CKA Solutions

Monday, October 25, 2021 11:11 AM

Cluster Administration

Task #1

On master node, upgrade Kubernetes to latest version, including kubelet and kubectl.

Solution Steps

1. **kubeadm upgrade plan**
2. **kubectl drain cka-cluster380-master1** (we need to upgrade only master- so we need to drain it. It gives us some warnings and the final command is like this
   a. **kubectl drain cka-cluster380-master1 --ignore-daemonsets --delete-local-data**
      i. Now we delete one more pod via : **kubectl delete pod metrics-server-84645594-h89sg -n kube-system**
   b. Rerun the drain command after deleting the pod: **kubectl drain cka-cluster380-master1 --ignore-daemonsets --delete-local-data**
3. **apt-get update** (to update the repository)
4. **apt-get install kubelet kubeadm kubectl -y**
5. **systemctl restart kubelet**
6. **kubectl uncordon cka-cluster380-master1**

Screen Command for multiple windows

1. **apt-get install screen**
2. **screen**
3. Run your commands on this screen
4. Use
   a. **CTRL + A + D** to detach from screen
   b. **screen -r** to resume to the session
   c. **CTRL + D** to kill the session
   d. **screen -ls** to check the available screens

Task #2

One of the Kubernetes nodes is in NotReady state. Investigate the root cause and bring the node to a Ready state.

Note: You can ssh to the appropriate node using: ssh nodeIP

Any changes made should be made permanent

Solution Steps

1. kubectl get nodes -o wide (to check the node which is not ready and its IP address
2. **ssh root@cka-cluster380-node4** or **ssh root@172.16.3.3**
3. **systemctl status docker** (working)
4. **systemctl status kubelet** (not working)
5. **systemctl start kubelet**
6. **systemctl enable kubelet** (to make it permanent)
7. **systemctl status kubelet** (working)

Task #3

Set the node5 as unavailable and reschedule all the pods running on it

Solution Steps

1. **kubectl get nodes** to get the node's full name (cka-cluster380-node5)
2. **kubectl drain cka-cluster380-node5**
3. Update command with the additional flags suggested by terminal:
   a. **kubectl drain cka-cluster380-node5 --ignore-daemonsets**
4. **kubectl get nodes** to confirm that the node is in scheduling disabled status

```
root@cka-cluster380-master1:~/metrics-server# kubectl get nodes
NAME                    STATUS                     ROLES    AGE   VERSION
cka-cluster380-master1   Ready,SchedulingDisabled   master   43h   v1.19.1
cka-cluster380-node1     Ready                      <none>   42h   v1.19.1
cka-cluster380-node2     Ready                      <none>   42h   v1.19.1
cka-cluster380-node3     Ready                      <none>   42h   v1.19.1
cka-cluster380-node4     Ready                      <none>   42h   v1.19.1
cka-cluster380-node5     Ready,SchedulingDisabled   <none>   42h   v1.19.1
root@cka-cluster380-master1:~/metrics-server#
```

Pods, Resource Requests/Limits

Task #4

Create a pod named **monolithic** with 2 containers running off of **nginx** + **memcached** images

Solution Steps

1. **kubectl run monolithic--image=nginx --dry-run=client -o yaml >task4.yaml**
2. **vi task4.yam**l
   a. Add another image" memcached
   b. wq!
3. **kubectl apply -f task4.yaml**
4. **kubectl get pods**
5. Confirm that the monolithic pod has 2/2 containers running

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE

root@cka-cluster380-master1:~/metrics-server# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
call                          1/1     Running   0          2d8h
jenkins-6f6c59db6c-52wgc      1/1     Running   0          2d8h
jenkins-6f6c59db6c-lqf2r      1/1     Running   0          2d8h
logger                        1/1     Running   0          2d8h
marketplace-app               1/1     Running   0          2d8h
mobile-app-575f49855c-vknh6   1/1     Running   0          13h
monolithic                    2/2     Running   0          15s
probe-app                     1/1     Running   0          2d8h
root@cka-cluster380-master1:~/metrics-server#
```

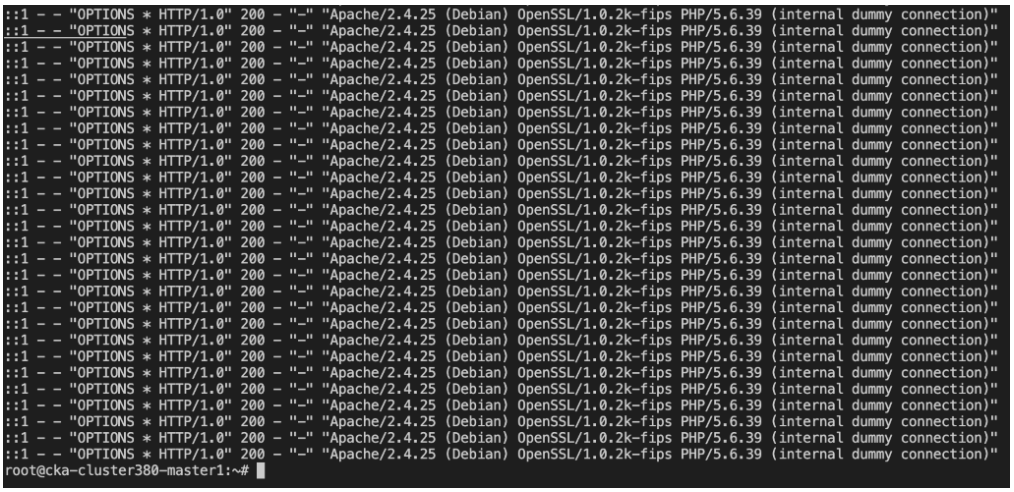
Task #5

Locate the logs of Pod **marketplace-app** and filter lines corresponding to message **dummy** and send them to **/tasks/JZNEIGGD1/marketplace-app.log**

Solution Steps

1. **kubectl logs marketplace-app |grep dummy > /tasks/JZNEIGGD1/marketplace-app.log**
2. **cat /tasks/JZNEIGGD1/marketplace-app.log**

```
root@cka-cluster380-master1:~# cat /tasks/JZNEIGGD1/marketplace-app.log
::1 - - "OPTIONS * HTTP/1.0" 200 - "-" "Apache/2.4.25 (Debian) OpenSSL/1.0.2k-fips PHP/5.6.39 (internal dummy connection)"
::1 - - "OPTIONS * HTTP/1.0" 200 - "-" "Apache/2.4.25 (Debian) OpenSSL/1.0.2k-fips PHP/5.6.39 (internal dummy connection)"
::1 - - "OPTIONS * HTTP/1.0" 200 - "-" "Apache/2.4.25 (Debian) OpenSSL/1.0.2k-fips PHP/5.6.39 (internal dummy connection)"
::1 - - "OPTIONS * HTTP/1.0" 200 - "-" "Apache/2.4.25 (Debian) OpenSSL/1.0.2k-fips PHP/5.6.39 (internal dummy connection)"
```



Task #6

Update an existing Pod named **logger**:

- Run sidecar container off of busybox image, name it logger
- Shared volume **/var/log** should be mounted on **both containers**, which does not persist when the pod is deleted
- The logger container should run following command: **tail -f /var/log/input.log**

Solution Steps

1. We will copy the existing .yaml configuration of logger
   a. **kubectl get pod logger -o yaml > logger.yaml**
2. We will add the second container named logger from busybox image
   a. **vi logger.yaml**
   b. add container (name: logger, image: busybox)
3. We will add the arguments to the logger container
   a. **kubectl run test --image=busybox --dry-run=client -o yaml --tail -f /var/log/input.log**
   b. copy and paste the lines about arguments to logger.yaml

```
spec:
  containers:
  - image: evolvecybertraining/logger-app
    imagePullPolicy: Always
    name: app
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:

  - image: busybox
    name: logger
    args:
    - tail
    - -f
    - /var/log/input.log
```

4. We will mount an emptyDir volume to both containers based on Kubernetes resource page
   a. Add the lines about **volume** to entire pod
   b. Add the lines about **volume mount** to both containers

```
containers:
- image: evolvecybertraining/logger-app
  imagePullPolicy: Always
  name: app
  resources: {}
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  volumeMounts:
  - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    name: default-token-pdclk
    readOnly: true

  volumeMounts:
  - mountPath: /var/log
    name: cache-volume


- image: busybox
  name: logger
  args:
  - tail
  - -f
  - /var/log/input.log

  volumeMounts:
  - mountPath: /var/log
    name: cache-volume
```

5. Apply the configuration file forcefully
   a. **kubectl apply --force -f logger.yaml**

6. Confirm the pod is running with two containers.
   a. **kubectl get pods**

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl get pods |grep logger
logger                          2/2     Running   0          26s
root@cka-cluster380-master1:~#
```

GREAT!!

Task #7

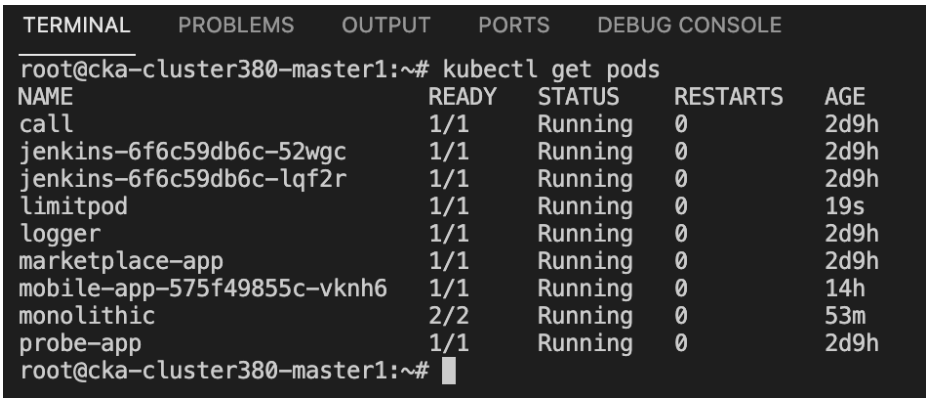
Create a Pod with 128MB memory and 100M CPU limit

Solution Steps

1. Create a random Pod with dry-run:
   a. **kubectl run limitpod --image=nginx --dry-run=client -o yaml > limitpod.yaml**
2. Edit the pod definition file to configure the pod definition for limits based on the Kubernetes documentation page (Search string: **Limit**).
   a. **vi limitpod.yaml**
3. Create the Pod:

   a. **kubectl apply -f limitpod.yaml**

4. Verify the Pod
   a. **kubectl get pods**



Liveness and Readiness Probes

Task #8
A Pod in your cluster often gets locked in a broken state, you have to remedy this situation.
The desired behavior is to restart the Pod when it returns HTTP 500 response and never route Service traffic to the Pod while it is failing.
Complete the following:

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

Solution Steps

1. Get the .yaml configuration of the existing probe-app Pod and save it in a .yaml file -
   a. **kubectl get pod probe-app -o yaml > probe.yaml**
2. Edit the .yaml file and add liveness and readiness probes to the container as indicated in the K8 resources page.

   a. **vi probe.yaml**
3. Create the resource by forcefully applying the changes.
   a. **kubectl apply --force -f probe.yaml**

```
spec:
  containers:
  - image: evolvecybertraining/probe-app:v1
    imagePullPolicy: Always
    name: probe-app
    resources: {}
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3

    readinessProbe:
      httpGet:
        path: /started
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3

    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-pdclk
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
```

```
root@cka-cluster380-master1:~# kubectl apply  --force -f task8.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
pod/probe-app configured
```

GREAT!

Labels, Taints and Tolerations

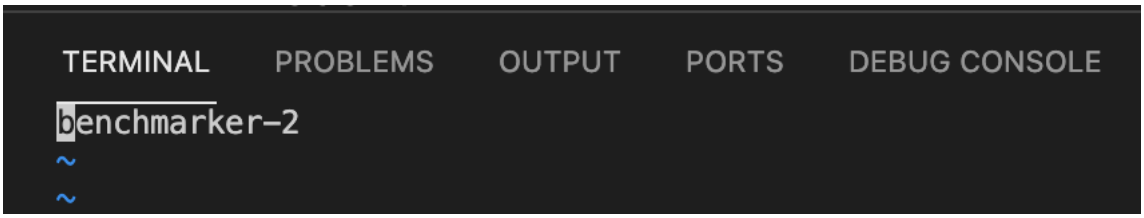
Task #9

In production namespace, out of the Pods labeled **group=tools**, find Pod which is using the most CPU resources and pipe the name to /tasks/JSCUERBQ1/pod.txt

Solution Steps

1. Rank the pods which uses the highest CPU
   a. **kubectl top pods -n production**
2. Check the labels of the pods

   a. **kubectl get pods –show-labels –n production**
3. See the pod and enter it into the text file
   a. **vi /tasks/JSCUERBQ1/pod.txt**



GREAT!

Task #10

Configure a static Pod on the Kubernetes node **node1**. Pod should be named **webtool** and run single container of **nginx** image.

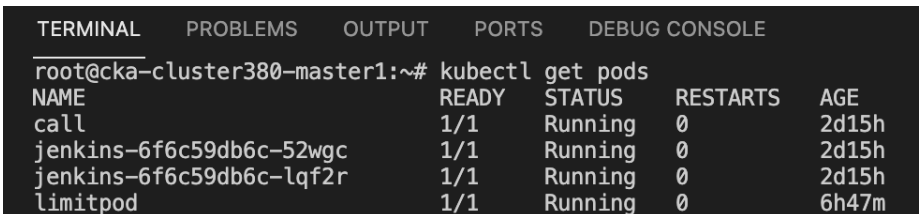
Note: You can ssh to the appropriate node using: ssh nodeIP

Solution Steps

1. On master, create a .yaml configuration using imperative command:
   a. **kubectl run webtool --image=nginx --dry-run=client -o yaml > webtool.yaml**
2. Copy this yaml file into the /etc/kubernetes/manifests directory of node01.
   a. **scp webtool.yaml cka-cluster380-node1: /etc/kubernetes/manifests/**
3. Check if the pod is running
   a. **kubectl get nodes**

If for some reason the pod is not working, edit the config.yaml file on the **node** :

1. **ssh cka-cluster380-node1**
2. **vim /var/lib/kubelet/config.yaml**
3. Add the following line into the document
4. staticPodPath:/etc/kubernetes/manifests



```
logger                          1/1     Running   0          2d15h
marketplace-app                 1/1     Running   0          2d15h
mobile-app-575f49855c-vknh6     1/1     Running   0          20h
monolithic                      2/2     Running   0          7h40m
probe-app                       1/1     Running   0          3h1m
webtool-cka-cluster380-node1    1/1     Running   0          32s
root@cka-cluster380-master1:~#
```

Task #11

Check the number of nodes in the cluster (not including nodes tainted test:NoSchedule) and write the number to /tasks/LCZIKXUY1/nodes.txt

Solution Steps

1. Get the number of pods with taint
    a. **kubectl describe nodes |grep Taints**

```
TERMINAL    PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl describe nodes |grep Taints
Taints:             node.kubernetes.io/unschedulable:NoSchedule
Taints:             <none>
Taints:             <none>
Taints:             test:NoSchedule
Taints:             <none>
Taints:             node.kubernetes.io/unschedulable:NoSchedule
root@cka-cluster380-master1:~#
```

2. Type **5** into /tasks/LCZIKXUY1/nodes.txt
    a. **vi /tasks/LCZIKXUY1/nodes.txt**

Services

Task #12

Get the list of all Pods attached to Service **fleetman** in a namespace called **tools** and put it to **/tasks/DHHSFRHV08/pod-list.txt**

Note: List should include one Pod per line

Solution Steps

1. Get the details of the fleetman service:
   a. **kubectl get service fleetman -n tools -o wide**

```
root@cka-cluster380-master1:~# kubectl get service fleetman -n tools -o wide
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE     SELECTOR
fleetman   ClusterIP   10.98.251.89    <none>        80/TCP     3d10h   app=fleetman
```

2. Get the pods with the selector app=fleetman
   a. **kubectl get pods -n tools --show-labels | grep app=fleetman**

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get pods -n tools --show-labels | grep app=fleetman
web-release-v1-1   1/1   Running   0          3d10h   app=fleetman
web-release-v1-2   1/1   Running   0          3d10h   app=fleetman
web-release-v1-3   1/1   Running   0          3d10h   app=fleetman
root@cka-cluster380-master1:~#
```

3. **Write the pod names into the** /tasks/DHHSFRHV08/pod-list.txt with awk command
   a. **kubectl get pods -n tools --show-labels | grep app=fleetman |awk '{print $1}' >/tasks/DHHSFRHV08/pod-list.txt**

```
Recent
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE
web-release-v1-1
web-release-v1-2
web-release-v1-3
~
~
~
~
~
```

GREAT!

Ingresses

Task #13

Create a new Ingress named **front-end** in a namespace called **fleet.** Route all traffic for path **/hello** to Service named **hello**

Note: Service already exists

Solution Steps

1. Create a .yaml definition file for the ingress
   a. **vi front-end-ingress.yaml**
   b. Update the .yaml file accordingly

```
TERMINAL  PROBLEMS  OUTPUT  PORTS

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: front-end
spec:
  rules:
  - host:
    http:
      paths:
      - path: /hello
        backend:
          serviceName: hello
          servicePort: 80
```

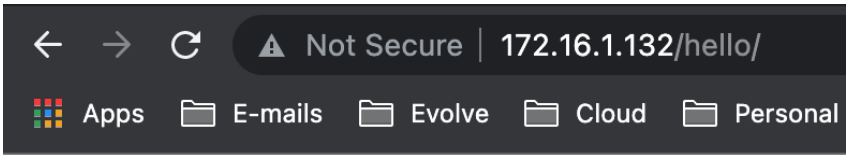
2. Create the ingress on namespace : fleet
   a. **kubectl apply -f vi front-end-ingress.yaml -n fleet**

3. Confirm the ingress is created
   a. **kubectl get ingresses –n fleet**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl get ingresses -n fleet
Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
NAME        CLASS    HOSTS   ADDRESS         PORTS   AGE
front-end   <none>   *       172.16.1.132    80      5m42s
root@cka-cluster380-master1:~#
```

4. Check if the ingress works on the browser
   a. **172.16.1.132/hello**



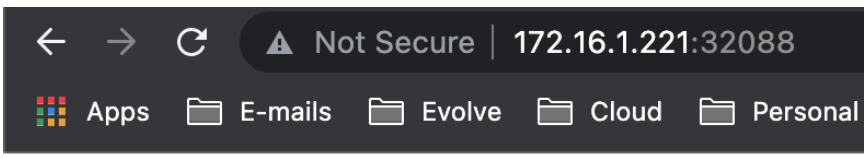
Hello

5. Check the service (NodeIP: service NopePortIP)
   a. **kubectl get svc -n fleet**

```
root@cka-cluster380-master1:~# kubectl get svc -n fleet
NAME    TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
hello   NodePort   10.105.218.180   <none>        80:32088/TCP   2d19h
root@cka-cluster380-master1:~#
```

   b. **kubectl get nodes -n fleet**

```
root@cka-cluster380-master1:~# kubectl get nodes  -n fleet -o wide
NAME                    STATUS                     ROLES    AGE     VERSION   INTERNAL-IP
cka-cluster380-master1  Ready                      master   2d19h   v1.19.1   172.16.2.69
cka-cluster380-node1    Ready                      <none>   2d19h   v1.19.1   172.16.2.123
cka-cluster380-node2    Ready                      <none>   2d19h   v1.19.1   172.16.3.253
cka-cluster380-node3    Ready                      <none>   2d19h   v1.19.1   172.16.3.95
cka-cluster380-node4    Ready                      <none>   2d19h   v1.19.1   172.16.1.221
cka-cluster380-node5    Ready,SchedulingDisabled   <none>   2d19h   v1.19.1   172.16.2.142
root@cka-cluster380-master1:~#
```



Hello

Hello

GREAT!

Volumes

Task #14

- Create a new Pod off of **jenkins** image, name it **dev-jenkins**
- Attach a PersistentVolumeClaim named **data-volume** with **1Gi** capacity and access mode of **ReadWriteOnce**
- Mount the volume to **/var/lib/jenkins/** path
- Schedule the Pod in **dev** namespace
- Use kubectl edit or kubectl patch command to resize PersistentVolumeClaim to 2Gi

Solution Steps

1. We need to create .yaml files for
   a. Persistent Volume Claim (take until **Storage** line)
   b. Pod

   using the Kubernetes Resource Page
2. **vi data-volume.yaml**

PersistentVolumeClaims

Each PVC contains a spec and status, which is the specification and status of the claim. The name of a PersistentVolumeClaim object must be a valid DNS subdomain name.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

3. **vi dev-jenkins.yaml** (we are using the template on the same page)
   We make sure we pull the image from **jenkins/jenkins** path **NOT jenkins**

Claims As Volumes

Pods access storage by using the claim as a volume. Claims must exist in the same namespace as the Pod using the claim. The cluster finds the claim in the Pod's namespace and uses it to get the PersistentVolume backing the claim. The volume is then mounted to the host and into the Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

4. We create the resources on the dev namespace using the .yaml files
   a. **kubectl apply -n dev -f data-volume.yaml**
   b. **kubectl apply -n dev -f dev-jenkins.yaml**

5. We confirm the pvc and pod have been created
   a. **kubectl get pvc -n dev**
   b. **kubectl get pods-n dev**

6. We resize the PVC
   a. **kubectl edit pvc data-volume**
   b. **Change 1 Gi to 2 Gi**
   c. **Exit**
   d. **(in this cluster, the resize failed but in the real exam we should be good with kubectl edit pvc …. command)**

Task #15

Perform the following tasks:

- Create a file on node **node1** at **/opt/FGUF00101/data/index.html** with the content **Safari**
- Create a **PersistentVolume** named **task-pv-volume** using **hostPath** and allocate **1Gi** to it, use**/opt/FGUF00101/data** as a destination directory. The configuration should specify the **access mode** of **ReadWriteOnce**. It should define the **StorageClass** name **storage**, which will be used to bind PersistentVolumeClaim requests to this PersistentVolume
- Create a PersistentVolumeClaim named **task-pv-claim** that requests a volume of **100Mi** and specifies an access mode of **ReadWriteOnce**
- Create a Pod that uses the PersistentVolumeClaim as a volume with a label **app: my-storage-app** mounting the resulting volume to a **mountPath /usr/share/nginx/html** inside the Pod

Note: You can access node1 by running ssh node1_IP

Solution Steps

1. Create a file on node **node1**
   a. **kubectl get nodes** (learn the full name of node1)
   b. **ssh cka-cluster380-node1**
   c. **mkdir -p /opt/FGUF00101/data**
   d. **vi /opt/FGUF00101/data/index.html**

2. Create a **task15.yaml file** based on the Kubernetes resource page (search string: **hostPath**)
   a. Add the following code to the file to add Persistent volume
      i. Change the size to 1 Gi
      ii. Storage class: storage
      iii. path:/opt/FGUF00101/data

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
```

```
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

b. Add the following code from the same page to the .**yaml** file after putting **---**
   i. Change the size to 100Mi
   ii. storageClassName: storage

Create a PersistentVolumeClaim

The next step is to create a PersistentVolumeClaim. Pods use PersistentVolumeClaims to request physical storage. In this exercise, you create a PersistentVolumeClaim that requests a volume of at least three gibibytes that can provide read-write access for at least one Node.

Here is the configuration file for the PersistentVolumeClaim:

pods/stora

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

c. Add the following code from the same page to the .**yaml** file after putting **---**
   i. Add **label** as a child of metadata:
      1. labels:
         a. app: my-storage-app
   ii. **Add nodeSelector as child of spec**
      1. **nodeSelector:**
         a. kubernetes.io/hostname:cka-cluster380-node1 (specific to node1 – found through kubectl get nodes –show-labels |grep abc)

Create a Pod

The next step is to create a Pod that uses your PersistentVolumeClaim as a volume.

Here is the configuration file for the Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
```

```
    ports:
      - containerPort: 80
        name: "http-server"
    volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: task-pv-storage
```

3. Create the three resources : PV, PVC , and Pod
   a. **kubectl apply -f task15.yaml**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl apply -f  task15.yaml
persistentvolume/task-pv-volume created
persistentvolumeclaim/task-pv-claim created
pod/task-pv-pod created
root@cka-cluster380-master1:~#
```

4. Confirm the services one by one:
   a. **kubectl get pv**
   b. **kubectl get pvc**
   c. **kubectl get Pods**

5. **curl http://10.40.0.15** (Private IP of the Pod- learned via kubectl get pods -o wide)

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

root@cka-cluster380-master1:~# curl http://10.40.0.15
Safari
root@cka-cluster380-master1:~#
```

GREAT!!!

ConfigMaps / Secrets

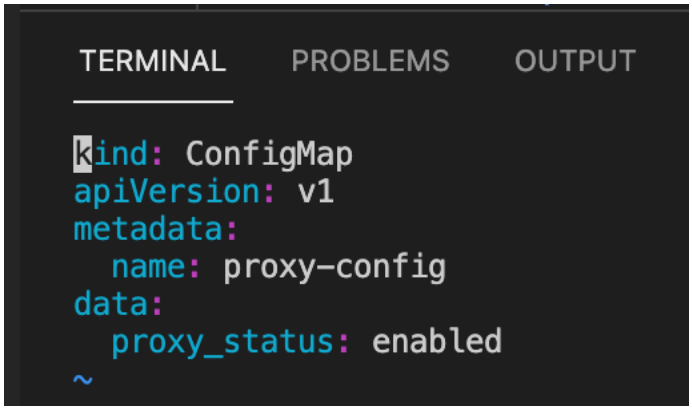
Task #16

- Create a ConfigMap named **proxy-config** containing the key/value pair: **proxy_status/enabled**
- Start a Pod named **nginx-proxy** containing a single container using the **nginx** image, and attach the ConfigMap you just created as an environment variable **PROXY_STATUS**.

Solution Steps

1. Create a .yaml file for proxy-config.
   a. **vi proxy-config.yaml**

2. Edit the .yaml file as follows:



3. Create the config map
   a. **kubectl apply -f proxy-config.yaml**

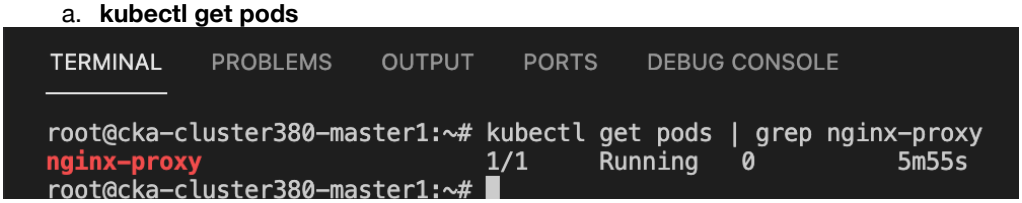
4. Or run the following command:
   a. **kubectl create configmap proxy-config --from-literal=proxy_status=enabled**

5. Create the initial .yaml configuration of the Pod
   a. **kubectl run nginx-proxy --image=nginx --dry-run=client -o yaml > nginx-proxy.yaml**

6. Edit the .yaml file to attach the configMap to the Pod as indicated here

7. Create the pod
   a. **kubectl apply -f nginx-proxy.yaml**

8. Check if it is running
   a. **kubectl get pods**



8. We further confirm by running the --env command
   a. **kubectl exec nginx-proxy -- env |grep PROXY_STATUS**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl exec nginx-proxy -- env  |grep PROXY_STATUS
PROXY_STATUS=enabled
root@cka-cluster380-master1:~#
```

GREAT!

Task #17

Create a **Secret** as follows:

- Name: **db-secret**
- Data: **password/confidential**
- Create a Pod named **mysql-pod**, using the **mysql:5.7** image, which uses the secret as an environment variable **MYSQL_ROOT_PASSWORD** and mounts the secret at **/credentials** directory.

Solution Steps

1. Create the secret
   a. **kubectl create secret generic db-secret --from-literal=password=confidential**
2. Create the Pod imperatively
   a. **kubectl run mysql-pod --image=mysql:5.7 --dry-run=client -o yaml > mysql-pod.yaml**
3. Edit the **env section** of the pod as indicated here (Search: Using Secrets as environment variables, also check volume mount)

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mysql-pod
  name: mysql-pod
spec:
  containers:
  - image: mysql:5.7
    name: mysql-pod
    env:
      - name: MYSQL_ROOT_PASSWORD
        valueFrom:
          secretKeyRef:
            name: db-secret
```

```
          key: password
    volumeMounts:
    - name: foo
      mountPath: "/credentials"
  volumes:
  - name: foo
    secret:
      secretName: db-secret
~
```

4. Create the pod
   a. **kubectl apply -f mysql-pod.yaml**
5. Confirm the pod is running
   a. **Kubectl get pod mysql-pod**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get pod mysql-pod
NAME        READY   STATUS    RESTARTS   AGE
mysql-pod   1/1     Running   0          6m9s
root@cka-cluster380-master1:~#
```

6. We further confirm by running the --env command
   a. **kubectl exec mysql-pod -- env |grep MYSQL_ROOT_PASSWORD**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl exec mysql-pod -- env  |grep MYSQL_ROOT_PASSWORD
MYSQL_ROOT_PASSWORD=confidential
root@cka-cluster380-master1:~#
```

GREAT!

Deployments

Task #18
Create a new deployment, name it **gitlab**, use **gitlab/gitlab-ce** image for the template. **Expose** the deployment with **ClusterIP** service. Service should listen on port **80**, name of the service has to be **gitlab-svc**.

Solution Steps

1. Create the deployment imperatively
   a. **kubectl create deployment gitlab --image=gitlab/gitlab-ce**

2. Check if the dployment has been created along with the replicaset and the pod
   a. **kubectl get deploy**
   b. **kubectl get rs**
   c. **kubectl get pod**

```
TERMINAL    PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get deploy
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
gitlab       1/1     1            1           6m11s
jenkins      2/2     2            2           10d
mobile-app   1/1     1            1           10d
root@cka-cluster380-master1:~# kubectl get rs
NAME                   DESIRED   CURRENT   READY   AGE
gitlab-74b878d645      1         1         1       6m13s
jenkins-6f6c59db6c     2         2         2       10d
mobile-app-575f49855c  1         1         1       10d
root@cka-cluster380-master1:~# kubectl get pods |grep gitlab
gitlab-74b878d645-vzgcm        1/1     Running   0          6m23s
root@cka-cluster380-master1:~#
```

3. Create the service by exposing the deployment
   a. **kubectl expose deployment gitlab --name=gitlab-svc --type=ClusterIP --port=80**

4. Check if the service has successfully been created
   a. **kubectl get services**

```
TERMINAL    PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
gitlab-svc   ClusterIP   10.100.42.189   <none>        80/TCP    108s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   10d
probe-http   ClusterIP   10.100.99.43    <none>        80/TCP    10d
root@cka-cluster380-master1:~#
```

GREAT!

Task #19
Complete the following:

- Update the **webapp** Deployment in the **production** Namespace with
  a maxSurge of 10% and
  a maxUnavailable of 5%

- Perform a rolling update of the **webapp** Deployment, changing the **evolvecybertraining/nginx** image version
  to **1.11.3**

- Roll back the webapp Deployment to the previous version

Solution Steps

1. Edit the **webapp** deployment on production workspace (maxSurge=10% and maxUnavailable=5%)
   a. **kubectl edit deployment webapp -n production**
   b. **Exit : wq!**

```
root@cka-cluster380-master1:~# kubectl edit deployment webapp -n production
deployment.apps/webapp edited
root@cka-cluster380-master1:~#
```

2. Change the image version to **1.11.3**
   a. **kubectl edit deployment webapp -n production**
   b. **Exit : wq!**

```
root@cka-cluster380-master1:~# kubectl edit deployment webapp -n production
deployment.apps/webapp edited
root@cka-cluster380-master1:~# kubectl edit deployment webapp -n production
deployment.apps/webapp edited
root@cka-cluster380-master1:~#
```

3. Rollback to previous version
   a. **kubectl rollout undo deployment webapp -n production**

4. Check the rollout history to confirm that rollout has happened
   a. **kubectl rollout history deployment webapp -n production**
   b. **kubectl get rs -n production –o wide**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl rollout history deployment webapp -n production
deployment.apps/webapp
REVISION  CHANGE-CAUSE
2         <none>
3         <none>

root@cka-cluster380-master1:~# kubectl get rs -n production -o wide
NAME                DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES                              SELECTOR
webapp-6b59bfddc6   0         0         0       7m44s   webapp       evolvecybertraining/nginx:1.11.3    pod-template-hash=6b59bfddc
6,run=webapp
webapp-f9d777846    1         1         1       10d     webapp       evolvecybertraining/nginx:1.11.1    pod-template-hash=f9d777846
,run=webapp
root@cka-cluster380-master1:~#
```

GREAT!

Task #20
Complete the following:

- Launch a deployment
- Name: **web-deploy**
- Image: **nginx**
- Expose via a service: **web-svc**
- Service and Pod should be accessible via DNS records
- Run nslookup command to check the A record of service and pod and redirect the output to /tasks/ORMNZZON07/service.txt and /tasks/ORMNZZON07/pod.txt respectively.

Solution Steps

1. Create a deployment
   a. **kubectl create deployment web-deploy --image=nginx**
2. Expose via a service: web-svc
   a. **kubectl expose deployment web-deploy --name=web-svc --port=80**
3. Create a centos Pod to check nslookup command , install nslookup command
   a. kubectl run -it centos-pod --image=centos
   b. **yum install bind-utils -y**
   c. **CTRL + p+q** (not to kill the container as you log out)
4. Check the A record of the service and write it to the file
   a. kubectl get services
   b. **kubectl exec centos-pod -- nslookup web-svc> /tasks/ORMNZZON07/service.txt**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# cat /tasks/ORMNZZON07/service.txt
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   web-svc.default.svc.cluster.local
Address: 10.106.6.188

root@cka-cluster380-master1:~#
```

5. Check the A record of the Pod and write it to the file
   a. kubectl get pods -o wide
   b. kubectl exec centos-pod -- nslookup 10-44-0-11.web-svc > /tasks/ORMNZZON07/pod.txt

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# cat /tasks/ORMNZZON07/pod.txt
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   10-44-0-11.web-svc.default.svc.cluster.local
Address: 10.44.0.11

root@cka-cluster380-master1:~#
```

GREAT!

Task #21
Create YAML formatted manifest for a **deployment** as follows:

- Name: **galaxy05**
- **2** replicas of **httpd** image with the **label: version=development**

  Save a copy of YAML file to /tasks/IOABEFZE21/deployment.yaml

Solution Steps

1. Create the deployment yaml file
   a. **kubectl create deployment galaxy05 --image=httpd --replicas=2 --dry-run=client -o yaml > /tasks/IOABEFZE21/deployment.yaml**
2. Add the label to the .yaml file **under pod template**
   a. **vi /tasks/IOABEFZE21/deployment.yaml**

GREAT!

Task #22

- Reconfigure **jenkins** deployment and add port specification named **http** for **TCP** port **8080**.
- Expose the port **http** with **NodePort** service, name it **jenkins-service**

Solution Steps

1. Learn how port configuration is in .yaml format
   a. **kubectl run test-pod –image=nginx --port=8080 --dry-run=client -o yaml**
2. Paste that port info into deployment .yaml file under container
   a. **kubectl edit deployment jenkins**
   b. **Wq!**

```
    template:
      metadata:
        creationTimestamp: null
        labels:
          app: jenkins
      spec:
        containers:
        - image: jenkins/jenkins
          imagePullPolicy: Always
          name: jenkins
          ports:
          - containerPort: 8080
            name: http
            protocol: TCP
          resources: {}
```

3. We will expose the deployment
   a. **kubectl expose deployment jenkins --name=jenkins-service --type=NodePort --target-port=http**
4. Confirm that the service is running
   a. **kubectl get services**

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl get services
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
gitlab-svc        ClusterIP   10.100.42.189   <none>        80/TCP           23h
jenkins-service   NodePort    10.103.76.22    <none>        8080:25936/TCP   18s
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP          11d
probe-http        ClusterIP   10.100.99.43    <none>        80/TCP           11d
```

```
web-svc          ClusterIP   10.106.6.188    <none>        80/TCP          156m
root@cka-cluster380-master1:~#
```

GREAT!

Authentication / Authorization

Task #23

Update ServiceAccount at **prime** namespace which will embed **ImagePullSecret** to Pods

Use below credentials for **ImagePullSecret**:

- **pullai2hew**
- **3a131780-67a1-405a-a153-edb768c14467**

Use **pullai2hew/prime** image for testing

Solution Steps

1. Create a secret for private repository as explained here on prime namespace (**search string : ImagePullSecret) Third link: Configure Service Accounts for Pods**
   a. **kubectl create secret docker-registry myregistrykey --docker-username=pullai2hew --docker-password=3a131780-67a1-405a-a153-edb768c14467 -n prime**

2. Now we will need to **patch** the default service account (as indicated on the resource page)
   a. **kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "myregistrykey"}]}' -n prime**

3. Now we will create the test pod on namespace: prime with private image: pullai2hew/prime
   a. **kubectl run test --image=pullai2hew/prime -n prime**

   It should work now

   GREAT!

Network Policy

Task #24

You need to allow a Pod to communicate with the **awx** and **nagios** Pods but nothing else.

Edit the **deploy** Pod to use a NetworkPolicy that will allow it to send and receive traffic only to and from **awx** and **nagios** Pods.

All Pods and NetworkPolicies are deployed in **tools** namespace.

All required NetworkPolicies are already created and ready for use. You should not create, modify or delete any NetworkPolicies.

Solution Steps

1. We get all the network policies on namespace:tools
   a. **kubectl get netpol -n tools**

```
TERMINAL   PROBLEMS   OUTPUT   PORTS   DEBUG CONSOLE
root@cka-cluster380-master1:~/metrics-server# kubectl get netpol -n tools
NAME                 POD-SELECTOR   AGE
awx-netpol           tool=awx       12d
default-deny-netpol  <none>         12d
deploy-netpol        tool=deploy    12d
monitor-netpol       tool=monitor   12d
root@cka-cluster380-master1:~/metrics-server#
```

2. We would like to check the .yaml format of network policies one by one.
   a. **kubectl get netpol deploy-netpol -n tools -o yaml**

```
spec:
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tool: awx
    - podSelector:
        matchLabels:
          tool: monitor
  podSelector:
    matchLabels:
      tool: deploy
  policyTypes:
  - Ingress
```

   b. **kubectl get netpol awx-netpol-n tools -o yaml**

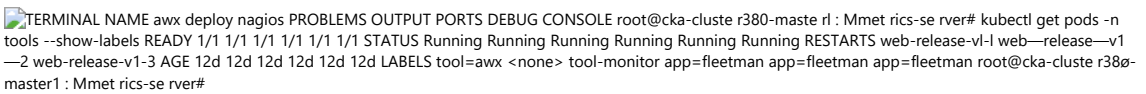
```
spec:
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tool: monitor
    - podSelector:
        matchLabels:
          tool: deploy
  podSelector:
    matchLabels:
      tool: awx
  policyTypes:
  - Ingress
```

c. **kubectl get netpol monitor-netpol -n tools -o yaml**

```
spec:
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tool: awx
    - podSelector:
        matchLabels:
          tool: deploy
  podSelector:
    matchLabels:
      tool: monitor
  policyTypes:
  - Ingress
```

d. We confirm that these policies cumulatively:
   i. Allows communication to the pods labeled:
      1. tool: awx
      2. tool: monitor
      3. tool:deploy
   ii. From pods label
      1. tool: deploy
      2. tool: awx
      3. tool: monitor

3. We check the labels of the pods in the namespace:tools
   a. **kubectl get pods -n tools --show-labels**



4. To ensure communication TO and FROM **awx** and **nagios** pods, we label the deploy pod with **tool:deploy**
   a. **kubectl label pod deploy tool=deploy -n tools**

5. We check the labels and IP addresses of the pods on namespace:tools:
   a. **kubectl get pods -n tools --show-labels -n tools -o wide**

```
root@cka-cluster380-master1:~/metrics-server# kubectl get pods -n tools --show-labels  -n tools -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP           NODE                  NOMINATED NODE   READINESS GATES   LABELS
awx               1/1     Running   0          12d   10.36.0.7    cka-cluster380-node2   <none>           <none>            tool=awx
deploy            1/1     Running   0          12d   10.40.0.9    cka-cluster380-node1   <none>           <none>            tool=deploy
nagios            1/1     Running   0          12d   10.36.0.8    cka-cluster380-node2   <none>           <none>            tool=monitor
web-release-v1-1  1/1     Running   0          12d   10.40.0.5    cka-cluster380-node1   <none>           <none>            app=fleetman
```

```
web-release-v1-2   1/1   Running   0   12d   10.36.0.4   cka-cluster380-node2   <none>   <none>   app=fleetman
web-release-v1-3   1/1   Running   0   12d   10.40.0.6   cka-cluster380-node1   <none>   <none>   app=fleetman
root@cka-cluster380-master1:~/metrics-server#
```

6. We check the communication FROM awx pod TO deploy pod
   a. **kubectl exec awx -n tools -- ping 10.40.0.9**

```
root@cka-cluster380-master1:~/metrics-server# kubectl exec awx -n tools -- ping 10.40.0.9
PING 10.40.0.9 (10.40.0.9) 56(84) bytes of data.
64 bytes from 10.40.0.9: icmp_seq=1 ttl=64 time=1.01 ms
64 bytes from 10.40.0.9: icmp_seq=2 ttl=64 time=0.379 ms
64 bytes from 10.40.0.9: icmp_seq=3 ttl=64 time=0.486 ms
^C
root@cka-cluster380-master1:~/metrics-server#
```

WORKS!!

7. We check the communication FROM deploy pod TO nagios pod
   1. **kubectl exec deploy -n tools -- ping 10.36.0.8**

Task #25

In a namespace called **blackbird**, create a NetworkPolicy which:

- Is named **allow-80**
- Will allow connections from Pods in the namespace to any Pod at port TCP/80
- Deny connections to any other ports
- Deny any connections from other namespaces

Solution Steps

1. We get the pod IP's and labels
   a. **kubectl get pods -n blackbird --show-labels -o wide**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl get pods -n blackbird --show-labels -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP           NODE                   NOMINATED NODE   READINESS GATES   LABELS
black   1/1     Running   0          15d   10.36.0.10   cka-cluster380-node2   <none>           <none>            run=black
call    1/1     Running   1          15d   10.36.0.9    cka-cluster380-node2   <none>           <none>            run=call
data    1/1     Running   0          15d   10.40.0.11   cka-cluster380-node1   <none>           <none>            run=data
root@cka-cluster380-master1:~#
```

2. We label all three nodes with run=eighty
   a. **kubectl label pod black -n blackbird env=eighty**
   b. **kubectl label pod call -n blackbird env=eighty**

c. **kubectl label pod data-n blackbird env=eighty**

3. Create a .yaml file for the network policy
   a. **vi test-netpol.yaml**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-80
spec:
  podSelector:
    matchLabels:
      env: eighty
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: eighty
      ports:
        - protocol: TCP
          port: 80
~
```

==========

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-80
spec:
 podSelector:
  matchLabels:
   env: eighty
 ingress:
 - from:
  - podSelector:
     matchLabels:
      env: eighty
  ports:
  - protocol: TCP
   port: 80
```

==========

4. Apply this .yaml to create a network policy resource
   a. **kubectl apply -f test-netpol.yaml**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get netpol -n blackbird
NAME       POD-SELECTOR   AGE
allow-80   env=eighty     18s
root@cka-cluster380-master1:~#
```

5. We create a test pod to check if it can curl the Pods
   a. **kubectl run centos-test --image=centos -n blackbird -- sleep 1000000**

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
root@cka-cluster380-master1:~# kubectl get pods -n blackbird --show-labels -o wide
NAME          READY   STATUS    RESTARTS   AGE     IP            NODE                    NOMINATED NODE   READINESS GATES   LABELS
black         1/1     Running   0          15d     10.36.0.10    cka-cluster380-node2    <none>           <none>            env=eighty,run=black
call          1/1     Running   1          15d     10.36.0.9     cka-cluster380-node2    <none>           <none>            env=eighty,run=call
centos-test   1/1     Running   0          2m40s   10.36.0.6     cka-cluster380-node2    <none>           <none>            run=centos-test
data          1/1     Running   0          15d     10.40.0.11    cka-cluster380-node1    <none>           <none>            env=eighty,run=data
root@cka-cluster380-master1:~#
```

a. We see centos-test pod cannot communicate with pods: It CANNOT

```
root@cka-cluster380-master1:~# kubectl exec centos-test  -n blackbird -- curl 10.36.0.9
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:19 --:--:--     0
```

```
root@cka-cluster380-master1:~# kubectl exec centos-test  -n blackbird -- curl 10.40.0.11
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:09 --:--:--     0
```

```
root@cka-cluster380-master1:~# kubectl exec centos-test  -n blackbird -- curl 10.36.0.10
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:06 --:--:--     0
```

6. We check if the pods with env=eighty label can communicate with each other

a. From nginx >> Centos – NOT CONNECTING

```
root@cka-cluster380-master1:~# kubectl exec black  -n blackbird -- curl 10.36.0.9
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0curl: (7) Failed to connect to 10.36.0.9 port 80: Connection refused
command terminated with exit code 7
```

a. From nginx >> Test Centos – NOT CONNECTING

```
root@cka-cluster380-master1:~# kubectl exec black  -n blackbird -- curl 10.36.0.9
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0curl: (7) Failed to connect to 10.36.0.9 port 80: Connection refused
command terminated with exit code 7
```

a. From nginx >> nginx – CONNECTING

```
TERMINAL    PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE

root@cka-cluster380-master1:~# kubectl exec black  -n blackbird -- curl 10.40.0.11
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100     6  100     6    0     0   1200      0 --:--:-- --:--:-- --:--:--  1500
Hello
root@cka-cluster380-master1:~#
```

SOLUTIONS STEPS CORRECT VERSION

1. Create the .yaml file below as indicated on K8 resource page (search string: **Network Policy** - delete the unnecessary parts)

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    DEBUG CONSOLE

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-80
  namespace: blackbird
spec:
  policyTypes:
  - Ingress
```

```
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          ns: blackbird
    ports:
    - protocol: TCP
      port: 80
  podSelector: {}
~
~
~
~
~
~
~
~
~
```

2. Create the resource
    a. kubectl apply -f test-netpol.2.yaml

3. Label the blackbird namespace
    a. kubectl label ns blackbird ns=blackbird
4. Check if pods can connect to each other

```
ikambarov@cloudshell:~$
ikambarov@cloudshell:~$
ikambarov@cloudshell:~$ wget https://get.helm.sh/helm-v3.7.1-linux-amd64.tar.gz
--2021-11-06 21:50:44--  https://get.helm.sh/helm-v3.7.1-linux-amd64.tar.gz
Resolving get.helm.sh (get.helm.sh)... 152.195.19.97, 2606:2800:11f:1cb7:261b:1f9c:2074:3c
Connecting to get.helm.sh (get.helm.sh)|152.195.19.97|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13862382 (13M) [application/x-tar]
Saving to: 'helm-v3.7.1-linux-amd64.tar.gz'

helm-v3.7.1-linux-amd64.tar.gz  100%[=====================================================>]  13.22

2021-11-06 21:50:44 (112 MB/s) - 'helm-v3.7.1-linux-amd64.tar.gz' saved [13862382/13862382]

ikambarov@cloudshell:~$ tar -xf helm-v3.7.1-linux-amd64.tar.gz
ikambarov@cloudshell:~$
ikambarov@cloudshell:~$ ls
archive  charts  file1  helm-v3.7.1-linux-amd64.tar.gz  jenkins  k8-playground  linux-amd64  README
ikambarov@cloudshell:~$
```

Etcd cluster backup