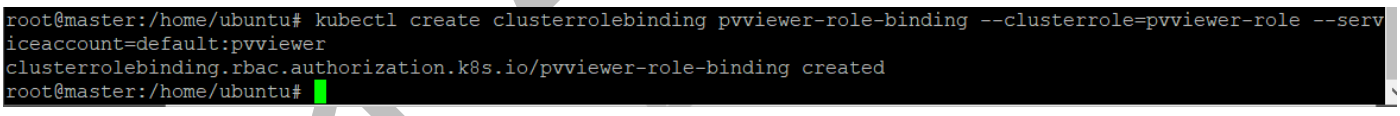
Create cluster role

```
$ kubectl create clusterrole pvviewer-role --verb=list --resource=PersistentVolumes
```



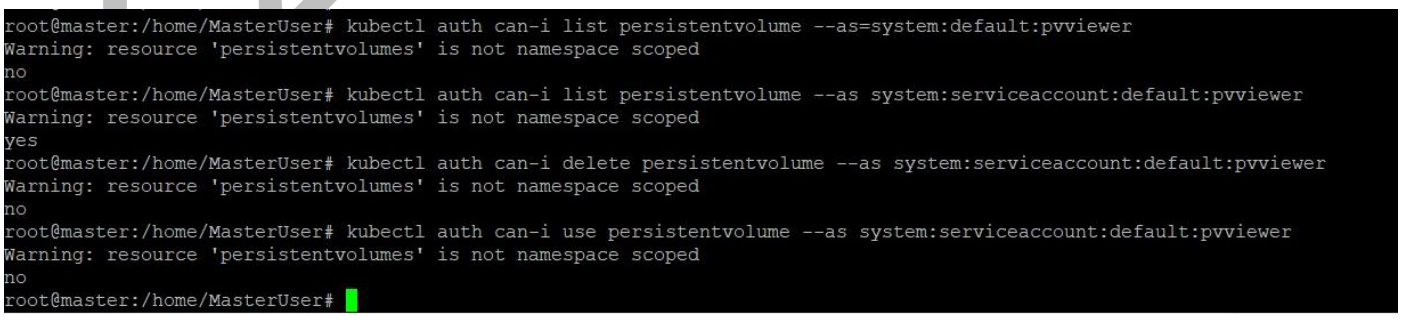
Create cluster role binding

```
$ kubectl create clusterrolebinding pvviewer-role-binding --clusterrole=pvviewer-role --serviceaccount=default:pvviewer
```



Verify//

```
$ kubectl auth can-i list PersistentVolumes --as system:serviceaccount:default:pvviewer
```



Create a Pod using this service account

```
$ vi q17pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pvviewer
spec:
  containers:
  - image: redis
    name: pvviewer
  serviceAccountName: pvviewer
```

```
$ kubectl create -f q17pod.yaml
```

Q19) Create a New NetworkPolicy named all-port that allows Pods in the existing namespace test-net to connect to port 80 of other Pods in same namespace.

Ensure that the new NetworkPolicy:
1. does not allow access to Pods not listening on port 80
2. does not allow access from Pods not in namespace test-net

Ans:

https://kubernetes.io/docs/concepts/services-networking/network-policies/#networkpolicy-resource

```
$ kubectl create ns test-net
$ kubectl label ns test-net project=test-net
$ vim policy.yaml
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: all-port
  namespace: test-net
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector: {}
    ports:
    - port: 80
      protocol: TCP
```

```
$ kubectl create -f policy.yaml
```