CKA EXAM PREPARATION GUIDE

Important Notes and Tips

MARCH 13, 2020

RAVIKUMAR, RAMASAMY

Table of Contents

Reference Courses:

- https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/
- https://www.udemy.com/course/certified-kubernetes-administrator/

Final exam preparation:

- Book the date: -
  - https://training.cncf.io/portal >> schedule exam >> distribution ubuntu 16 >> cka-english (select) >> from & time zone >> handbook >> confirm reservation
  - www.examslocal.com/linuxfoundation >> 15 mins prior >> Launch exam >> exam to start
  - webcam >> chrome >> internet access >> chrome plugin installation >> identity card (passport / discard)
- Import the **K8S_BookMarks.html** to your chrome (To save time during exam else it would eat up our time to search for each topic that want to refer to)
- Make sure that in task manager all the other tasks are closed except chrome prior to exam start time
- Make sure that your laptop got enough charging and power back
- Make sure that you have good internet bandwidth as well

Exam Restrictions:

- Exam will be running for 3 continues hours. Try to avoid taking breaks during exam.
- Exam pass percentage is 74% and given with 24 tasks. Roughly if you get 18 to 19 questions right, you would exceed the % expectation.
- You can keep a glass of water in a transparent glass during exam.
- You are not allowed to have any other gadgets on the exam table and the ambience must be noise free
- Proctor will continuously monitor you throughout the exam
- Try avoiding murmuring the questions and avoid covering your mouth with your hands (even if you have a habit of)
- Try maintaining placing your face in the mid of the camera. Sometimes we adjust our posture that may lead to move away from camera focus. Proctor will bare for three times. The third time will be the final warning. Fourth time, if we repeat the same then he will terminate the exam session and will be disqualified.

Import pointer for Exam Terminal:

- Exam will contain 6 K8s clusters
- Each question will start with the context switching command like
  kubectl config use-context k8s
  Please ensure that you are copying and pasting on the terminal before you start answer any question
- Copy and paste from the K8s documentation and from the question panel will work seamlessly. No need to panic as you can use mouse right click to copy and paste
- There are certain questions that you will need to ssh to the nodes to setup cluster or to launch static pods. The ssh command will be given along with the question itself. You can copy and use as it is.
- Cluster administration commands and tasks will be done using root id. The root sudo command will be given with the question itself. Please do not use your own sudo and practice copy the command from the question panel
- When you login to the Exam terminal, you will be given with the student id on one of the Ubuntu machine that will have the available clusters configured in the kubeconfig
  - **student@<machine> $**
  - Mostly you will be performing all your tasks here only.
  - In case of ssh to any other nodes and perform administration activities then you would have,
    - **student@<machine>$** -- via ssh--> **student@<target_node>$** --su to root--> **root@<target_node>$**
    - Please remember that two stages for switching you have done from your exam terminal via the above steps.
    - Ensure when you are done with you task in the target node, you type **twice exit** to back to your exam terminal student console.
  - Important point to note, **please do not exit third time as it will lead to close your exam terminal** and again you will need to open the Exam terminal
    - Your tasks would have saved in the exam terminal when you reopen the session will get it back
    - However, the **ephemeral data on the earlier terminal like the aliases that you set will be lost.**
    - In that case, please redo the setting of aliases in the newly loaded terminal

Important Tip – Time management:

- Primary Focus is Time Management throughout the exam.
- Setting up aliases & Autocompletions would really save a lot of or time.

- First step upon starting your exam window before read your first question, please do this step
  - Use the K8s cheatsheet bash profile autocompletion
  - Please redo this if you exit from the exam terminal in the mid of your exam and relaunch the exam terminal

- Second step set the vim shortcuts profile, this is to ease the yaml indentation correction via tab press and view the line numbers in the vi editor.

  vim ~/.vimrc

```
set nu
set ic
set expandtab
set shiftwidth=2
set tabstop=2
```

- Use shortcut forms for any kubectl commands
  k get netpol (kubectl get networkpolicies)
  k get cs (kubectl get componentsstatuses)
  k get csr (kubectl get certificate signing request)

  As we set the kubectl autocompletions, we would not required to set any other aliases for kubectl commands rather you can use the **tab** to autocomplete the commands

  Note: Please remember **if you do any typo error** in the mid of the kubectl command then the **autocompletion will not work.**

Important Tip – Answering questions:

- Try to use the notepad available in the exam console. You can close the notepad once you add some notes. Then you can reopen anytime.
  - Especially if you are asked to create a static pod in any target nodes then dry run the static pod as yaml from exam terminal and copy the yaml to notepad
  - Once you logged in to the target terminal that you need to launch the static pod,
  - Go to /etc/kubernetes/manifests/ and vi static-pod.yaml and copy the static pod creation yaml content form the notepad and save the file

- If you are trying to dry-run the pod definition as yaml along with the commands and arguments, please place the --dry-run -o yaml prior -- commands section, else the dry run yaml content will be empty.
  - **Example**: Correct command

```
k run busybox --image=busybox --restart=Never --dry-run -o yaml -- sh -c
“sleep 3600” > busybox-pod.yaml
```

  - Resultant yaml Output will be as given below
    cat busybox-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - sh
    - -c
    - sleep 3600
    image: busybox
    name: busybox
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

  - **Example**: Wrong pattern (dry run param is used at the end of the pod creation command)

```
k run busybox --image=busybox --restart=Never -- sh -c “sleep 3600” --dry-run
-o yaml > busybox-pod.yaml
```

  - Resultant yaml will be as given below which is wrong
    cat busybox-pod.yaml

```
pod/busybox created
```

- If you are trying to dry-run the pod definition as yaml and the pod to be created in any specific namespace.
    - For example, in **qa** namespace, you want to create busybox pod, you have opted to dry run the yaml first and apply it
    - Please note that if you use the **--namespace=qa** along with the dry-run command option, it will not be effective and the resultant yaml will not contain the namespace section added
    - Example of wrong usage,

```
k run busybox --image=busybox --restart=Never --namespace=qa --dry-run -o
yaml -- sh -c “sleep 3600” > busybox-pod.yaml
```

- Correct usage is dry run the yaml without the namespace and when apply the yaml use the --namespace=qa parameter.

```
k run busybox --image=busybox --restart=Never --dry-run -o yaml -- sh -c
“sleep 3600” > busybox-pod.yaml

k apply -f busybox-pod.yaml --namespace=qa
```

- Please note that busybox container image will not have the required nettool package installed in it. Hence use busybox:1.28 container image if you are required to any pod / service networking troubleshooting or dns tasks.

- Useful Command shortcut and references

```
# Useful help commands
kubectl explain pod --recursive | less
kubectl describe pod <pod_name> | grep -i events -A 10
kubectl run -h | grep -i "#" -A 2
kubectl create job -h | less
k explain job.spec --recursive | less
kubectl config -h | less

# useful exec commands
kubectl exec -it busybox -- /bin/sh wget -O- 10-40-0-1.<namespace>.pod
kubectl exec -it busybox -- /bin/sh -c -- echo "Hello CKA"

# Kubectl run various restart options
#POD imperative command
k run <podname> --restart=Never

#Deployment imperative command
k run <deploymentname>
```

```
#Job imperative command
k run <jobname> --restart=OnFailure

#Job imperative command to run parallel X number and complete after Y times
k run <Jobname> --restart=OnFailure  | dry-run as yaml  & edit  yaml to add
.spec.parallelism : X  | .spec.completions : Y

#CronJob imperative command
k run <podname> --schedule=*/1 * * * * --restart=OnFailure

# Useful labelling commands
# Assigning the label to nodes
k label node node01 disktype=ssd

# Deleting the label
k label node node01 disktype-

# List the pods with label info
k get po --show-labels

# List the pods filtered with a specific matching label
k get po -l disktype=ssd
-----------------------------------------------------------------------------------------------------------------
## useful systemctl commands

# To see running processes
systemctl
systemctl | grep <service>

# Check status for a system process
systemctl status | grep kube

#Tell systemd to start at boot
systemctl enable <process name>

# Edit service
systemctl edit kubelet.service
or
vim /etc/systemd/system/kubelet.service

# If you change a systemd service, please do the following
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet.service
```

```
# If the service was not started
systemctl enable kubelet
systemctl start kubelet

# Display unit file
systemctl cat kubelet.service

#Show low level properties of a unit
systemctl show cron.service
```

Exam Topics sort by weightage in desc order:

5% - Scheduling
5% - Logging/Monitoring
7% - Storage
8% - Application Lifecycle Management
10% - Troubleshooting
11% - Cluster Maintenance
11% - Networking
12% - Installation, Configuration & Validation
12% - Security
19% - Core Concepts

-- General notes by Topic ---

Domain 1: core concepts

- Create a single/multiple pod
  - Familiar with commands and arguments for pods
  - Sleep command / expose instruction / multi container in a given pod
- If you are unsure of the spec or parameters of a yaml, always use **kubectl explain <resource>.<key>**

json path useful scenarios:

- Medium article reference is here for the better understanding of json path structure and understanding the array and map concepts.

Additional scenarios that is for your practice:

1. Retrieve the name,OS Image and the Internal IP address of the nodes in a new line.

```
master $ k get nodes -o jsonpath="{range .items[*]} {.metadata.name}
{.status.nodeInfo.osImage}
{.status.addresses[?(@.type=='InternalIP')].address}{\"\n\"}{end}"

Result:
master Ubuntu 16.04.6 LTS 172.17.0.61
node01 Ubuntu 16.04.6 LTS 172.17.0.48
```

- Please retrieve the nodes spec to file using **k get nodes -o yaml**
- Understand the structure of the yaml elements on the screen
- Identify the list and map elements from the yaml. Generally, if you are listing all the nodes then it will be listed as array of items.
- If you want to print the data into new lines, then you should range over the items.

2) Retrieve the nodes that are schedulable

There are many ways you can provide solution for this scenario.

Solution 1: Range over node items > Get node metadata name > List the taint info | Exclude NoSchedule taint effect via grep command

```
master $ k get nodes -o jsonpath="{range .items[*]}{.metadata.name}
{.spec.taints[*].effect} {\"\n\"}{end}" | grep -v NoSchedule

Result:
node01
```

Solution 2: Get nodes as custom columns > Get node metadata name > Get the taint info | Exclude NoSchedule taint effect via grep command

```
master $ k get no -o custom-
columns=NODENAME:.metadata.name,TAINTS:.spec.taints[*].effect | grep -
vNoSchedule

Result:
NODENAME   TAINTS
node01     <none>
```

3) Retrieve the nodes that are not schedulable and the time that they are being made as unschedulable.

Solution: Range over node items > Get node metadata name > Filter the taint effect is equal to NoSchedule and fetch the timeAdded map key to list the time that is being made unscheduled or drained.

```
master $ k get no -o jsonpath="{range .items[*]}{.metadata.name}
{.spec.taints[?(@.effect=='NoSchedule')].timeAdded} {\"\n\"} {end}" | awk '$2 !=""'

Result:
node01 2020-04-03T03:33:03Z
```

4) Check the Image version of the nginx pod without the describe command?

```
master $ kubectl get po nginx -o jsonpath='{.spec.containers[].image}{"\n"}'
Result:
nginx
```

Busybox test pod creation note:

Create a busybox pod with command sleep 3600?

```
kubectl run busybox --image=busybox -- /bin/sh -c "sleep 3600"
kubectl run --generator=run-pod/v1 busybox --image=busybox --command -- sleep 3600
```

Domain 2 : Application Life cycle management

Important topics (rolling updates | rollbacks | scaling | record instruction | Jobs and cron Jobs)

i) jobs >> parallelism, completions and cronjob >> basics about creating cron job

ii) labels and selectors >> how to add a label to pods and nodes >> how you can search for pods with specific labels >> how you can search for pods associated with a service based on labels

scenario 1:- Update Images

Change the Image version to 1.15-alpine for the pod you just created and verify the image version is updated?

```
kubectl set image pod/nginx nginx=nginx:1.15-alpine --record
kubectl describe po nginx
kubectl get po nginx -w # watch it
```

<u>scenario 2 :- Performing rolling updates</u>

[https://kubernetes.io/docs/tasks/run-application/rolling-update-replication-controller/]

[Understand Deployments and how to perform rolling updates and rollbacks]

[wish to change the image of the container for the deployment to a new version such as image: nginx to image: nginx:1.7.1]

```
kubectl set image deplyoment/myapp-deployment nginx=nginx:1.7.1 --record
# --record is to record the status of the deployment upgrade
```

[Check the status of the deployment rollouts]

```
kubectl rollout status deployment/myapp-deployment
```

[Check the rollout history of the deployment]

```
kubectl rollout history deployment/myapp-deployment
```

[Check the replicasets during the rolling updates]

```
kubectl get rs
```

[In case of error in the deployment of your app, if you want to rollback to previous version of the deployment then run rollout to undo]

```
kubectl rollout undo deployment/myapp-deployment

# if you want to rollback to specific version of the deployment then use --to-
revision=<version#>

kubectl rollout undo deployment/myapp-deployment --to-revision=1
```

<u>scenario 3:-</u> Cron job that will be executed every minutes that prints the current date and Hello from the Kubernetes cluster

https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/

```
kubectl run hello --schedule="*/1 * * * *" --restart=OnFailure --image=busybox -- /bin/sh -c
"date;
echo Hello from the Kubernetes cluster"

# --restart=OnFailure represents the run is for Job and –schedule represents this Job is a
cron
```

scenario 5 :- ConfigMaps

General Exam scenario would be creating a config map from given literal and attach that to pod as environment variable or as volume mount.

https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

- You can use ConfigMap as the Key Value Pair to inject the env variable to the POD definition.

  ```
  kubectl create configmap app-config --from-literal=app_color=blue --from-
  literal=app_type=prod
  ```

  We can create config map from file as well. Lets assume the app_config.properties has the key value pairs defined in it and we are about to create a configMap using the file.

  ```
  kubectl create configmap app-config --from-file=app_config.properties
  ```

  List the config maps created

  ```
  kubectl get configmaps
  ```

- You can refer to the kubectl bookmarks for the reference K8s documentation page.

scenario 6 :- Secrets

https://kubernetes.io/docs/concepts/configuration/secret/

General Exam scenario would be creating a config map from given literal and attach that to pod as environment variable or as volume mount.

In case, you want to pass on the env variable such as DB Host, User, Password to the web application then use Kubernetes Secrets.

```
kubectl create secret generic app-secret --from-literal=DB_host=mysql --from-
literal=DB_user=root --from-literal=DB_passwd=mysql
```

```
kubectl create secret generic dev-db-secret --from-literal=username=devuser
--from-literal=password='S!B\*d$zDsb'

# Secret from file
kubectl create secret generic app-secret --from-file=app_secret.properties
```

Store the secret in encoded format

echo -n 'mysql' | base64 (repeat this for user and password)

cat secret.yaml

echo -n 'hashvalue' | base64 --decode [if you want to decode the secret value]

List the Secrets

```
kubectl get secrets
```

scenario 7 : - Scale the deployment to 4 replicas

```
kubectl scale deployments/kubernetes-bootcamp --replicas=4
```

There is a concept called horizontal pod autoscaling (HPA) and is important for exam point of view that you will be asked to scale your deployment to max of 3 replicas if the cpu usage hits 80%

```
kubectl autoscale deployment/my-nginx --min=1 --max=3 --cpu-percent=80
```

List the HPA objects

```
kubectl get hpa
```

$ kubectl edit (replicas object)

scenario 8 :- [Define Environment Variables for a Container] >> [Inject data to the applications] >>

https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/

```
# Use imperative commands to achieve this
master $ k run nginx-envpod --image=nginx --restart=Never --env=ENV_APP=WEBAPP
```

```
pod/nginx-envpod created

#Verify the env var is added to the container
master $ kubectl exec -it nginx-envpod -- printenv | grep ENV_APP
ENV_APP=WEBAPP
```

scenario 9 :- Labelling the nodes, pods and deployments

```
# Label node with key env and type as ssd
master $ k label nodes node01 type=ssd
node/node01 labeled

# List the nodes with label
master $ k get no --show-labels

#List the running deployments
master $ k get deploy --show-labels
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
nginx      3/3           3                    3              61s   run=nginx
# Label deployment with key env and value as dev
master $ k label deployments nginx  env=dev
deployment.extensions/nginx labelled

master $ k get deploy --show-labels
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
nginx     3/3                3                  3             81s   env=dev,run=nginx

#List the deployment that has the label env=dev
master $ k get deploy -l env=dev
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx     3/3                3                  3          4m35s
```

scenario 10 :- Understand how resource limits can affect Pod Scheduling

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/

Create a pod with the maximum resource limit and request specific amount of resource.

Max CPU is 4 , Max Memory is 4 Gi and Min cpu requested by pod is 2 and memory is 2 Gi

```
master $ k run nginx-limit-req-pod --image=nginx --restart=Never --limits=cpu=4,memory=4Gi
--requests=cpu=2,memory=2Gi
pod/nginx-limit-req-pod created

# Verify that the pod is created with the specified limit and request range
master $ k describe pod nginx-limit-req-pod | grep -i limits -A 5
    Limits:
      cpu:     4
      memory:  4Gi
    Requests:
      cpu:        2
      memory:     2Gi
```

Note: Always practice using the imperative kubectl command to save your time in the exam.

Domain 3 :-Networking

Important points

Familiarize with how to create a service and understand diff types of services clusterIP and NodePort

- ➢ How to set a port for NodePort >> Familiar with port and target port within a service
- ➢ If you expose any deployment or pod without explicitly defining the type , it will be defaulted to ClusterIP service
- ➢ For any service,
  - o **port** is Service's own front end port which client will be accessing
  - o **target-port** is back end application's port that are running inside the container
- ➢ A service will internally create an End point object that collects the pool of POD ips that are matching the service's selector label and create an Ip table to route the actual request.
- ➢ Whenever you expose any service, ensure the selector used for the service is correct.
  - o The Service will not be able to directly connect to the pods rather it connects via end points
  - o If the selector doesn't match with the running pods label, then there will be no underlying end points created for the service. Hence the actual client request to the service will timeout.
  - o

Example Scenarios

Service - Scenarios

- **Example Scenario:** There is a nginx deployment running and the underlying pods are with label run=nginx and expose the service as nginx-dev-service type cluster IP and the service port should be 8080. Note that the target nginx service is running with port 80 in the container.

```
# List the deployment with show-labels option
master $ k get deployments --show-labels
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
nginx                   3/3          3                  3          29m   env=dev,run=nginx
nginx-limit-req-pod  1/1          1                  1          17m   run=nginx-limit-req-pod

# Filter the pods to confirm the pods are running with run=nginx label

master $ k get po -l run=nginx -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP         NODE     NOMINATED
NODE   READINESS GATES
nginx-7db9fccd9b-q96c7   1/1     Running   0         41m   10.44.0.3   node01   <none>
<none>
nginx-7db9fccd9b-w9nfh   1/1     Running   0         41m   10.44.0.2   node01   <none>
<none>
nginx-7db9fccd9b-x6qcg   1/1     Running   0         41m   10.44.0.4   node01   <none>
<none>



# Expose the deployment with the selector option
master $ k expose deployment nginx --name=nginx-dev-service --port=8080 --target-port=80 --
selector=run=nginx

# List the Service that is created
master $ k get svc -o wide
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   SELECTOR
nginx-dev-service   ClusterIP   10.103.71.108   <none>        8080/TCP   15s   run=nginx

# Get the underlying end points created by the service nginx-dev-service
# The End points are created by filtering the pod Ips with the selector given in the
expose command. The target-port of the expose command is used to for port
forwarding rule in the endpoint

master $ k get ep
NAME                         ENDPOINTS                              AGE
nginx-dev-service   10.44.0.2:80,10.44.0.3:80,10.44.0.4:80   2m50s
```

- Verify the service accessibility using busybox pod

```
# Launch the busybox pod with image busybox:1..28  for any serevice / dns checks

master $ kubectl run servicecheck --image=busybox:1.28 --restart=Never -- sleep 3600
pod/servicecheck created

# We will need to test the service accessibility from a client. We will use the
servicecheck pod and the client. Get into the servicecheck pod and access the nginx-
dev-service over the service's front end port 8080

master $ k exec -it servicecheck -- wget --spider --timeout 1 nginx-dev-service:8080
Connecting to nginx-dev-service:8080 (10.111.45.86:8080)

# Connection is successful to the nginx app running inside pod's container with port 80
```

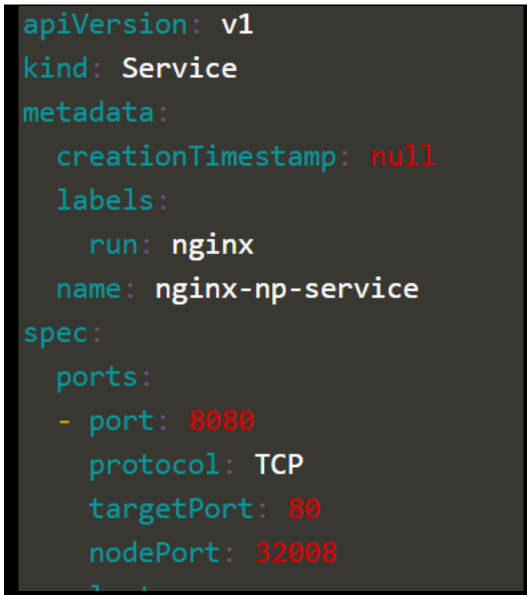
- Assume that when you expose the deployment with wrong selector then the result would be,
  - Service creation will be successful
  - Underlying endpoints would not be created. Hence there will be no Ip tables for request routing from service to target pods
  - Client connection will be unsuccessful
  - Let's simulate the scenario, take the resources from the above example.

```
# Expose the deployment with the wrong selector option app=web
master $ k expose deployment nginx --name=nginx-wrong-service --port=8080 --target-
port=80 --selector=app=web
service/nginx-wrong-service exposed

# List the Service that is created
master $ k get svc -o wide
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
SELECTOR
nginx-wrong-service   ClusterIP   10.111.72.149 <none>        8080/TCP   15s   app=web

# Get the underlying end points created by the service nginx-wrong-service
# There is an end object created with no end point pod Ips mapped.
```

```
The End point object couldn't find a running pod with the defined selector label
app=web

master $ k get ep
NAME                        ENDPOINTS                          AGE
nginx-wrong-service            <none>                            80s

# We will need to test the service accessibility from a client. We will use the
servicecheck pod that was created earlier. Get into the servicecheck pod and access
the nginx-wrong-service over the service's front end port 8080

master $ k exec -it servicecheck -- wget --spider --timeout 1 nginx-wrong-service:8080
Connecting to nginx-wrong-service:8080 (10.111.72.149:8080)
wget: download timed out
command terminated with exit code 1

# Failed to connect to the nginx app running inside pod's container with port 80
```

- When you are asked to expose a deployment as NodePort and given with the specific node port then you will need to dry-run the expose imperative command as yaml and manually edit the yaml with the given node port
  - **Example Scenario:** There is a nginx deployment running and the underlying pods are with label run=nginx and expose the service as nginx-np-service type Node Port and the service port should be 8080 and the node port will be 32008. Note that the target nginx service is running with port 80 in the container.

```
# Expose the service as NodePort Type and dry run as yaml

master $ k expose deployment nginx --name=nginx-np-service --type=NodePort --port=8080 -
-target-port=80 --selector=run=nginx --dry-run -o yaml > nginx-np-service.yaml

# Edit the nginx-np-service.yaml file and add the nodePort: 32008 as sibling to the
targetport element to match with the asked node port
```



```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx-np-service
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 80
    nodePort: 32008
```

# Apply the service yaml spec

```
master $ k apply -f nginx-np-service.yaml
service/nginx-np-service created

master $ k get svc -o wide
NAME              TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
SELECTOR
nginx-np-service     NodePort    10.96.234.6     <none>        8080:32008/TCP   11s
run=nginx

master $ k get ep
NAME                         ENDPOINTS                              AGE
nginx-np-service      10.32.0.2:80,10.32.0.3:80,10.32.0.4:80   118s
```

# Verify the ap accessibility over service. Please note that the node port is used to access the service from LB end and from internal cluster if you want to access the service then use the service's own front-end port 8080

```
master $ k exec -it servicecheck -- wget --spider --timeout 1 nginx-np-service:8080
Connecting to nginx-np-service:8080 (10.96.234.6:8080)
```

# Connection is successful to the nginx app running inside pod's container with port 80

DNS – Service and Pod DNS checks

- There would be certain scenarios that you would need to write the DNS record of service or pod in a file
- Make use of the busybox:1.28 pod (servicecheck) that was launched earlier for DNS check as well
- The service's dns records can be retrieved directly querying **nslookup <service-name>**
- Pod's dns records can be retrieved by querying **nslookup <pod-ips>.<namespace>.pod** (Pod Ip's period (.) must be replaced with -)

Example Scenario: Service DNS Check. Record the dns entry of service nginx-dev-service to the file /opt/nginx-dev-service-dns

```
# Make use of the busybox servicecheck pod to nslookup the service
master $ k exec -it servicecheck -- nslookup nginx-dev-service > /opt/nginx-dev-service-dns

# Verify the dns entries logged in the file
master $ cat /opt/nginx-dev-service-dns
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      nginx-dev-service
Address 1: 10.96.141.251 nginx-dev-service.default.svc.cluster.local
```

Example Scenario: Pod DNS Check. Record the dns entry of pod nginx-7db9fccd9b-7b5j7 to the file /opt/nginx-pod-dns

```
# List the pod nginx-7db9fccd9b-7b5j7   and with the wide option to see the pod IP
master $ k get po nginx-7db9fccd9b-7b5j7 -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP          NODE     NOMINATED NODE   READINESS GATES
nginx-7db9fccd9b-7b5j7   1/1     Running   0          48m   10.32.0.2   node01   <none>           <none>

# Retrieve the POD's DNS record
master $ k exec -it servicecheck -- nslookup 10-32-0-2.default.pod > /opt/nginx-pod-dns

# View the content of the pod's dns record file
```

```
master $ cat /opt/nginx-pod-dns
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10-32-0-2.default.pod
Address 1: 10.32.0.2 10-32-0-2.nginx-dev-service.default.svc.cluster.local
```

Network policies

Scenario 1 :- deployment > service expose > map ingress network policy >> testing

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

scenario 2 :- network policy resource (ingress/egress) - default deny/allow all / limits pods.

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

Domain 4:- Scheduling

Important Questions and pointer:

- Schedule a pod on a node which as disktype=SSD
  - NodeSelector >> [understanding labels and selectors is important]
  - Once your dry run the pod add the nodeSelector to he yaml definition. Refer to the Kubernetes docs for reference.

- Fix the broken cluster. All the pods are in pending state,
  - If you are asked to fix the broken cluster where the Pods are in pending state or API Server itself inaccessible
    - There is a high possibility that the kubelet is not referring to the right static pod manifest path in the master node.
  - Verify the static pod path entry **staticPodpath: /etc/kubernetes/manifests** in the kubelet config file `/var/lib/kubelet/config.yaml`

- If the above entry is not there or the entry is pointed to **staticPodpath: BROKEN** then adding/updating the entry **staticPodpath: /etc/kubernetes/manifests** and save the file.
- Restart and enable the kubelet in the node
  - systemctl daemon-reload
  - systemctl restart kubelet
  - systemctl enable kubelet
- Try access the Kube API server by issuing **kubectl get nodes**.
- You will get results now as the master node's kubelet is launched all the master component pod's from the manifest path **/etc/kubernetes/manifests**

➢ Launch a static pod named **static-pod** in node **node01**

- Initially dry run the static pod as yaml from main exam terminal and copy the yaml to exam window notepad

- If you are asked to launch a static pod in any target node, once you ssh to the target node

- Verify the static pod path entry **staticPodpath: /etc/kubernetes/manifests** in the kubelet config file `/var/lib/kubelet/config.yaml`
- If the above entry is not there then adding the entry and save the file.
- Restart and enable the kubelet in the node
  - systemctl daemon-reload
  - systemctl restart kubelet
  - systemctl enable kubelet
- Go to /etc/kubernetes/manifests/ and vi static-pod.yaml and copy the static pod creation yaml content form the notepad and save the file
- Exit from the node and go to the exam terminal look for the pods created

```
master $ k get po
NAME              READY   STATUS    RESTARTS   AGE
static-pod-node01   1/1     Running   0          11s
```

Note: The static pod is differentiated with the name. Always the static pods are appended with the node name that was launched. In our case the created pod is static-pod and is appended with it's node name node01.

Domain 5 :-Security

- ➢ Basic understanding SSL/TLS certificates
- ➢ Certificate Authority
- ➢ Component's Certificate
- ➢ Good understanding on Kebernetes secrets >> create a secret from a literal value >> Create a secret from a file >> Mount Secret to a pod-based Environment variable >> Mount secret to a pod
- ➢ Basics of Authentication and Authorization >> create a role and role binding >> create a cluster role and cluster role binding >> Associate with the user / service account
- ➢ Familiar with pod security context >> runAsUser >> runAsGroup >> fsGroup
- ➢ To verify that the user <ravi> has the access to create pods in namespace dev,

```
master $ kubectl auth can-i create pod --as="ravi" --namespace dev
Result:
no
```

Domain 6 :- Storage

- ➢ Understanding of Volumes
  - o Mostly Empty directory volume will be asked in the exam
  - o You will need to map the empty directory volume to the pod volume mount
- ➢ Creating Persistent Volume and Volume Claim might be asked.
- ➢ Refer to K8s bookmarks' s Storage section for the reference pages.

Domain 7:-Installation, configuration, Validation (Highest weightage 8% in the Exam)

How to setup a kubeadm cluster?

Master Node:

- ▪ Follow the Kubernetes documentation
  - • Ensure you are switched to the respective master node and switched to root user
  - • Install kubelet kubeadm kubectl
  - • Create control-plane with kubeadm
    - o
    - o There are few parameters given with the question like --ignore-preflight-errors. use them

```
master$ kubeadm init --config=/etc/cka-exam.config --ignore-preflight-errors
```

- Copy the Node joining token from the terminal to exam notepad
- Install the Pod network Add on
  - Flannel – Copy the kubectl apply command and apply as is

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65
e414cf26827915/Documentation/kube-flannel.yml
```

  - Please do not add additional add on. Flannel is enough to bring your cluster up and running
- Ensure you enable kubelet as a precautionary measure to auto start the kubelet process upon node restarts
  - systemctl daemon-reload
  - systemctl restart kubelet
  - systemctl enable kubelet

Worker Node:

- Follow the Kubernetes documentation
  - Ensure you are switched to the respective worker node and switched to root user
  - Install kubelet kubeadm kubectl
  - Join your nodes
  - Ensure you enable kubelet as a precautionary measure to auto start the kubelet process upon node restarts
    - systemctl daemon-reload
    - systemctl restart kubelet
    - systemctl enable kubelet

Verification:

- ssh to master node and issue **k get nodes** command to confirm the Cluster is setup is completed.

ETCD back-up and restore

- Given you the ETCD backup and restore procedure for study purpose
- In the exam you will be asked to backup the ETCD cluster.

# 1. Backup

```
ETCDCTL_API=3 etcdctl --endpoints=https://[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 /tmp/snapshot-pre-boot.db
```

```
# ----------------------------
# Disaster Happens
# ----------------------------
```

# 2. Restore ETCD Snapshot to a new folder

```
ETCDCTL_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--name=master \
--cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key \
--data-dir /var/lib/etcd-from-backup \
--initial-cluster=master=https://127.0.0.1:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-advertise-peer-urls=https://127.0.0.1:2380 \
snapshot restore /tmp/snapshot-pre-boot.db
```

Help Note : ps -aux | grep etcd > you can get the below result and find the required info used to restore

```
etcd --advertise-client-urls=https://172.17.0.65:2379 --cert-file=/etc/kubernetes/pki/etcd/server.crt
--client-cert-auth=true
--data-dir=/var/lib/etcd-from-backup
--initial-advertise-peer-urls=https://172.17.0.65:2380
--initial-cluster=master=https://172.17.0.65:2380
--name=master
--key-file=/etc/kubernetes/pki/etcd/server.key
--listen-client-urls=https://127.0.0.1:2379,https://172.17.0.65:2379
--listen-metrics-urls=http://127.0.0.1:2381 --listen-peer-urls=https://172.17.0.65:2380
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt --peer-client-cert-auth=true
--peer-key-file=/etc/kubernetes/pki/etcd/peer.key
--peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt --snapshot-count=10000
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
```

# 4. Modify /etc/kubernetes/manifests/etcd.yaml

Update --data-dir to use new target location

```
--data-dir=/var/lib/etcd-from-backup
```

```
Update new initial-cluster-token to specify new cluster
```
--initial-cluster-token=etcd-cluster-1
```
Update volumes and volume mounts to point to new path
```
volumeMounts:
- mountPath: /var/lib/etcd-from-backup
name: etcd-data
- mountPath: /etc/kubernetes/pki/etcd
name: etcd-certs
hostNetwork: true
priorityClassName: system-cluster-critical
volumes:
- hostPath:
path: /var/lib/etcd-from-backup
type: DirectoryOrCreate
name: etcd-data
```

----------------------------------------------------------------------------------------------------------------

## Domain 8 :- Logging / Monitoring

- Be familiar with Kubernetes Metric Server
- Know on how to get top utilization metrics of a nodes
  - kubectl top nodes
- Know on how to get top utilization metrics of a pods
  - kubectl top pods
- Metric Server will be pre-installed so no need to worry
- Basic scenarios,
  - Write all the container log lines that are matching with the text 02 in a file /opt/all-containers.log

    ```
    kubectl logs <pod_name> --all-containers | grep "02" > /opt/all-containers.log
    ```

  - List all PV sorted by capacity and write to a file /tmp/pv.txt

    ```
    kubectl get pv --sort-by=.spec.capacity.storage > /tmp/pv.txt
    ```

    Note: use json spec sort-by to answer this kind of questions

  - List all top three hungry pods from **all namespaces** in a file /tmp/cpu-hungry-pods.txt
    ```

```
kubectl top pod --all-namespaces | sort --reverse --key 3 --numeric| head -3 >
/tmp/cpu-hungry-pods.txt
```

Domain 9:- Troubleshooting

- Monitor cluster Component and it's logs
- Know the status of the cluster componenets
  - kubectl get cs
- journalctl - see the logs for troubleshooting

Useful approach towards debugging the broken cluster

Scenario: No one can reach the k8s master Node. Fix it

- kubectl get cs >> Error like did you specify the right host or port?
- kubectl get pods >> Error like did you specify the right host or port ??
- systemctl status kube-apiserver >> Unit kube-apiserver.service could not be found
- systemctl status kubelet >> if kubelet is running then check >> docker ps >> if you couldn't see the results then the control-plane containers are not running
- Go to /etc/kubernetes/manifests/ >> ls -lrt *.yaml
- Verify that the control-plane manifest files are available
- If everything is fine, then look for the static pod path configuration in the kubelet config
  - ssh to the master node of the broken cluster
  - Verify the static pod path entry **staticPodpath: /etc/kubernetes/manifests** in the kubelet config file `/var/lib/kubelet/config.yaml`
  - If the above entry is not there or the entry is pointed to **staticPodpath: BROKEN** then adding/updating the entry **staticPodpath: /etc/kubernetes/manifests** and save the file.
  - Restart and enable the kubelet in the master node
    - systemctl daemon-reload
    - systemctl restart kubelet
    - systemctl enable kubelet
- You are fixed your cluster. Can verify with the **kubectl get nodes** command in the master node.

Scenario: Troubleshoot worker node failure

- Check Node Status
  - kubectl get nodes
- ssh to the failure node and check the kubelet service status

  - systemctl status kubelet
  - There could be a possibility that the kubelet service is down in the node.
- Restart and enable the kubelet in the node
  - systemctl daemon-reload
  - systemctl restart kubelet
  - systemctl enable kubelet
  - systemctl status kubelet

- Exit from the node and verify the node status from the exam terminal **kubectl get nodes**

Domain 10: Cluster Maintenance

- Basic understanding of drain | cordon | uncordon nodes

  Scenario: Make the worker node node01 as un-schedulable. Ensure all the running pods are evicted from the node

```
master $ kubectl drain node01 --ignore-daemonsets
```

  Scenario: Make the worker node node01 as un-schedulable. Ensure all the running pods are evicted from the node01 and migrated to new node.

```
master $ kubectl drain node01 --ignore-daemonsets --force
```

-- End --