ALL CKA EXAM QUESTIONS

Author: Mehul Solanki

Date: 27-OCT-2024

Version: 1.0

Document History

Author	*Date*	*Comments*
Mehul Solanki	27-Oct-2024	This document will be using for exam preparation only.

Contact Details

Name : Mehul Solanki
Phone : +919879729795
Email : er.mehulsolanki1887@gmail.com
LinkedIn : https://in.linkedin.com/in/mehulsolanki1887

Table of Contents

Table of Figures

No table of figures entries found.

1. CKA-QUESTIONS-KODE KLOUD (LIGHTNING LAB – 1)

Q-1. Upgrade the **current version of kubernetes from 1.30.0 to 1.31.0** exactly using the kubeadm utility. Make sure that the upgrade is carried out one node at a time starting with the **controlplane nodes**. To minimize downtime, the deployment gold-nginx should be rescheduled on an alternate node before upgrading each node.
Upgrade **controlplane node** first and drain **node node01** before upgrading it. **Pods for gold-nginx** should run on the **controlplane node** subsequently.

Solutions

MasterNode

Step 1 : cat /etc/*release*
Step 2 : vim /etc/apt/sources.list.d/kubernetes.list (Change version in the file)
Step 3 : apt-get update
Step 4 : apt-cache madison kubeadm
Step 5 : apt-get install kubeadm=1.31.0-1.1
Step 6 : kubeadm version
Step 7 : kubeadm upgrade plan v1.31.0
Step 8 : kubeadm upgrade apply v1.31.0
Step 9 : kubectl get nodes
Step 10 : kubectl drain controlplane --ignore-daemonsets
Step 11 : apt-cache madison kubeadm
Step 12 : kubectl get nodes
Step 13 : apt-get install kubelet=1.31.0-1.1
Step 14 : systemctl daemon-reload
Step 15 : systemctl restart kubelet
Step 16 : kubectl get nodes
Step 17 : kubectl uncordon controlplane

WorkerNode

Step 1 : ssh node01
Step 2 : cat /etc/*release*
Step 3 : vim /etc/apt/sources.list.d/kubernetes.list (Change Version in the file)
Step 4 : apt-get update
Step 5 : apt-cache madison kubeadm
Step 6 : apt-get install kubeadm=1.31.0-1.1
Step 7 : kubeadm upgrade node
Step 8 : kubeadm version
Step 9 : kubectl get nodes
Step 10 : ssh controlplane
Step 11 : kubectl drain node01 --ignore-daemonsets
Step 12 : ssh node01
Step 13 : apt-get install kubelet=1.31.0-1.1
Step 14 : systemctl daemon-reload
Step 15 : systemctl restart kubelet
Step 16 : ssh controlplane
Step 17 : kubectl get nodes
Step 18 : kubectl uncordon node01
Step 19 : kubectl get nodes

Q-2. Print the names of all deployments in the **admin2406** namespace in the following format:
DEPLOYMENT   CONTAINER_IMAGE   READY_REPLICAS   NAMESPACE
<deployment name>   <container image used>   <ready replica count>   <Namespace>
The data should be sorted by the increasing order of the deployment name.
Example: **DEPLOYMENT   CONTAINER_IMAGE   READY_REPLICAS   NAMESPACE**
deploy0   nginx: alpine   1   admin2406 Write the result to the file **/opt/admin2406_data**.

Solutions

Run the below command to get the correct output:

Step 1 : kubectl get deployment -o custom-columns=DEPLOYMENT:.metadata.name,CONTAINER_IMAGE:.spec.template.spec.containers[].image,READY_REPLICAS:.status.readyReplicas,NAMESPACE:.metadata.namespace --sort-by=.metadata.name -n admin2406 > /opt/admin2406_data

Q-3. A kubeconfig file called **admin**. kubeconfig has been created in **/root/CKA**. There is something wrong with the configuration.

- Troubleshoot and fix it.
- Fix **/root/CKA/admin.kubeconfig**

Solutions

Make sure the port for the kube-apiserver is correct. So, for this change port from 4380 to 6443.
Run the below command to know the cluster information:

Step 2 : kubectl cluster-info --kubeconfig /root/CKA/admin.kubeconfig

Q-4. Create a new deployment **called nginx-deploy**, with **image nginx:1.16** and **1 replica**.
Next upgrade the deployment to **version 1.17 using rolling update**.

- Image: **nginx:1.16**
- Task: **Upgrade the version of the deployment to 1:17**

Solutions

Make use of the kubectl create command to create the deployment and explore the --record option while upgrading the deployment image.
Run the below command to create a deployment nginx-deploy:

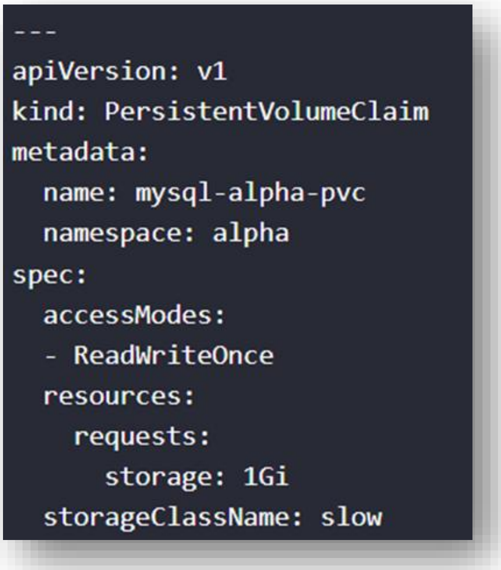
Step 3 : kubectl create deployment nginx-deploy --image=nginx:1.16
Step 4 : kubectl set image deployment nginx-deploy nginx=nginx:1.17 --record
Step 5 : kubectl rollout history deployments nginx-deploy

Q-5. A new deployment called **alpha-mysql** has been deployed in the alpha namespace. However, the pods are not running. Troubleshoot and fix the issue. The deployment should make use of the persistent volume **alpha-pv** to be mounted at **/var/lib/mysql** and should use the environment variable **MYSQL_ALLOW_EMPTY_PASSWORD=1** to make use of an **empty root password**.
Important: **Do not alter the persistent volume**.
Troubleshoot and fix the issues

Solutions

Use the command kubectl describe and try to fix the issue.
Solution manifest file to create a pvc called mysql-alpha-pvc as follows:

Step 1 : kubectl get pod -n alpha
Step 2 : kubectl describe pod alpha-mysql -n alpha
Step 3 : vim pvc.yaml



```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-alpha-pvc
  namespace: alpha
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: slow
```

Step 4 : kubectl apply -f pvc.yaml

Q-6. Take the **backup of ETCD** at the location **/opt/etcd-backup.db** on the controlplane node.
Troubleshoot and fix the issues.

Solutions

Step 1 : cat /etc/kubernetes/manifests/etcd.yaml | grep file
Step 2 : ETCDCTL_API=3 etcdctl -h
Step 3 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /opt/etcd-backup.db
Step 4 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key --write-out=table snapshot status /opt/etcd-backup.db

Q-7. Create a pod called **secret-1401** in the **admin1401** namespace using the **busybox image**. The container within the **pod** should be called **secret-admin** and should sleep for **4800 seconds**.

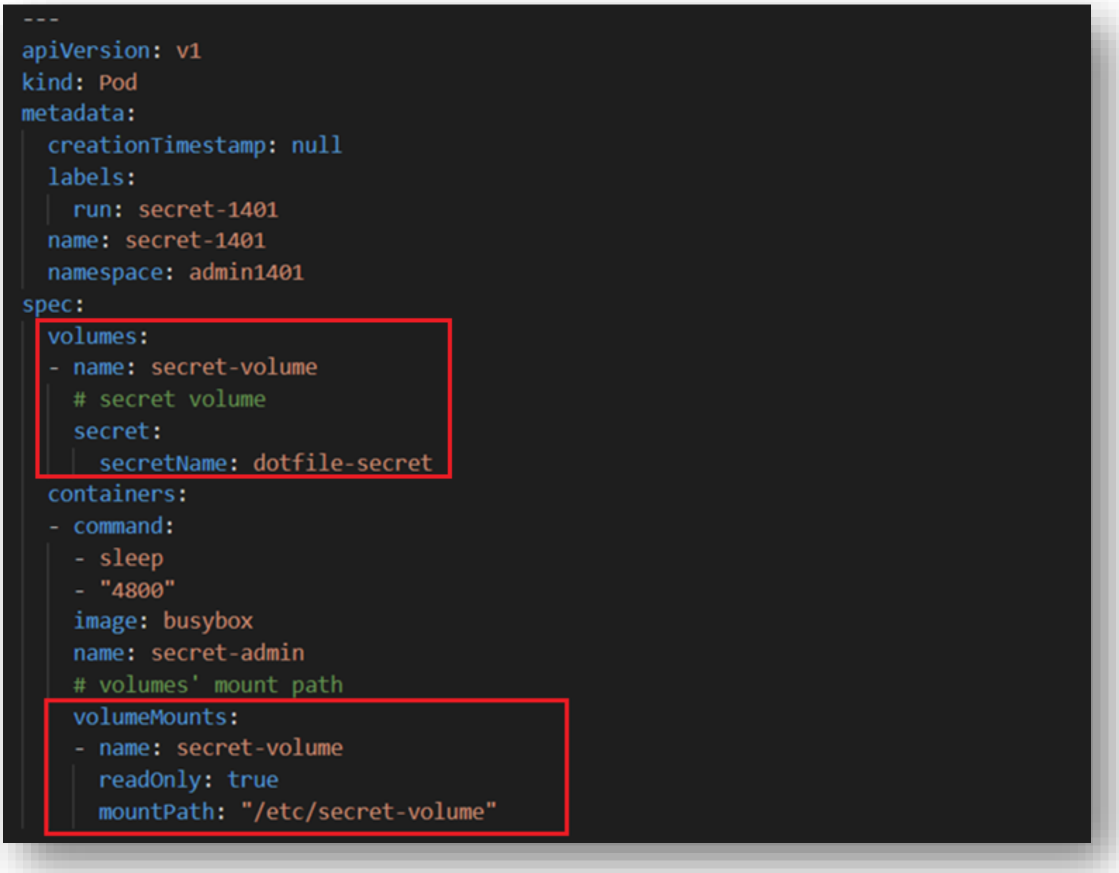
The container should mount a **read-only** secret volume called **secret-volume** at the path **/etc/secret-volume**. The secret being mounted has already been created for you and is called **dotfile-secret**

Solutions

Use the command kubectl run to create a pod definition file. Add secret volume and update container name in it.

Alternatively, run the following command:

Step 1 : kubectl run secret-1401 -n admin1401 --image=busybox --command sleep 4800 --dry-run=client -o yaml > pod.yaml



```
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: secret-1401
  name: secret-1401
  namespace: admin1401
spec:
  volumes:
  - name: secret-volume
    # secret volume
    secret:
      secretName: dotfile-secret
  containers:
  - command:
    - sleep
    - "4800"
    image: busybox
    name: secret-admin
    # volumes' mount path
    volumeMounts:
    - name: secret-volume
      readOnly: true
      mountPath: "/etc/secret-volume"
```

Step 2 : Refer documents and add lines as per above.

Q-1. Deploy a pod named **nginx-pod** using the **nginx:alpine image**.

- Name: **nginx-pod**
- Image: nginx:alpine

Solutions

Step 1 : kubectl run nginx-pod --image=nginx:alpine

Q-2. Deploy a **messaging pod** using the **redis:alpine** image with the labels set to **tier=msg**

- Pod Name: **messaging**
- Image: **redis:alpine**
- Labels: t**ier=msg**

Solutions

Step 1 : kubectl run messaging --image=redis:alpine --labels=tier=msg
Step 2 : kubectl get pods --show-labels

Q-3. Create a **namespace** named **apx-x9984574**.

- Namespace: apx-x9984574

Solutions

Step 1 : kubectl create ns apx-x9984574

Q-4. Get the list of nodes in **JSON** format and store it in a file at **/opt/outputs/nodes-z3444kd9.json.**

Solutions

Step 1 : kubectl get nodes -o Json > /opt/output/nodes-z3444kd9.jason

Q-5. Create a service **messaging-service** to expose the messaging application within the cluster on **port 6379**.

- Service: **messaging-service**
- Port: **6379**
- Type: **ClusterIp**
- **Use the right labels**

Solutions

```
Step 1 :  kubectl expose pod messaging --type=ClusterIP --port=6379 --labels=messaging=service
          --name=messaging-service
Step 2 :  kubectl get svc --show-labels
```

Q-6. Create a deployment named **hr-web-app** using the image **kodekloud/webapp-color** with **2 replicas**.

- Name: **hr-web-app**
- Image: **kodekloud/webapp-color**
- Replicas: **2**

Solutions

```
Step 1 :  kubectl create deployment hr-web-app --image=kodecloud/webapp-color --replicas=2
```

Q-7. Create a static pod named **static-busybox** on the controlplane node that uses the **busybox image** and the command **sleep 1000**.

- Name: static-busybox
- Image: busybox

Solutions

```
Step 1 :  kubectl run static-busybox --image=busybox --command sleep 1000 --dry-run=client -o
          yaml  > /etc/kubernetes/manifests/static-busybox.yaml
```

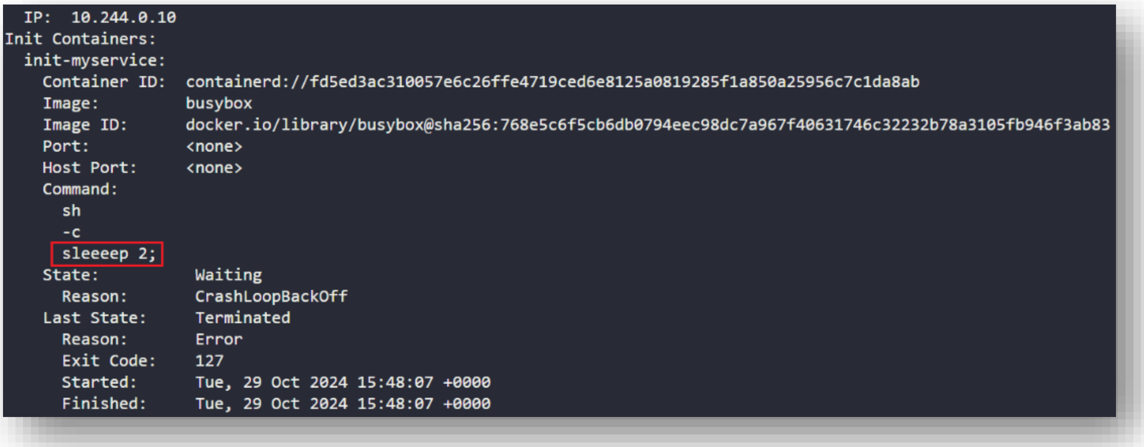
Q-8. Create a **POD** in the finance namespace named **temp-bus** with the image **redis:alpine**.

- Name: **temp-bus**
- Image Name: **redis:alpine**

Solutions

```
Step 1 :  kubectl run temp-bus --image=redis:alpine -n finance
```

Q-9. A new application **orange** is deployed. There is something wrong with it. Identify and fix the issue.



```
  IP:  10.244.0.10
Init Containers:
  init-myservice:
    Container ID:  containerd://fd5ed3ac310057e6c26ffe4719ced6e8125a0819285f1a850a25956c7c1da8ab
    Image:         busybox
    Image ID:      docker.io/library/busybox@sha256:768e5c6f5cb6db0794eec98dc7a967f40631746c32232b78a3105fb946f3ab83
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      sleeeep 2;
    State:          Waiting
      Reason:       CrashLoopBackOff
    Last State:     Terminated
      Reason:       Error
      Exit Code:    127
      Started:      Tue, 29 Oct 2024 15:48:07 +0000
      Finished:     Tue, 29 Oct 2024 15:48:07 +0000
```

Q-10. Expose the **hr-web-app** created in the previous task as a service named **hr-web-app-service**, accessible on **port 30082** on the nodes of the cluster.

- The web application listens on **port 8080**.
- Name: **hr-web-app-service**
- Type: **NodePort**
- Endpoints: **2**
- Port: **8080**
- NodePort: **30082**

Solutions

Step 1 : kubectl expose deploy hr-web-app --type=NodePort --port=8080 --name=hr-web-app-service --dry-run=client -o yaml > hr-web-app-service.yaml

Q-11. Use **JSON PATH** query to retrieve the **osImages** of all the nodes and store it in a file **/opt/outputs/nodes_os_x43kj56.txt**.

- The **osImage** are under the **nodeInfo** section under status of each node.

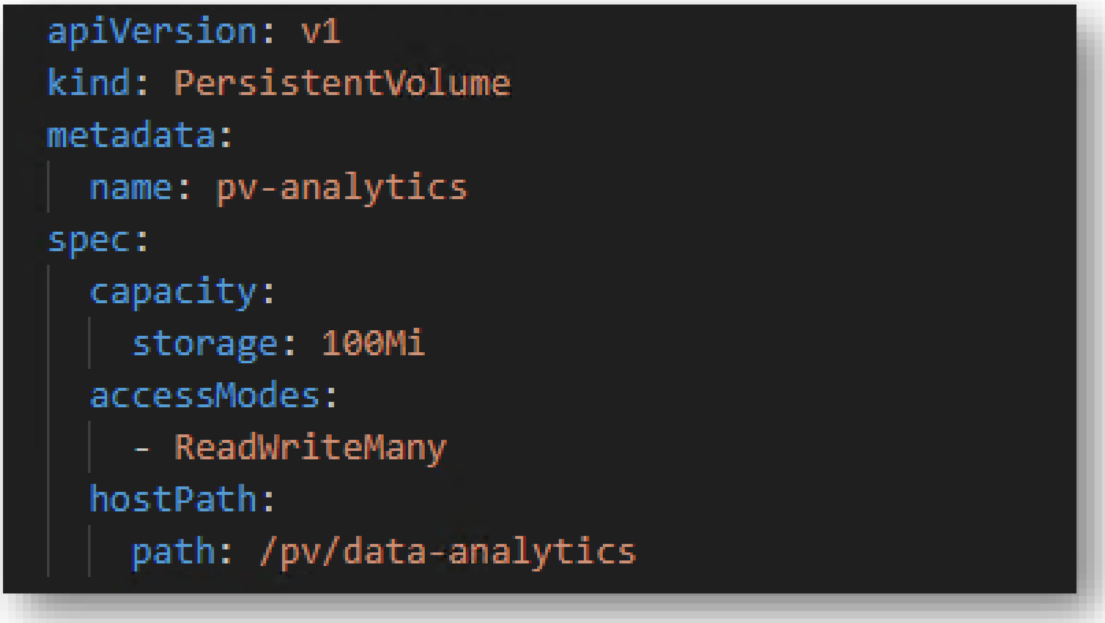
Solutions

Step 1 : kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.osImage}' > /opt/outputs/nodes_os_x43kj56.txt

Q-12. Create a Persistent Volume with the given specification: -

- Volume name: **pv-analytics**
- Storage: **100Mi**
- Access mode: **ReadWriteMany**
- Host path: **/pv/data-analytics**

Solutions



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-analytics
spec:
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /pv/data-analytics
```

Q-1. Take a backup of the **etcd** cluster and save it to **/opt/etcd-backup.db**.

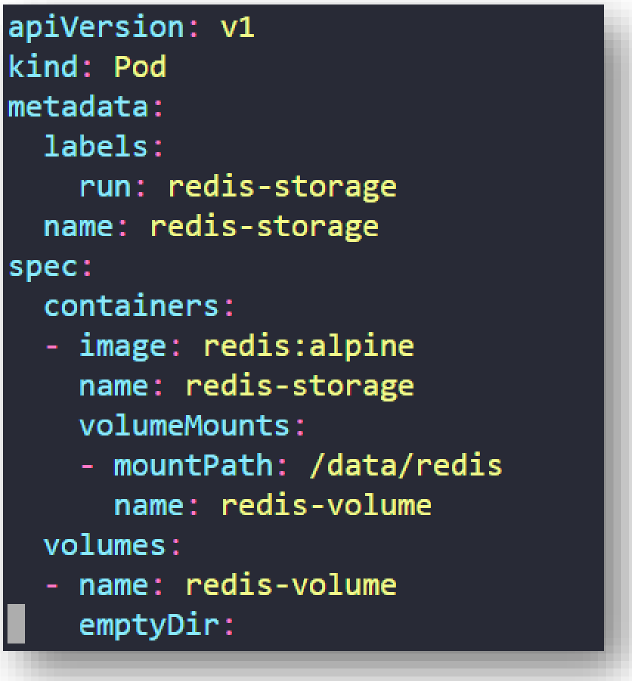
Solutions

Step 1 : cat /etc/kubernetes/manifests/etcd.yaml | grep file
Step 2 : ETCDCTL_API=3 etcdctl -h
Step 3 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /opt/etcd-backup.db
Step 4 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key --write-out=table snapshot status /opt/etcd-backup.db
Step 5 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 snapshot restore --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key --data-dir=/var/lib/etcd-backup /opt/etcd-backup.db
Step 6 : ETCDCTL_API=3 etcdctl snapshot restore --data-dir=/var/lib/etcd-backup /opt/etcd-backup.db

Q-2. Create a Pod called **redis-storage** with **image: redis:alpine** with a Volume of type emptyDir that lasts for the life of the Pod.

- Pod named **'redis-storage'** created
- Pod **'redis-storage'** uses Volume type of **emptyDir**
- Pod **'redis-storage'** uses **volumeMount** with **mountPath = /data/redis**

Solutions



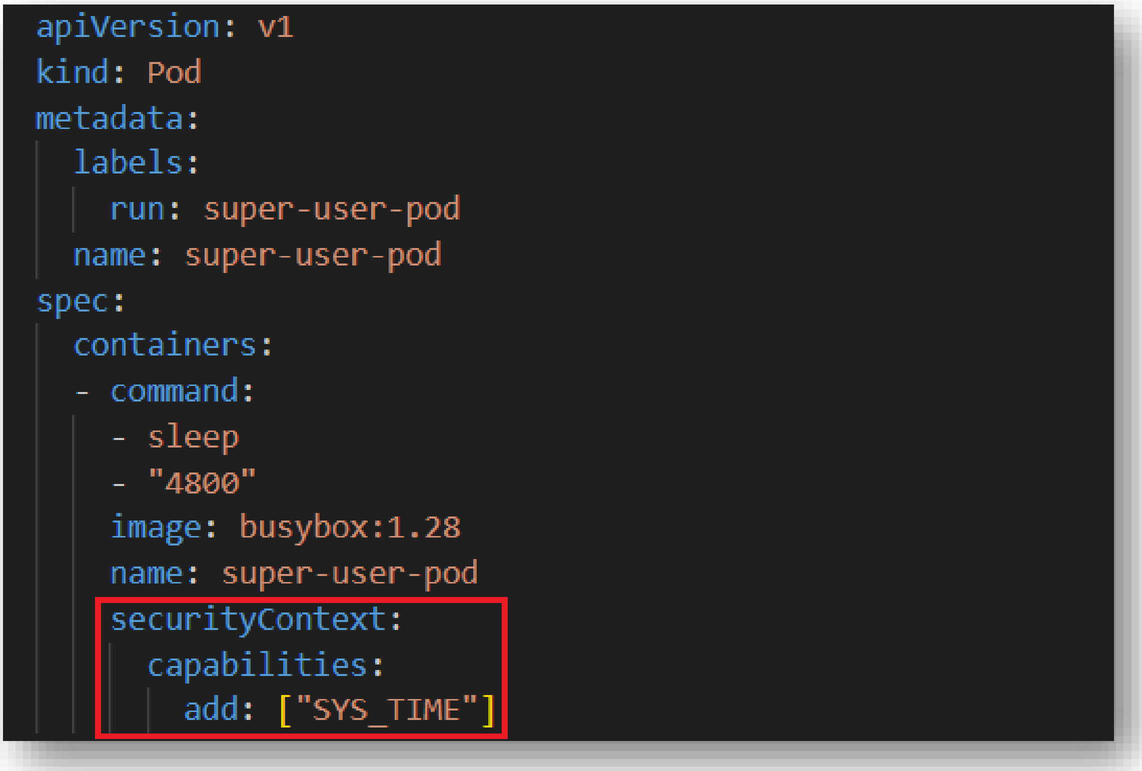
```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: redis-storage
  name: redis-storage
spec:
  containers:
  - image: redis:alpine
    name: redis-storage
    volumeMounts:
    - mountPath: /data/redis
      name: redis-volume
  volumes:
  - name: redis-volume
    emptyDir:
```

Q-3. 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.
- Pod: **super-user-pod**
- Container Image: **busybox:1.28**
- Is **SYS_TIME** capability set for the container?

Solutions

Step 1 : kubectl run super-user-pod --image=busybox:1.28 --command sleep 4800 --dry-run=client -o yaml > server-user-pod.yaml

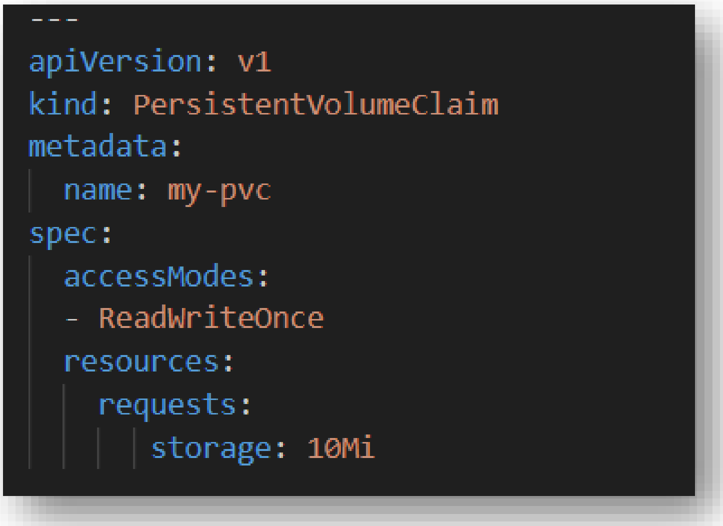


```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: super-user-pod
  name: super-user-pod
spec:
  containers:
  - command:
    - sleep
    - "4800"
    image: busybox:1.28
    name: super-user-pod
    securityContext:
      capabilities:
        add: ["SYS_TIME"]
```

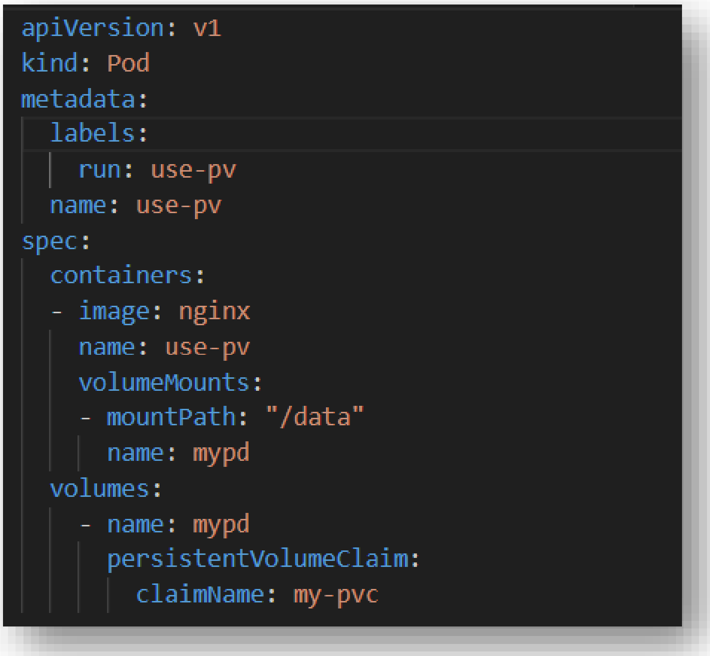
Q-4. A **pod** definition file is created at **/root/CKA/use-pv.yaml**. Make use of this manifest file and mount the persistent volume called **pv-1**. Ensure the pod is running and the **PV** is **bound**.

- mountPath: **/data**
- persistentVolumeClaim Name: **my-pvc**
- **persistentVolume Claim** configured correctly
- pod using the correct **mountPath**
- pod using the **persistent volume claim**?

Solutions



```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
```



```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: use-pv
  name: use-pv
spec:
  containers:
  - image: nginx
    name: use-pv
    volumeMounts:
    - mountPath: "/data"
      name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: my-pvc
```

Q-5. Create a new deployment called **nginx-deploy**, with **image nginx:1.16** and 1 replica. Next upgrade the deployment to **version 1.17** using rolling update.

- Deployment : **nginx-deploy**. Image: **nginx:1.16**
- Image: **nginx:1.16**
- Task: **Upgrade** the version of the deployment to **1:17**
- Task: **Record** the changes for the **image upgrade**

Solutions

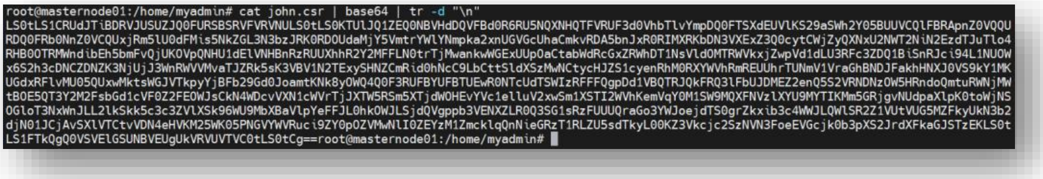
Step 1 : kubectl create deploy nginx-deploy --image=nginx:1.16 --replicas=1
Step 2 : kubectl set image deploy nginx-deploy --image=nginx:1.17 --record
Step 3 : kubectl rollout history deployments nginx-deploy

Q-6. Create a new user called **john**. Grant him access to the cluster. **John** should have permission to create, **list, get, update** and **delete pods** in the development namespace. The private key exists in the location: **/root/CKA/john.key** and csr at **/root/CKA/john.csr**.

- **Important Note**: As of **kubernetes 1.19**, the CertificateSigningRequest object expects a signerName.
- Please refer the documentation to see an example. The documentation tab is available at the top right of terminal.
- CSR: **john-developer** Status:**Approved**
- Role Name: **developer**, namespace: **development**, Resource: **Pods**
- Access: User '**john**' has appropriate permissions

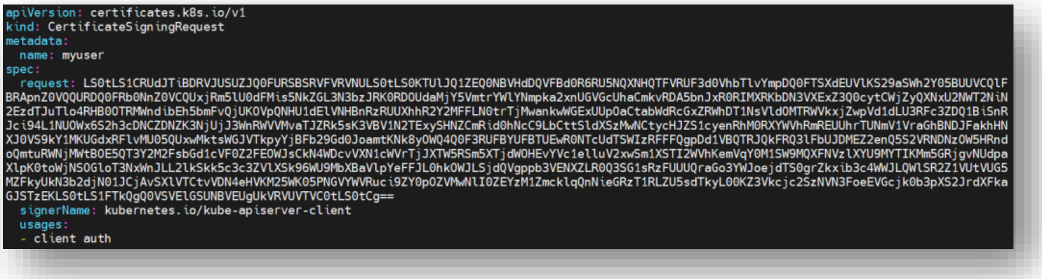
Solutions

Step 1 : openssl genrsa -out john.key 2048
Step 2 : openssl req -new -key john.key -out john.csr -subj "/CN=john"
Step 3 : cat john.csr | base64 | tr -d "\n" (Get the n code value)



```
root@masternode01:/home/myadmin# cat john.csr | base64 | tr -d "\n"
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ1ZEQ0NBVHdDQVFBd0R6RU5NQXNHQTFVRUF3d0VhbTlvYmpDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRApnZ0VQQU
RDQ0FRb0NnZ0VCQUxjRm5lU0dFMis5NkZGL3N3bzJRK0RDOUdaMjY5VmtrYWlYNmpka2xnUGVGcUhaCmkvRDA5bnJxR0RIMXRKbDN3VXExZ3Q0cytCWjZyQXNxU2NWT2NiN2EzdTJuTlo4
RHB0OTRMWndibEh5bmFvQjUKOVpQNHU1dElVNHBnRzRUUXhhR2Y2MFFLN0trTjMwankwWGExUUp0aCtabWdRcGxZRWhDT1NsVldOMTRWVkxjZwpVd1dLU3RFc3ZDQ1BiSnRJci94L1NUOW
x6S2h3cDNCZDNZK3NjUjJ3WnRWVVMvaTJZRk5sK3VBV1N2TExySHNZCmRid0hNcC9LbCttSldXSzMwNCtycHJZS1cyenRhM0RXYWVhRmREUUhrTUNmV1VraGhBNDJFakhHNXJ0VS9kY1MK
UGdxRFlvMU05QUxwMktsWGJVTkpyYjBFb29Gd0JoamtKNk8yOWQ4Q0F3RUFBYUFBTUEwR0NTcUdTSWIzRFFFQgpDd1VBQTRJQkFRQ3lFbUJDMEZ2enQ5S2VRNDNzOW5HRndoQmtuRWNjMW
tBOE5QT3Y2M2FsbGd1cVF0Z2FEOWJsCkN4WDcvVXN1cWVrTjJXTW5RSm5XTjdWOHEvYVc1elluV2xwSm1XSTI2WVhKemVqY0M1SW9MQXFNVzlXYU9MYTIKMm5GRjgvNUdpaXlpK0toWjNS
OGloT3NxWnJLL2lkSkk5c3c3ZVlXSk96WU9MbXBaVlpYeFFJL0hkOWJLSjdQVgppb3VENXZLR0Q3SG1sRzFUUUQraGo3YWJoejdTS0grZkxib3c4WWJLQWlSR2Z1VUtVUG5MZFkyUkN3b2
djN01JCjAvSXlVTCtvVDN4eHVKM25WK05PNGVYWVRuci9ZY0pOZVMwNlI0ZEYzM1ZmcklqQnNieGRzT1RLZU5sdTkyL00KZ3Vkcjc2SzNVN3FoeEVGcjk0b3pXS2JrdXFkaGJSTzEKLS0t
LS1FTkQgQ0VSVElGSUNBVEUgUkVRVUVTVC0tLS0tCg==root@masternode01:/home/myadmin# 
```

Step 4 : Copy above code and put into john-user.yaml
Step 5 : vim john-user.yaml



```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myuser
spec:
  request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ1ZEQ0NBVHdDQVFBd0R6RU5NQXNHQTFVRUF3d0VhbTlvYmpDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlF
BRApnZ0VQQURDQ0FRb0NnZ0VCQUxjRm5lU0dFMis5NkZGL3N3bzJRK0RDOUdaMjY5VmtrYWlYNmpka2xnUGVGcUhaCmkvRDA5bnJxR0RIMXRKbDN3VXExZ3Q0cytCWjZyQXNxU2NWT2NiN
2EzdTJuTlo4RHB0OTRMWndibEh5bmFvQjUKOVpQNHU1dElVNHBnRzRUUXhhR2Y2MFFLN0trTjMwankwWGExUUp0aCtabWdRcGxZRWhDT1NsVldOMTRWVkxjZwpVd1dLU3RFc3ZDQ1BiSnR
Jci94L1NUOWx6S2h3cDNCZDNZK3NjUjJ3WnRWVVMvaTJZRk5sK3VBV1N2TExySHNZCmRid0hNcC9LbCttSldXSzMwNCtycHJZS1cyenRhM0RXYWVhRmREUUhrTUNmV1VraGhBNDJFakhHN
XJ0VS9kY1MKUGdxRFlvMU05QUxwMktsWGJVTkpyYjBFb29Gd0JoamtKNk8yOWQ4Q0F3RUFBYUFBTUEwR0NTcUdTSWIzRFFFQgpDd1VBQTRJQkFRQ3lFbUJDMEZ2enQ5S2VRNDNzOW5HRnd
oQmtuRWNjMWtBOE5QT3Y2M2FsbGd1cVF0Z2FEOWJsCkN4WDcvVXN1cWVrTjJXTW5RSm5XTjdWOHEvYVc1elluV2xwSm1XSTI2WVhKemVqY0M1SW9MQXFNVzlXYU9MYTIKMm5GRjgvNUdpa
XlpK0toWjNSOGloT3NxWnJLL2lkSkk5c3c3ZVlXSk96WU9MbXBaVlpYeFFJL0hkOWJLSjdQVgppb3VENXZLR0Q3SG1sRzFUUUQraGo3YWJoejdTS0grZkxib3c4WWJLQWlSR2Z1VUtVUG5
MZFkyUkN3b2djN01JCjAvSXlVTCtvVDN4eHVKM25WK05PNGVYWVRuci9ZY0pOZVMwNlI0ZEYzM1ZmcklqQnNieGRzT1RLZU5sdTkyL00KZ3Vkcjc2SzNVN3FoeEVGcjk0b3pXS2JrdXFka
GJSTzEKLS0tLS1FTkQgQ0VSVElGSUNBVEUgUkVRVUVTVC0tLS0tCg==
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
```

Step 6 : kubectl apply -f john-user.yaml (Read the documents to create CSR)
Step 7 : kubectl get csr
Step 8 : kubectl certificate approve
Step 9 : kubectl create role developer --verb=create,list,get,updates,delete --resource=pod -n development
Step 10 : kubectl get role -n development
Step 11 : kubectl describe role development -n development
Step 12 : kubectl auth can-i create pods -n development --as john
Step 13 : kubectl create rolebinding john-developer --role=developer --user=john -n development
Step 14 : kubectl get rolebinding -n development
Step 15 : kubectl describe rolebinding -n development
Step 16 : kubectl auth can-i get pods -n development --as john
Step 17 : kubectl auth can-i create pods -n development --as john
Step 18 : kubectl auth can-i updates pods -n development --as john
Step 19 : kubectl auth can-i watch pods -n development --as john

Q-7. Create a nginx pod called **nginx-resolver** using image **nginx**, expose it internally with a service called **nginx-resolver-service**.

Test that you are able to look up the **service** and **pod** names from within the **cluster**. Use the image: **busybox:1.28** for dns lookup. Record results in **/root/CKA/nginx.svc** and **/root/CKA/nginx.pod**.

- Pod: **nginx-resolver** created
- Service **DNS Resolution** recorded correctly
- Pod **DNS resolutio**n recorded correctly

Solutions

Step 1 : kubectl run nginx-resolver --image=nginx

Step 2 : kubectl get pod

Step 3 : kubectl expose pod nginx-resolver --name=nginx-resolver-service --port=80

Step 4 : kubectl run busybox --image=busybox:1.28 --command sleep 4000

Step 5 : kubectl exec busybox -- nslookup nginx-resolver-service

Step 6 : kubectl exec busybox -- nslookup nginx-resolver-service > /root/CKA/nginx.svc

Step 7 : kubectl get pod -o wide

Step 8 : kubectl exec busybox ip.namespace.pod.cluster.local -- nslookup nginx-resolver-service

Step 9 : kubectl exec busybox ip.namespace.pod.cluster.local -- nslookup nginx-resolver-service > /root/CKA/nginx.pod

Q-8. Create a static pod on **node01** called **nginx-critical** with image **nginx** and make sure that it is **recreated/restarted** automatically in case of a failure.

- Use **/etc/kubernetes/manifests** as the **Static Pod** path for example.
- static pod configured under **/etc/kubernetes/manifests** ?
- Pod **nginx-critical-node01** is up and running

Solutions

Step 1 : ssh node01

Step 2 : kubectl run nginx-critical --image=nginx --restart=Always --dry-run=client -o yaml

Step 3 : Copy file and create YAML on node01

Step 4 : cat > /etc/kubernetes/manifests/nginx-critical.yaml

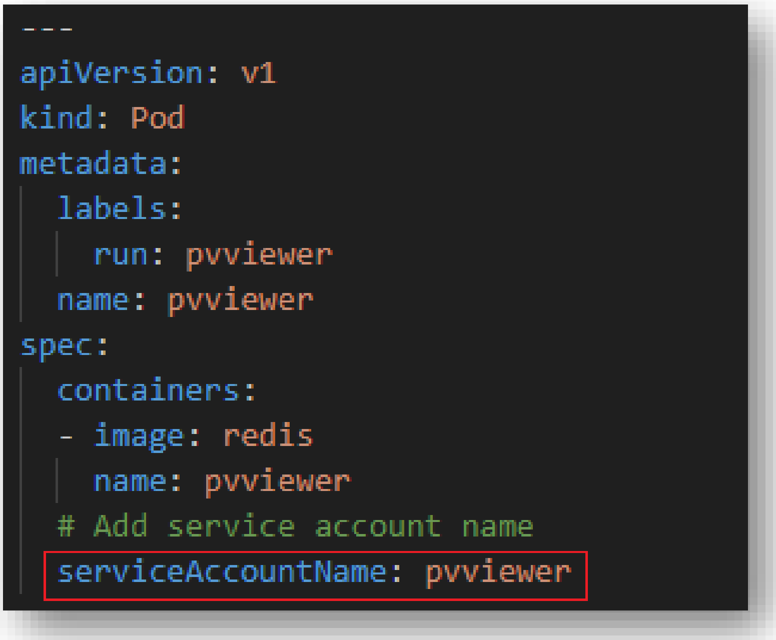
Q-1. Create a new service account with the name **pvviewer**, Grant this Service account access to **list** all **PersistentVolumes** in the cluster by creating an appropriate cluster role called **pvviewer-role** and ClusterRoleBinding called **pvviewer-role-binding**.

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

- ServiceAccount: **pvviewer**
- ClusterRole: **pvviewer-role**
- ClusterRoleBinding: **pvviewer-role-binding**
- Pod: **pvviewer**
- Pod configured to use **serviceaccount pvviewer**

Solutions

Step 1 : kubectl create service account pvviewer
Step 2 : kubectl create clusterrole pvviewer-role --verb=list --resource=persistentvolume
Step 3 : kubectl create get clusterrole pvviewer-role
Step 4 : kubectl create clusterrolebinding pvviewer-role-binding --clusterrole=pvviewer-role --serviceaccount=default:pvviewer
Step 5 : kubectl describe cluserrolebinding pvviewer-role-binding
Step 6 : kubectl run pvviewer --image=redis --dry-run=client -o yaml > pvviewer.yaml



```yaml
---
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pvviewer
  name: pvviewer
spec:
  containers:
  - image: redis
    name: pvviewer
  # Add service account name
  serviceAccountName: pvviewer
```

Q-2. List the InternalIP of all nodes of the cluster. Save the result to a file /root/CKA/node_ips.

Answer should be in the format: InternalIP of controlplane<space>InternalIP of node01 (in a single line)

Solutions

Step 1 : kubectl get node -o wide

Step 2 : kubectl get node -o jsonpath='{.items[0].status.addresses[?(@.type=="InternalIP")].address}' > /root/CKA/node.ips

Q-3. Create a pod called **multi-pod** with two containers.

Container 1: name: **alpha**, image: **nginx**

Container 2: name: **beta**, image: **busybox**, command: **sleep 4800**

Environment Variables:

Container 1:

name: **alpha**

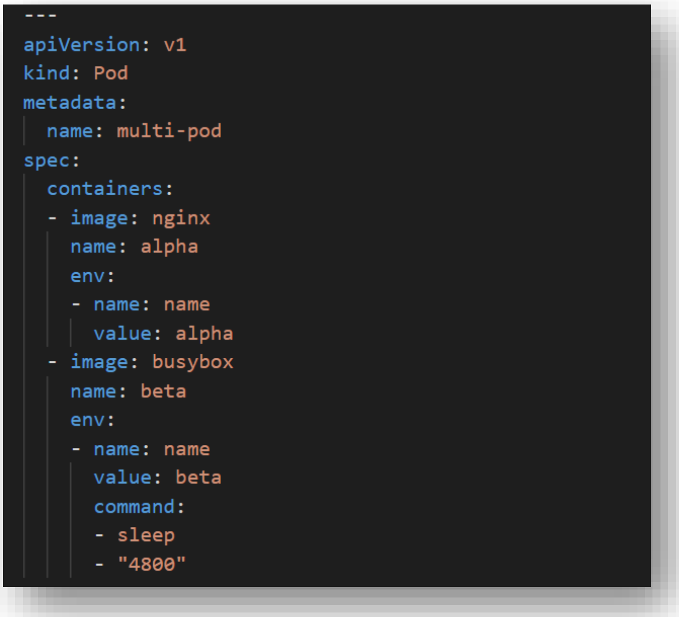
Container 2:

name: **beta**

- Pod Name: **multi-pod**
- Container 1: **alpha**
- Container 2: **beta**
- Container beta commands set correctly?
- **Container 1 Environment Value Set**
- **Container 2 Environment Value Set**

Solutions

Step 1 : kubectl run multi-pod --image=nginx --dry-run=client -o yaml > multi-pod.yaml



```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: multi-pod
spec:
  containers:
  - image: nginx
    name: alpha
    env:
    - name: name
      value: alpha
  - image: busybox
    name: beta
    env:
    - name: name
      value: beta
      command:
      - sleep
      - "4800"
```

Step 2 : kubectl apply -f multi-pod.yaml

Step 3 : kubectl describe pod multi-pod

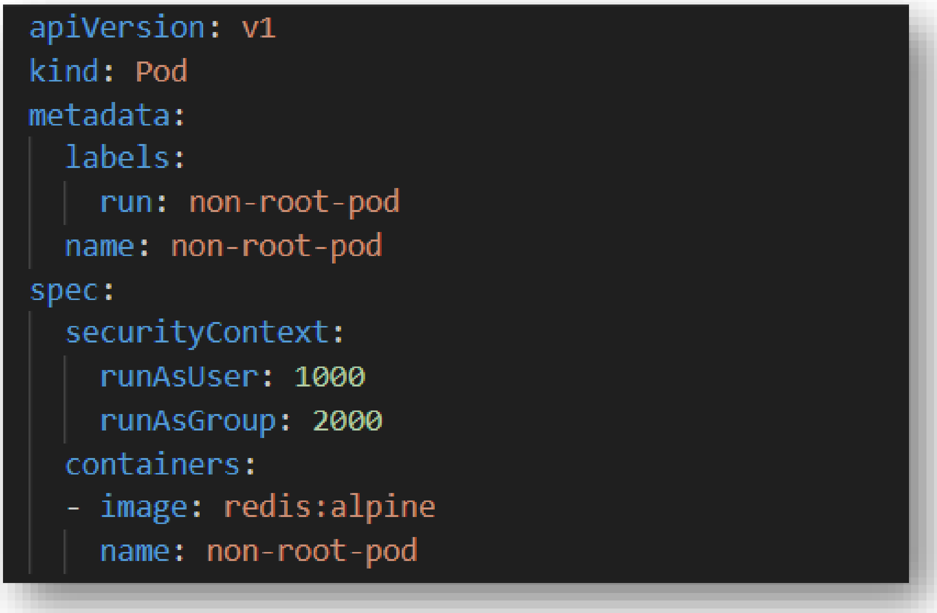
Q-4. Create a Pod called **non-root-pod** , image: **redis:alpine**

runAsUser: 1000
fsGroup: 2000

- Pod **non-root-pod fsGroup** configured
- Pod **non-root-pod runAsUser** configured

Solutions

Step 1 : kubectl run non-root-pod --image=redis:alpine --dry-run=client -o yaml > non-root-pod.yaml (Read Documents).



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: non-root-pod
  name: non-root-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 2000
  containers:
  - image: redis:alpine
    name: non-root-pod
```

Step 2 : kubectl get pod
Step 3 : kubectl describe pod non-root-pod
Step 4 : kubectl exec -it non-root-pod -- sh
Step 5 : whoami
Step 6 : id -G

Q-5. We have deployed a new pod called **np-test-1** and a service called **np-test-service**. Incoming connections to this service are not working. Troubleshoot and fix it.

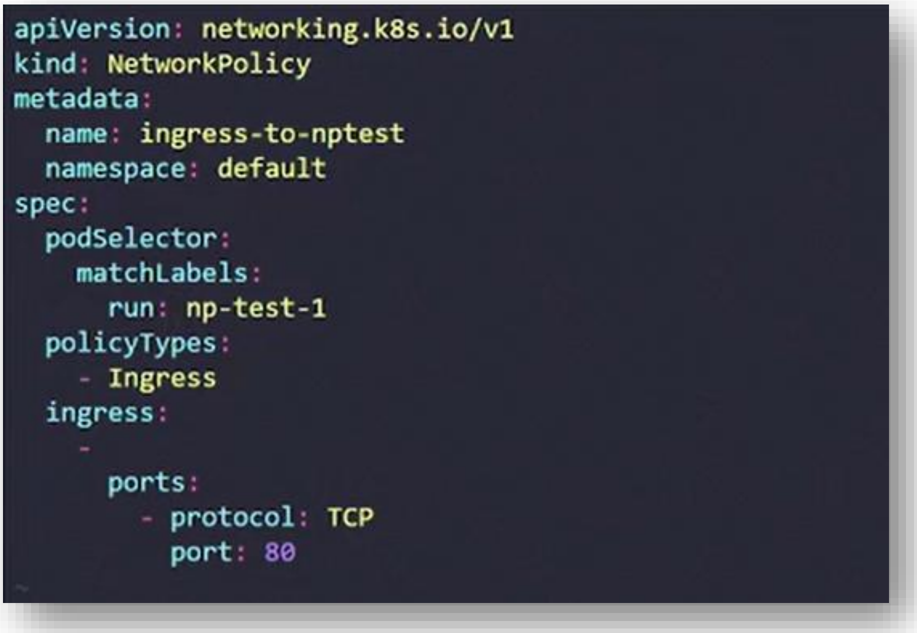
Create NetworkPolicy, by the name **ingress-to-nptest** that allows incoming connections to the service over **port 80**.

Important: Don't delete any current objects deployed.

- Important: **Don't Alter Existing Objects**!
- NetworkPolicy: **Applied to All sources** (Incoming traffic from all pods)?
- NetWorkPolicy: **Correct Port**?
- NetWorkPolicy: **Applied to correct Pod**?

Solutions

Step 1 : kubectl get pod



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ingress-to-nptest
  namespace: default
spec:
  podSelector:
    matchLabels:
      run: np-test-1
  policyTypes:
    - Ingress
  ingress:
    -
      ports:
        - protocol: TCP
          port: 80
```

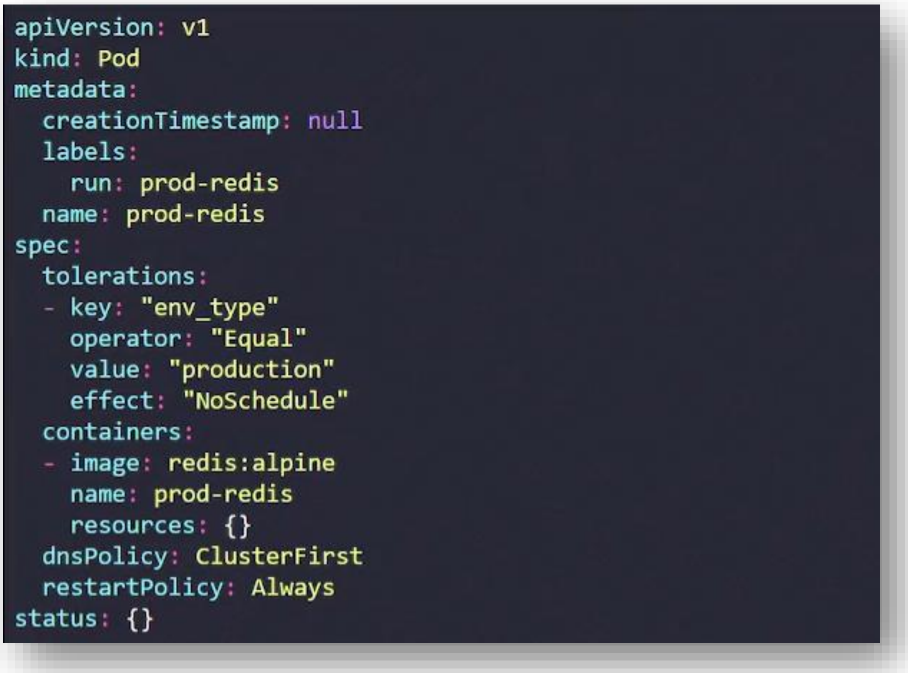
Q-6. Taint the worker node **node01** to be Unschedulable. Once done, create a pod called **dev-redis**, image **redis:alpine**, to ensure workloads are not scheduled to this worker node. Finally, create a new pod called **prod-redis** and image: **redis:alpine** with toleration to be scheduled on **node01**.

Key: **env_type**, value: **production**, operator: **Equal** and effect: **NoSchedule.**

- Key = **env_type**
- Value = **production**
- Effect = **NoSchedule**
- pod '**dev-redis**' (no tolerations) is not scheduled on **node01**?
- Create a pod '**prod-redis**' to run on **node01**

Solutions

Step 1 : kubectl get nodes
Step 2 : kubectl taint node node01 env_type=production:NoSchedule
Step 3 : kubectl describe node node01
Step 4 : kubectl run dev-redis --image=redis:alpine
Step 5 : kubectl get pods
Step 6 : kubectl run pod prod-redis --image=redis:alpine --dry-run=client -o yaml > pod-redis.yaml



```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: prod-redis
  name: prod-redis
spec:
  tolerations:
  - key: "env_type"
    operator: "Equal"
    value: "production"
    effect: "NoSchedule"
  containers:
  - image: redis:alpine
    name: prod-redis
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Step 7 : kubectl apply -f pod-redis.yaml
Step 8 : kubectl get pods
Step 9 : kubectl describe pods pod-redis
Step 10 : kubectl get pods -o wide

Q-7. Create a pod called **hr-pod** in hr namespace belonging to the **production environment** and **frontend tier**. image: **redis:alpine**

Use appropriate labels and create all the required objects if it does not exist in the system already.

- **hr-pod** labeled with **environment production**?
- **hr-pod** labeled with **tier frontend**?

Solutions

Step 1 : kubectl run hr-pod --image=redis:alpine --labels=environment=productions,tier=frontend
Step 2 : kubectl get pods --show-labels

Q-8. A kubeconfig file called **super.kubeconfig** has been created under **/root/CKA.** There is something wrong with the configuration. Troubleshoot and fix it

- Fix **/root/CKA/super.kubeconfig**

Solutions

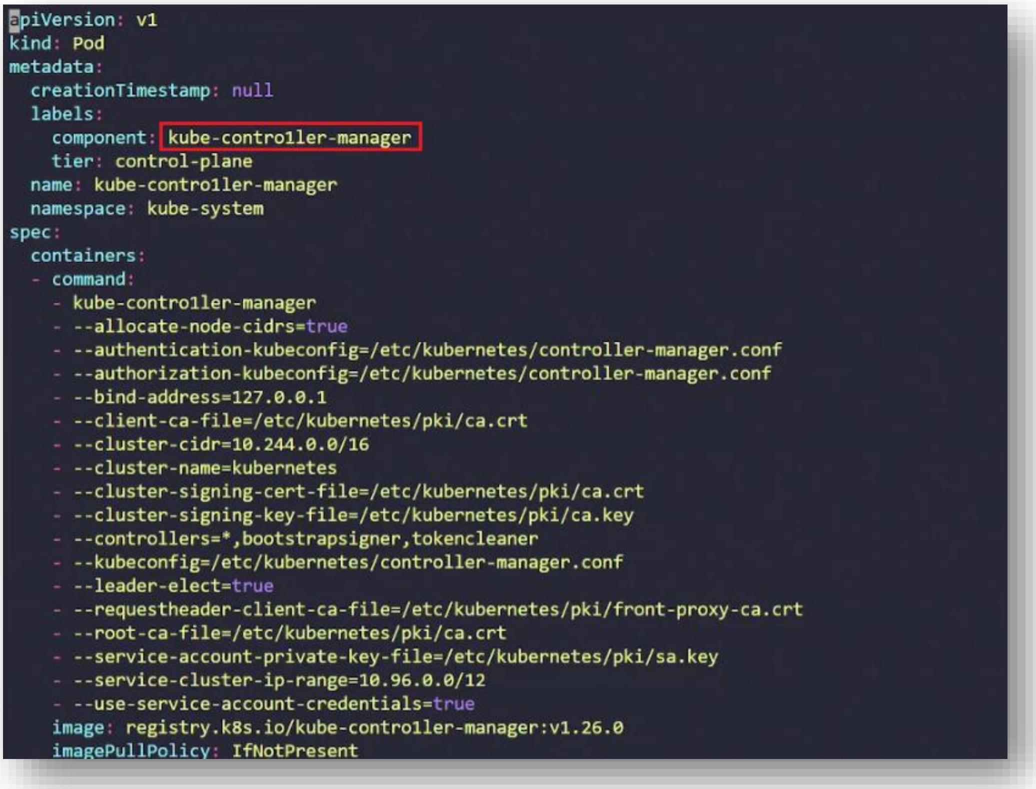
Step 1 : kubectl get nodes
Step 2 : Cat .kube/config (To Check port)
Step 3 : Change the exact port 6443

Q-9. We have created a new deployment called **nginx-deploy**. scale the deployment to **3 replicas**. Has the replica's increased? Troubleshoot the issue and fix it.

- deployment has **3 replicas**

Solutions

Step 1 : kubectl get deploy
Step 2 : kubectl scale deployment nginx-deploy --replicas=3
Step 3 : kubectl describe deploy nginx-deploy
Step 4 : kubectl get pods -n kube-system
Step 5 : vim /etc/kubernetes/manifests/kube-controller-manager.yaml



```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-contro1ler-manager
    tier: control-plane
  name: kube-contro1ler-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-contro1ler-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-cidr=10.244.0.0/16
    - --cluster-name=kubernetes
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --root-ca-file=/etc/kubernetes/pki/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.96.0.0/12
    - --use-service-account-credentials=true
    image: registry.k8s.io/kube-contro1ler-manager:v1.26.0
    imagePullPolicy: IfNotPresent
```

Q-1. Create new cluster role Named **deployment-clusterrole** which only allows to create the following resource type.

- Deployment
- StatefulSet
- DaemonSet

Create new service account named **cicd-token** in the existing namespace **app-team-1**
Bind the new ClusterRole **deployment-clusterrole** to the new ServiceAccount **cicd-token,** limited to the **namespace app-team1.**

Solutions

```
Step 1 :  kubectl config use-context k8s
Step 2 :  kubectl create clusterrole deployment-clusterrole --verb=create --
          resource=Deployment,StatefulSet,DaemonSet
Step 3 :  kubectl create sa cicd-token -n app-team1
Step 4 :  kubectl create clusterrolebinding deployment-clusterrole --clusterrole=deployment-
          clusterrole --serviceaccount=app-team1:cicd-token
Step 5 :  kubectl auth can-i create pod -n app-team1 --as system:serviceaccount:app-
          team1:cicd-token
```

Q-2. Set the node named **ek8s-node-0** as unavailable and reschedule all the pods running on it.

Solutions

```
Step 1 :  kubectl config use-context ek8s@kubernetes
Step 2 :  kubectl get nodes
Step 3 :  kubectl drain --ignore-daemonsets ek8s-node-0
Step 4 :  kubectl get nodes
```

Q-3. Given an existing kubernetes cluster running **version 1.18.8**. upgrade all of the kubernetes control plane and node components on the master node only to **version 1.19.0.**

You are also expected to upgrade **kubelet**, and **kubectl** on the **master node**.

Be sure to **drain** the **master node** before upgrading it and **uncordon** it after the upgrade. Do not upgrade the worker **noded**, **etcd**, the **container manager**, **the CNI plugin**, the **DNS service** or any **other addons**.

Solutions

```
Step 1 :  kubectl config use-context mk8s
Step 2 :  kubectl drain mk8s-master-o
Step 3 :  ssh mk8s-master-0
Step 4 :  sudo apt-get install kubeadm=1.19.0-00 kubelet=1.19.0-00 kubectl=1.19.0-00
Step 5 :  sudo systemctl daemon-reload
Step 6 :  sudo systemctl restart kubelet
Step 7 :  exit
```

Q-4. First create a snapshot of the existing **etcd** instance running at https://127.0.0.1:2379, saving the snapshot to **/srv/data/etcd-snapshot.db**

Creating a snapshot of the given instance is expected to complete in second. If the operations seems to hang, somethings likely, wrong with your command. Use **CTRL + C** to cancel the operation and try again

Next, restore an existing previous snapshot locat at /**data/backup/etcd-snapshot-previous.db**.

The following **TLS certificates/key** are supplied for connecting to the server with **etcdctl**:

- CA Certificate: **/opt/KUIN00601/ca.crt**
- Client Certificate: **/opt/KUIN00601/etcd-client.crt**
- Client Key: **/opt/KUIN00601/etcd-client.key**

Solutions

Step 1 : kubectl config use-context xn8s

Step 2 : ssh xn8s-node-0

Step 3 : cat /etc/kubernetes/manifests/etcd.yaml | grep file

Step 4 : ETCDCTL_API=3 etcdctl -h

Step 5 : ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/opt/KUIN0061/ca.crt --cert=/opt/KUIN0061/etcd-client.crt --key=/opt/KUIN0061/etcd-client.key snapshot save /srv/data/etcd-snapshot.db

Step 6 : ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/opt/KUIN0061/ca.crt --cert=/opt/KUIN0061/etcd-client.crt --key=/opt/KUIN0061/etcd-client.key --write-out=table snapshot status /srv/data/etcd-snapshot.db

Step 7 : ETCDCTL_API=3 etcdctl --endpoints=127.0.0.1:2379 snapshot restore --cacert/opt/KUIN00601/ca.crt --cert=/opt/KUIN00601/etcd-client.crt --key=/opt/KUIN00601/etcd-client.key --data-dir=/var/lib/etcd-backup /opt/etcd-backup.db

Step 8 : ETCDCTL_API=3 etcdctl snapshot restore --data-dir=/var/lib/etcd-backup2 /opt/etcd-backup.db

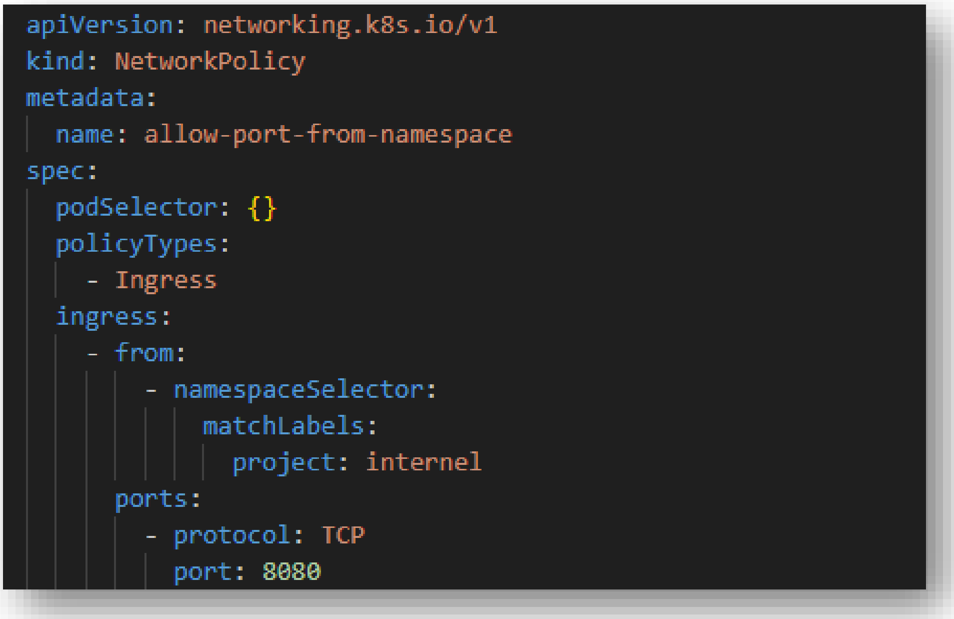
Q-5. Create new **NetworkPolicy** named **allow-port-from-namespace** that allows **Pods** in the existing namespace **my-app** to connect to port **8080** of other **Pods** in the same **namespace**.

Ensure that the new **NetworkPolicy** :

- Does not allow access to **Pods not listening on port 8080**
- Does not allow access from **Pods not in namespace my-app**

Solutions

Step 1 : kubectl config use-context nk8s
Step 2 : vim netpolicy.yml



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              project: internel
      ports:
        - protocol: TCP
          port: 8080
```

Step 3 : kubectl label ns internal project=internal
Step 4 : kubectl create -f netpolicy.yml -n my-app
Step 5 : kubectl get netpol -n my-app

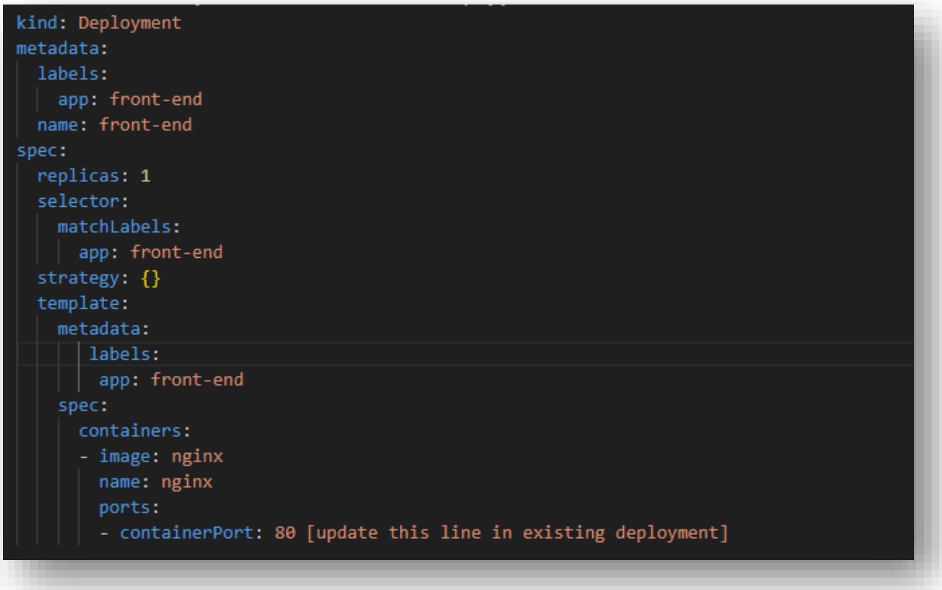
Q-6. Reconfigure the existing deployment **front-end** and add a port specification named **http** exposing **port 80/TCP** of the existing container **nginx.**

Create a **new service** named **front-end-svc** exposing the container port **http.**

Configure the new service to also **expose** the individual Pods via a **NodePort** on the nodes on which they are schedule.

Solutions

Step 1 : kubectl config use-context k8s
Step 2 : kubectl get deploy
Step 3 : kubectl get pod
Step 4 : kubectl edit deploy frontend



```yaml
kind: Deployment
metadata:
  labels:
    app: front-end
  name: front-end
spec:
  replicas: 1
  selector:
    matchLabels:
      app: front-end
  strategy: {}
  template:
    metadata:
       labels:
        app: front-end
    spec:
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80 [update this line in existing deployment]
```

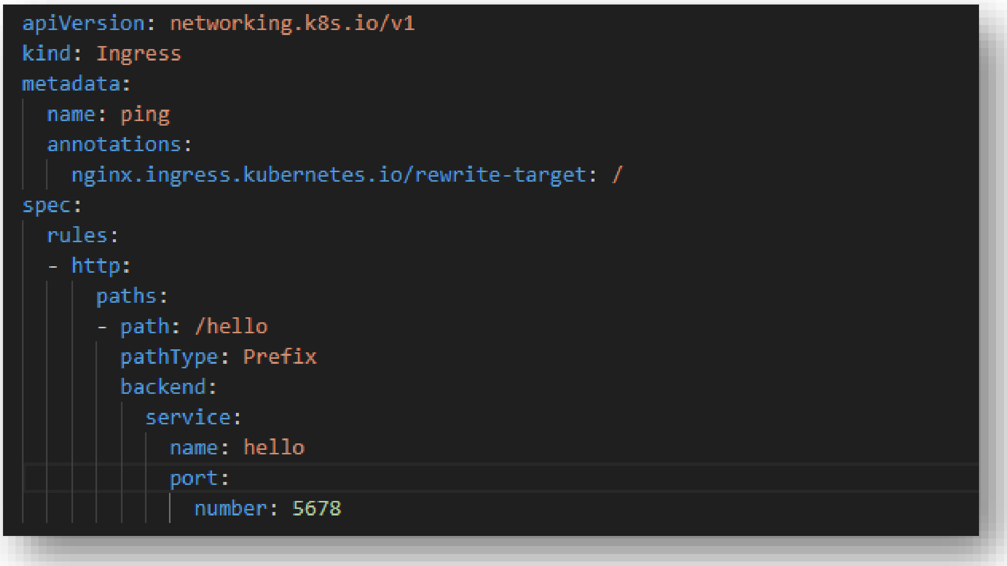
Step 5 : kubectl expose deploy front-end --type=NodePort --port=80 --name=front-end-svc
Step 6 : kubectl get service

Q-7. Create new nginx Ingress resource as follows:

- Name: **ping**
- Namespace: **ing-internal**
- Exposing service **hello** on path **/hello** using service port **5678**
- The available service **hello** can be checked using the following command, which should return **hello**. Curl **internal IP**

Solutions

Step 1 : kubectl config use-context net8s
Step 2 : vim ingress.yml



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678
```

Step 3 : kubectl create -f ingress.yml -n ing-internal

Q-8. Set Configuration context: **xt-k8s.**

- Scale the deployment webserver to **6 pods**.

Solutions

Step 1 : kubectl config use-context k8s
Step 2 : kubectl get deploy
Step 3 : kubectl scale deploy webserver --replicas=6
Step 4 : kubectl get pod

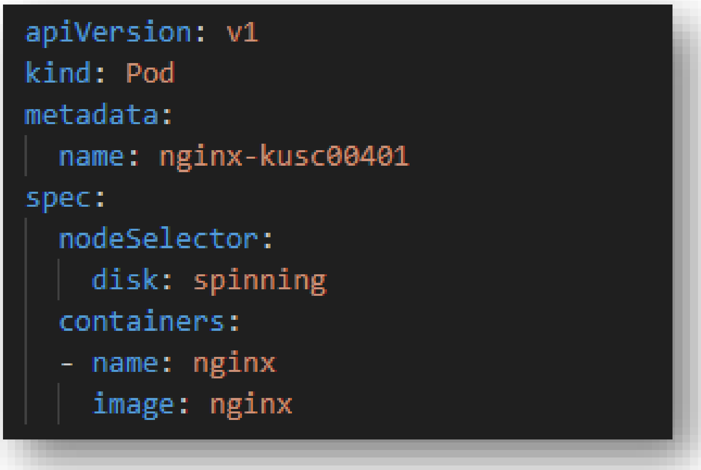
Q-9. Set Configuration context: **xt-k8s.**

Schedule a pod as follows:

- Name: **nginx-kusc00401**
- Image: **nginx**
- Node Selector: **disk=spinning**

Solutions

Step 1 : kubectl config use-context k8s
Step 2 : vim pod.yml



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
spec:
  nodeSelector:
    disk: spinning
  containers:
  - name: nginx
    image: nginx
```

Step 3 : kubectl create -f nodeselect-pod.yml
Step 4 : kubectl get pods -o wide (Pod should be created on specific node as per labels)

Q-10. Check see how many pods are ready (Not Including nodes tained NoSchedule) and write the number to **/opt/KUSC00401.txt**

Solutions

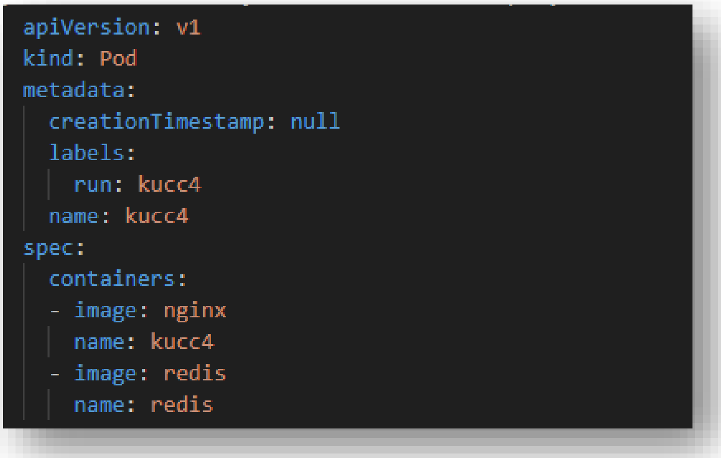
Step 1 : kubectl config use-context k8s
Step 2 : kubectl describe node | grep -i Taint or kubectl describe node | grep -A 5 Taint
Step 3 : echo 1 > /opt/KUSC00402/kusc00402.txt (Save file on the path)

Q-11. Set Configuration context: **xt-k8s.**

- Create a pod name **kucc4** with a single app container for each of the following images running inside (there may be between **1** and **4** images specified): **nginx + redis**.

Solutions

Step 1 : kubectl run kucc4 --image=nginx --dry-run -o yaml > multi-container-pod.yml
Step 2 : vim multi-container-pod.yml



```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: kucc4
  name: kucc4
spec:
  containers:
  - image: nginx
    name: kucc4
  - image: redis
    name: redis
```

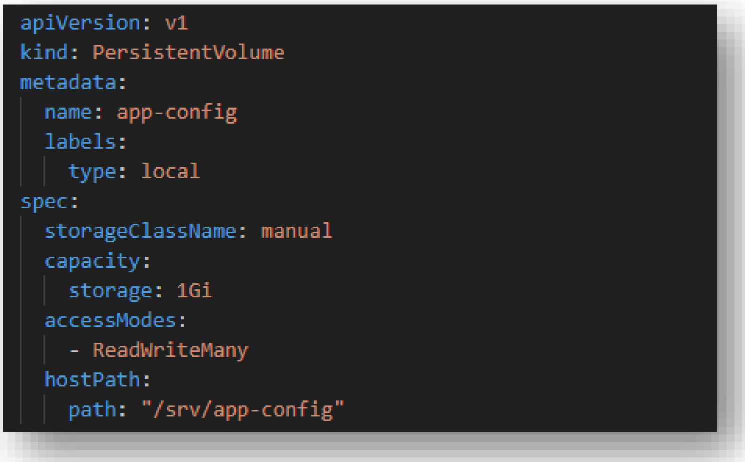
Step 3 : kubectl create -f multi-container-pod.yml

Q-12. Set Configuration context: **xt-hk8s.**

- Create a persistant volume with name **app-config** of capacity **1Gi** and access mode **ReadWriteMany**. The type of **volume path** is **host path** is **/srv/app-config**.

Solutions

Step 1 : kubectl config use-context hk8s
Step 2 : vim pv.yml



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-config
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/srv/app-config"
```

Step 3 : kubectl create -f pv.yml

Q-13. Create new **PersistantVolumeClaim**:

- Name: **pv-volume**
- Class: **csi-hostpath-sc**
- Capacity: **10 Mi**

Create a new **Pod** which mount the **PersistantVolumeClaim** as volume.

- Name: **web-server**
- Image: **nginx**
- Mount Path: **/user/share/nginx/html**

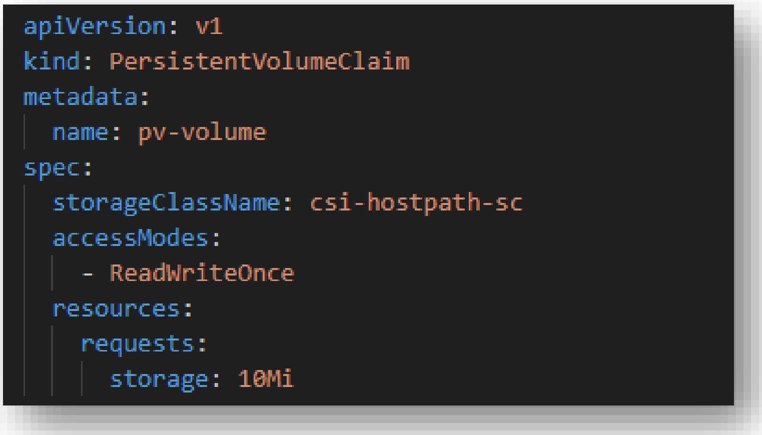
Configure the new Pod to have **ReadWriteOnce** access on the volume.

Finally, using **kubectl** edit or **kubectl** patch expand the **PersistentVolumeClaim** to a capacity of **70Mi** and record that change.

Solutions

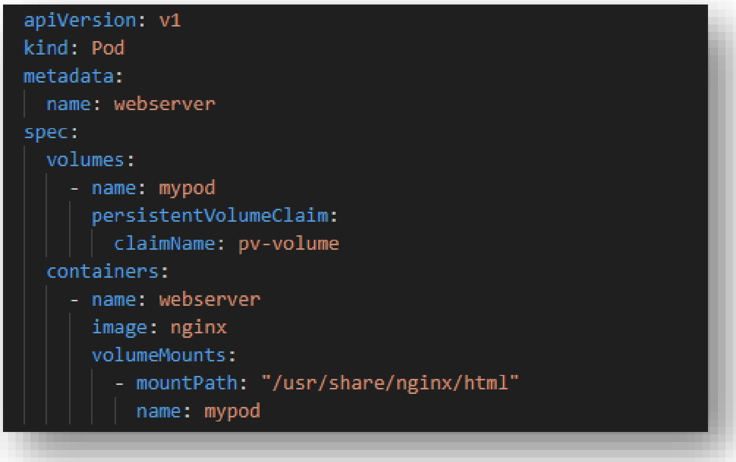
Step 1 : kubectl config use-context hk8s

Step 2 : vim pvc.yml



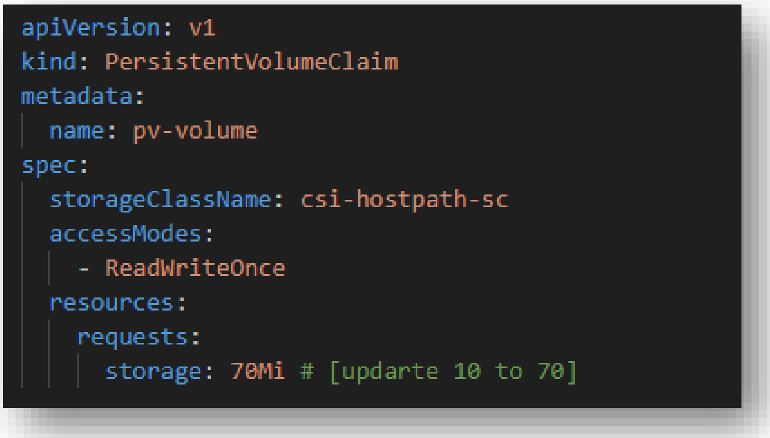
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
spec:
  storageClassName: csi-hostpath-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
```

Step 3 : vim pod-pvc.yml



```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  volumes:
    - name: mypod
      persistentVolumeClaim:
        claimName: pv-volume
  containers:
    - name: webserver
      image: nginx
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: mypod
```

Step 4 : kubectl edit pvc pv-volume



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
spec:
  storageClassName: csi-hostpath-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 70Mi # [updarte 10 to 70]
```

Step 5 : kubectl create -f pvc.yml
Step 6 : kubectl create -f pod-pvc.yml
Step 7 : kubectl get pv
Step 8 : kubectl get pod

Q-14. Set Configuration context: **xt k8s.**

Monitor the logs of Pod **bar** and:

- Extract log lines corresponding to error **Unable-to-access-website**

- Write them to **/opt/KUTR00101/bar**

Solutions

Step 1 : kubectl config use-context k8s
Step 2 : kubectl logs bar | grep unable-to-access-website > /opt/KUSSIN/bar

Q-15. Without changing its existing containers, an existing Pod needs to be integrated into kubernetes built-in login architecture **(e.g kubelet logs)**. Adding a streaming **sidecar** container is a good and common way to accomplish this requirement.

- Add a **busybox sidecar** container to the existing Pod **legacy-app**. The new sidecar container has to run the following command:

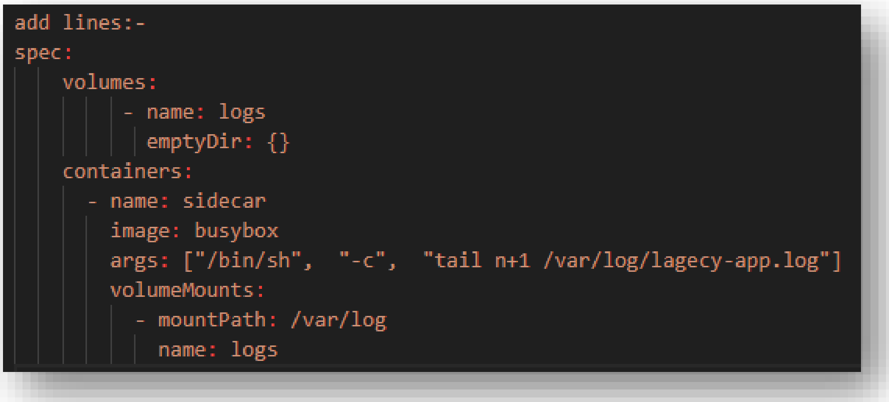
  /bin/sh -c tail -n+1 /var/log/legacy-app.log

- Use a volume mount named logs to make the file **/var/log/legacy-app.log** available to the sidecar container.

- Don’t modify the existing container, don’t modify the path of the log file, both container must access it at **/var/log/legacy-app.log**.

Solutions

Step 1 : kubectl config use-context wk8s
Step 2 : kubectl get pod
Step 3 : kubectl get pod big-corp-app -o yaml > sidecar.yml
Step 4 : kubectl delete pod big-corp-app
Step 5 : vim sidecar.yml



```
add lines:-
spec:
    volumes:
         - name: logs
           emptyDir: {}
    containers:
      - name: sidecar
        image: busybox
        args: ["/bin/sh",  "-c",  "tail n+1 /var/log/lagecy-app.log"]
        volumeMounts:
          - mountPath: /var/log
            name: logs
```

Step 6 : kubectl create -f sidecar.yml

Q-16. Set Configuration context: **xt k8s.**

- From the pod label **name=cpu-loader**, find pods running high CPU workloads and write the name of the pod consuming most CPU to the file **/opt/KUTR00401.txt** (Which already exists).

Solutions

Step 1 : kubectl config use-context k8s
Step 2 : kubectl top pod -labels name=cpu-loader
Step 3 : echo "fdkfgjk-ofkg-node" > /opt/KUTR4001/KUTR401.xt

Q-17. A kubernetes worker node, named **wk8s-node-0** is in state **NotReady.** Investigate why this is the case, and perform any appropriate steps to bring the node to a Ready state, ensuring that any changes are made permanent.

- You can SSH to the failed node using ssh wk8s-node-0

Solutions

Step 1 : kubectl config use-context wk8s
Step 2 : ssh wk8s-node-o
Step 3 : systemctl restart kubelet
Step 4 : systemctl enable kubelet

Q-1. Join a node to the cluster

Solutions

Step 1 : kubeadm token create --print-join-command

Q-2. Create **ReplicaSet** Name **appychip**, image:**nginx:1.18**, Replicaset:**4**
There is already a **pod** running in a **cluster**.
Make sure that the total count of pods running in the cli is not more than 4

Solutions

Step 1 : kubectl get replicasets
Step 2 : kubectl edit replicasets

Q-3. Create **DaemonSet**

Q-4. List all **persistent volumes** sorted by **capacity**, saving the full **kubectl output** to **/opt/pv/pv_list.txt**

Solutions

Step 1 : kubectl get pv --sort-by=.spec.capacity.storage -o wide > /opt/pv/pv_list.txt

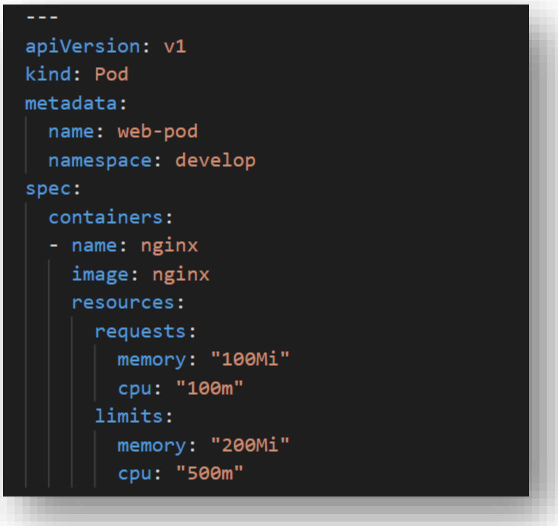
Q-5. Retrieve the logs from the pod name **'webpod',** search for any occurrences of the word **'failed'** within those **logs** and then save those findings into a file located at **'/opt/errorlogs.txt'**

Solutions

Step 1 : kubectl logs webpod | grep failed
Step 2 : kubectl logs webpod | grep failed > /opt/errorlogs.txt

Q-6. Create a pod **"web-pod"** using image **"nginx"** with a **limit** of **0.5 CPU** and **200 Mi memory** and **resource request** of **0.1 CPU** and **100 Mi memory** in **"develop"** namespace.

Step 1 : Refer the documents and create the below yaml file.
Step 2 : vim pod-resource-limit.yaml



```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
  namespace: develop
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests:
        memory: "100Mi"
        cpu: "100m"
      limits:
        memory: "200Mi"
        cpu: "500m"
```

Step 3 : kubectl apply -f pod-resource-limit.yaml

Q-7. You have access to **multiple clusters** from your main terminal through **kubectl contexts**. Write all those context those names into **/opt/course/1/contexts.**

Next write a command to display the current context into **/opt/course/1/context_default_kubectl.sh**, the command should use **kubectl**.

Finally write a second command doing the same thing into **/opt/course/1/context_default_no_kubectl.sh**, but **without the use of kubectl**.

Solutions

Step 1 : kubectl config get-contexts
Step 2 : kubectl config get-contexts > /opt/contexts
Step 3 : echo “kubectl config get-contexts” > /opt/context_default_kubectl.sh
Step 4 : sh /opt/context_default_kubectl.sh
Step 5 : cat ~/.kube/config | grep -i current-context | sed ‘s/current-context: //’
Step 6 : echo "cat ~/.kube/config | grep -i current-context | sed 's/current-context: //'" > /opt/context_default_no_kubectl.sh
Step 7 : sh /opt/ context_default_no_kubectl.sh

Q-1. The metrics-server has been installed in the **cluster**. Your college would like to know the **kubectl** command to:

- Show **Nodes** resource usage
- Show **Pods** and their container resource usage

Please write the commands into **/opt/course/7/node.sh** and **/opt/course/7/pod.sh**

Q-2. SSH into the master node with ssh **cluster-1master-1.** Check how the master components **kubelet**, **kube-apiserver, kube-scheduler, kube-controller-manager and etcd** are **started/installed** on the **master node.** Also find out the name of the **DNS** application and how it's started/installed on the **master node.**

Write your finding into file **/opt/course/8/master-components.txt.** The file should be structure like:

kubelet: **[TYPE]**
kube-apiserver: **[TYPE]**
kube-scheduler: **[TYPE]**
kube-controller-manager: **[TYPE]**
etcd: **[TYPE]**
dns: **[TYPE] [NAME]**

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

Q-3.

```
kubectl get services --sort-by=.metadata.name

kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

kubectl get pv --sort-by=.spec.capacity.storage

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl top pod POD_NAME --sort-by=cpu

kubectl api-resources | grep -i condigmap

kubectl api-resources | grep -i configmap

crictl ps

ps aux | grep -i kubelet
```

Q-1. For this question, please set the context to cluster1 by running:

kubectl config use-context cluster1

Create a **pod** with name **tester-cka02-svcn** in **dev-cka02-svcn** namespace with image **registry.k8s.io/e2e-test-images/jessie-dnsutils:1.3**. Make sure to use command sleep 3600 with restart policy set to Always .

Once the **tester-cka02-svcn** pod is running, store the output of the command nslookup **kubernetes.default** from tester pod into the file **/root/dns_output** on **student-node**.

- **'dev-cka02-svcn'** namespace exists?
- **'tester-cka02-svcn'** pod exists in **dev-cka02-svcn** namespace?
- correct image used?
- **Restart** policy set to "**Always**"?
- Command "**sleep 3600**" specified ?
- Correct dns output stored in **'/root/dns_output** ?

Solutions

```
Step 1 : kubectl config use-context cluster1
Step 2 : kubectl create ns dev-cka02-svcn
Step 3 : kubectl run tester-cka02-svcn --image registry.k8s.io/e2e-test-images/jessie-
         dnsutils:1.3 --command sleep 3600 --restart=Always -n  dev-cka02-svcn
Step 4 : kubectl exec -it tester-cka02-svcn -n dev-cka02-svcn -- nslookup kubernetes.default
Step 5 : kubectl exec -it tester-cka02-svcn -n dev-cka02-svcn -- nslookup kubernetes.default >
         /root/dns_output
Step 6 : kubectl cat /root/dns_output
```

Q-2. For this question, please set the context to cluster3 by running:

kubectl config use-context cluster3

Run a pod called **alpine-sleeper-cka15-arch** using the alpine image in the **default namespace** that will **sleep** for **7200 seconds**.

- alpine pod created?

Solutions

```
Step 1 : kubectl run alpine-sleeper-cka15-arch --image=nginx:alpine --command sleep 7200
```

Q-3. For this question, please set the context to **cluster1** by running:

kubectl config use-context **cluster1**

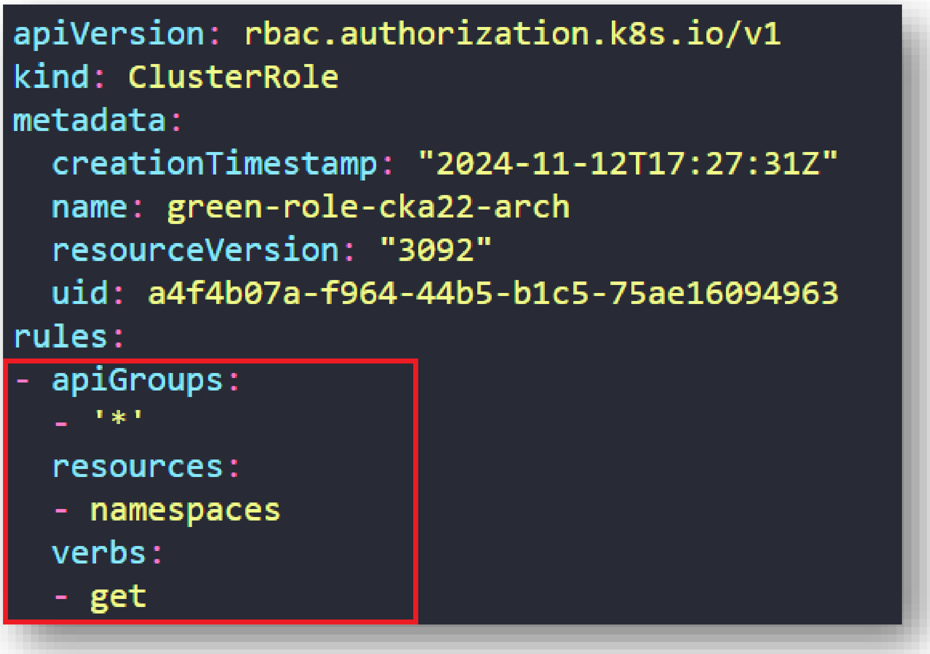
We have created a **service account** called **green-sa-cka22-arch**, a cluster role called **green-role-cka22-arch** and a cluster role binding called **green-role-binding-cka22-arch**.

Update the **permissions** of this **service account** so that it can only get all the **namespaces** in **cluster1**.

- service account permissions updated?

Solutions

Step 1 : kubectl describe clusterrole green-role-cka22-arch
Step 2 : kubectl describe clusterrolebinding green-role-binding-cka22-arch
Step 3 : kubectl edit clusterrole green-role-cka22-arch



```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: "2024-11-12T17:27:31Z"
  name: green-role-cka22-arch
  resourceVersion: "3092"
  uid: a4f4b07a-f964-44b5-b1c5-75ae16094963
rules:
- apiGroups:
  - '*'
  resources:
  - namespaces
  verbs:
  - get
```

Step 4 : kubectl describe clusterrolebinding green-role-binding-cka22-arch

Q-4. For this question, please set the context to cluster1 by running:

kubectl config use-context cluster1

There is a Cronjob called orange-cron-cka10-trb which is supposed to run every two minutes (i.e 13:02, 13:04, 13:06…14:02, 14:04…and so on). This cron targets the application running inside the orange-app-cka10-trb pod to make sure the app is accessible. The application has been exposed internally as a ClusterIP service.

However, this cron is not running as per the expected schedule and is not running as intended.

Make the appropriate changes so that the cronjob runs as per the required schedule and it passes the accessibility checks every-time.

Q-5. For this question, please set the context to cluster4 by running:

kubectl config use-context cluster4

The **pink-depl-cka14-trb** Deployment was **scaled** to **2 replicas** however, the current replicas is still **1**.

Troubleshoot and fix this issue. Make sure the **CURRENT** count is equal to the **DESIRED** count.

You can **SSH** into the **cluster4** using ssh **cluster4-controlplane** command.

Q-6. For this question, please set the context to cluster1 by running:

kubectl config use-context cluster1

A persistent volume called **papaya-pv-cka09-str** is already created with a **storage capacity** of **150Mi**. It's using the **papaya-stc-cka09-str** storage class with the path **/opt/papaya-stc-cka09-str**.

Also, a **persistent volume claim** named **papaya-pvc-cka09-str** has also been created on this cluster. This **PVC** has requested **50Mi** of storage from **papaya-pv-cka09-str** volume.

Resize the **PVC** to **80Mi** and make sure the **PVC** is in **Bound** state.

Q-7. For this question, please set the context to **cluster3** by running:

kubectl config use-context cluster3

A pod called **logger-cka03-arch** has been created in the **default** namespace. Inspect this pod and save **ALL INFO** and **ERROR's** to the file **/root/logger-cka03-arch-all** on the **student-node**.

Q-8. For this question, please set the context to cluster3 by running:

kubectl config use-context cluster3

Create a **ReplicaSet** with name **checker-cka10-svcn** in **ns-12345-svcn namespace** with image **registry.k8s.io/e2e-test-images/jessie-dnsutils:1.3**.

Make sure to specify the below specs as well:

- command **sleep 3600**
- replicas set to **2**
- container name: **dns-image**

Once the checker pods are up and running, store the output of the command **nslookup kubernetes.default** from any one of the checker pod into the file **/root/dns-output-12345-cka10-svcn** on **student-node**.

- namespace **"ns-12345-svcn"** created ?
- replicaset **"checker"** created ?
- image **"registry.k8s.io/e2e-test-images/jessie-dnsutils:1.3"** used ?
- command: **"sleep 3600"** ?
- Replicas: **2**
- Container Name: **"dns-image"** ?
- Correct output stored in **"/root/dns-output-12345-cka10-svcn"** ?

Q-9. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

Create a service account called **deploy-cka20-arch**. Further create a cluster role called **deploy-role-cka20-arch** with permissions to **get** the **deployments** in **cluster1**.

Finally create a cluster role binding called **deploy-role-binding-cka20-arch** to bind **deploy-role-cka20-arch** cluster role with **deploy-cka20-arch** service account.

- Task completed?

Q-10. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

Create a nginx pod called **nginx-resolver-cka06-svcn** using image **nginx**, expose it internally with a service called **nginx-resolver-service-cka06-svcn**.

Test that you are able to look up the service and pod names from within the cluster. Use the image: **busybox:1.28** for dns lookup. Record results in **/root/CKA/nginx.svc.cka06.svcn** and **/root/CKA/nginx.pod.cka06.svcn**

- Pod: **nginx-resolver-cka06-svcn** created
- Service **DNS Resolution** recorded correctly
- **"nginx-resolver-cka06-svcn"** pod exposed using **"nginx-resolver-service-cka06-svcn"** ?

Q-11. For this question, please set the context to **cluster4** by running:

kubectl config use-context cluster4

We tried to schedule **grey-cka21-trb** pod on **cluster4** which was supposed to be deployed by the kubernetes scheduler so far but somehow its stuck in **Pending** state. Look into the issue and fix the same, make sure the pod is in **Running** state.

You can SSH into the cluster4 using ssh cluster4-controlplane command.

- Issues fixed?
- **grey-cka21-trb** POD is in running state?

Q-12. For this question, please set the context to **cluster2** by running:

kubectl config use-context cluster2

The **yello-cka20-trb** pod is stuck in a **Pending** state. Fix this issue and get it to a **running** state. Recreate the pod if necessary.

Do not **remove** any of the existing **taints** that are set on the cluster nodes.

- Node taints unchanged?
- pod is running?

Q-13. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

In the **dev-wl07** namespace, one of the developers has performed a rolling update and upgraded the application to a newer version. But somehow, application pods are not being created.

To get back the working state, **rollback** the application to the previous version .

After rolling the deployment back, on the **controlplane** node, save the image currently in use to the **/root/rolling-back-record.tx**t file and increase the replica count to the **5**.

You can SSH into the **cluster1** using ssh **cluster1-controlplane** command.

- rolling back successful?
- image saved to the file?
- Replica set to **5**?

Q-14. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

The **db-deployment-cka05-trb** deployment is having **0** out of **1** PODs ready.

Figure out the issues and fix the same but make sure that you do not remove any DB related environment variables from the **deployment/pod**.

- **DB deployment** is fixed?

Q-15. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

Create a new deployment called **ocean-tv-wl09** in the default namespace using the image **kodekloud/webapp-color:v1**.
Use the following specs for the deployment:

1. Replica count should be **3**.
2. Set the **Max Unavailable to 40% and Max Surge to 55%.**
3. Create the **deployment** and ensure all the **pods** are ready.
4. After successful **deployment, upgrade the deployment** image to **kodekloud/webapp-color:v2** and inspect the deployment rollout status.
5. Check the rolling history of the deployment and on the **student-node**, save the **current** revision count number to the **/opt/revision-count.txt** file.
6. Finally, perform a rollback and revert back the deployment image to the older version.

- Deployment is running?
- **Replica set** to **3**?
- **maxSurge** set to **55%**?
- **maxUnavailable** set to **40%**?
- **Rolling back** successful?

Q-16. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

There is a script located at **/root/pod-cka26-arch.sh** on the **student-node**. Update this script to add a command to **filter/display** the label with value **component** of the pod called **kube-apiserver-cluster1-controlplane** (on **cluster1**) using **jsonpath**.

- script updated?

Q-17. For this question, please set the context to cluster3 by running:

kubectl config use-context cluster3

There is a deployment **nginx-deployment-cka04-svcn** in **cluster3** which is exposed using service **nginx-service-cka04-svcn**.

Create an **ingress resource nginx-ingress-cka04-svcn** to load balance the incoming traffic with the following specifications:

- pathType: Prefix and path: /
- Backend Service Name: nginx-service-cka04-svcn
- Backend Service Port: **80**
- **ssl-redirect** is set to **false**

Q-18. For this question, please set the context to **cluster1** by running:

kubectl config use-context cluster1

It appears that the **black-cka25-trb** deployment in **cluster1** isn't up to date. While listing the deployments, we are currently seeing **0** under the **UP-TO-DATE** section for this deployment. Troubleshoot, fix and make sure that this deployment is up to date.

Q-19. For this question, please set the context to cluster1 by running:

kubectl config use-context cluster1

The **purple-app-cka27-trb** pod is an nginx based app on the container port **80**. This app is exposed within the cluster using a **ClusterIP** type service called **purple-svc-cka27-trb**.

There is another pod called **purple-curl-cka27-trb** which continuously monitors the status of the app running within **purple-app-cka27-trb** pod by accessing the **purple-svc-cka27-trb** service using curl.

Recently we started seeing some errors in the logs of the **purple-curl-cka27-trb** pod.

Dig into the logs to identify the issue and make sure it is resolved.

Note: You will not be able to access this app directly from the **student-node** but you can **exec** into the **purple-app-cka27-trb** pod to check.

Q-20. For this question, please set the context to cluster1 by running:

kubectl config use-context cluster1

Create a storage class with the name **banana-sc-cka08-str** as per the properties given below:

- Provisioner should be **kubernetes.io/no-provisioner**.
- Volume binding mode should be **WaitForFirstConsumer**.
- Volume expansion should be **enabled**.