Q1.

Context

A web application requires a specific version of redis to be used as a cache.

Task

Create a pod with the following characteristics, and leave it running when complete:

* The pod must in the web namespace. The namespace has already been created.
* The name of the pod should be cache.
* Use the ifccnf/redis image with the 4.0-alpine tag.
* Expose port 6379.

*Answer:*

kubectl create ns web

kubectl -n web run cache --image=lfccncf/redis:4.0-alpine --port=6379

<Comment> Answer ok, Verification step missing

K get svc,ep –n web

Q2

Context

You are tasked to create a secret and consume the secret in a pod using environment variables as follows:

Task

* Create a secret named another-secret with a key/value pair: key1/value4
* Start a nginx pod named nginx-secret using container image nginx, and add an environment variable exposing the value of the secret key key1 using BEST\_VARIABLE as the name of the environment variable inside the pod

*Answer :*

kubectl create secret generic another-secret --from-literal=key1=value4

apiVersion: v1

kind: Pod

metadata:

name: nginx-secret

spec:

containers:

- name: mycontainer

image: nginx

env:

- name: BEST\_VARIABLE

valueFrom:

secretKeyRef:

name: another-secret

key: key1

restartPolicy: Never

<Comment> Answer ok,

* I hope yaml is created from documentation, if not suggest to use documentation Verification step missing
* Check if secret is accessible via pod

Q3

Task

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

* Create pod named nginx-resources in the pod-resources namespace that request a minimum of 400m CPU and 2Gi memory for its container
* The pod should use the nginx image
* The pod-resources namespace has already been created

*Answer:*

kubectl create ns pod-resources

apiVersion: v1

kind: Pod

metadata:

name: nginx-resources

namespace: pod-resources

spec:

containers:

- name: app

image: nginx

resources:

requests:

memory: "2Gi"

cpu: "400m"

<Comment> Answer ok, Verification step missing

Check the pod resource allocation status by describing the pod

Q4

Context

You are tasked to create a ConfigMap and consume the ConfigMap in a pod using a volume mount.

Task

Please complete the following:

* Create a ConfigMap named some-config containing the key/value pair: key4/value1
* Start a pod named nginx-configmap containing a single container using the nginx image, and mount the key you just created into the pod under directory /some/path.

*Answer :*

*## Create ConfigMap using imperative command ##*

kubectl create configmap some-config --from-literal=key4=value1

*## Create Pod using ConfigMap as Volume Mount ##*

apiVersion: v1

kind: Pod

metadata:

creationTimestamp: null

labels:

run: nginx-configmap

name: nginx-configmap

spec:

containers:

- image: nginx

name: nginx-configmap

volumeMounts:

- name: config-volume

mountPath: /some/path

volumes:

- name: config-volume

configMap:

name: some-config

*## Verify the answer by going to the mount path and see the value of configmap ##*

kubectl exec -it nginx-configmap -- /bin/sh

cat /some/path/key4

<Comment> Answer ok, I hope yaml is created from documentation, if not suggest to use documentation considering the time constraint.

Q5.

Context

Your application’s namespace requires a specific service account to be used.

Task

Update the app-1 deployment in the frontend namespace to run as the app-account1 service account. The service account has already been created.

*Answer:*

*##Create namespace##*

kubectl create ns frontend

*##Create Serviceaccount##*

kubectl create sa app-account1 -n frontend

*##Create deployment with the Serviceaccount in the namespace##*

apiVersion: apps/v1

kind: Deployment

metadata:

name: app-1

namespace: frontend

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

serviceAccountName: app-account1

*##Verification##*

controlplane $ kubectl get deployments -n frontend

NAME READY UP-TO-DATE AVAILABLE AGE

app-1 1/1 1 1 21m

controlplane $

<Comment> In exam, account will be already exists with necessary permission already. We just need to add the account to deployment

Use below command

k set serviceaccount deployment app-1 app-account1 -n frontend

Q6.

Context

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

Task

The desired behavior is to have Kubernetes restart the pod when an endpoint return an HTTP 500 on the /healthz endpoint. The service, probe-pod, should never send traffic to the pod while it is failing. Please complete the following:

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

*Answer:*

*##Create a probe with readiness and liveliness probe. Readiness probe to listen on /started and liveliness on /healthz on port 8080##*

apiVersion: v1

kind: Pod

metadata:

labels:

test: liveness

name: probe-pod

spec:

containers:

- name: liveness

image: k8s.gcr.io/liveness

args:

- /server

ports:

- containerPort: 8080

readinessProbe:

httpGet:

path: /started

port: 8080

initialDelaySeconds: 5

periodSeconds: 5

livenessProbe:

httpGet:

path: /healthz

port: 8080

initialDelaySeconds: 3

periodSeconds: 3

*##Expose the pod using service named probe-pod##*

kubectl expose pod liveness-http --port=8080 --name=probe-pod

*##Verification##*

controlplane $ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 18m

probe-pod ClusterIP 10.104.162.158 <none> 8080/TCP 2m37s

controlplane $ kubectl get po

NAME READY STATUS RESTARTS AGE

liveness-http 1/1 Running 4 114s

*##When Pod is running##*

controlplane $ curl -v http://10.104.162.158:8080

\* Rebuilt URL to: http://10.104.162.158:8080/

\* Trying 10.104.162.158...

\* TCP\_NODELAY set

\* Connected to 10.104.162.158 (10.104.162.158) port 8080 (#0)

> GET / HTTP/1.1

> Host: 10.104.162.158:8080

> User-Agent: curl/7.58.0

> Accept: \*/\*

>

< HTTP/1.1 404 Not Found

< Content-Type: text/plain; charset=utf-8

< Date: Wed, 11 Nov 2020 16:18:01 GMT

< Content-Length: 19

<

404 page not found

\* Connection #0 to host 10.104.162.158 left intact

controlplane $

*##When Pod is Crashloopback##*

controlplane $ kubectl get po

NAME READY STATUS RESTARTS AGE

liveness-http 0/1 CrashLoopBackOff 3 82s

controlplane $ curl -v http://10.104.162.158:8080

\* Rebuilt URL to: http://10.104.162.158:8080/

\* Trying 10.104.162.158...

\* TCP\_NODELAY set

\* connect to 10.104.162.158 port 8080 failed: Connection refused

\* Failed to connect to 10.104.162.158 port 8080: Connection refused

\* Closing connection 0

curl: (7) Failed to connect to 10.104.162.158 port 8080: Connection refused

<Comment> Answer is ok, this question is 2%, it will take time to attempt this question. Suggest to attempt this question at the end if any time left.

Q7.

Context

You sometimes need to observe a pod’s logs, and write those logs to a file for further analysis.

Task

Please complete the following:

Deploy the counter pod to the cluster using the provided YAML spec file at /opt/KDOB00201/counter.yaml

Retrieve all currently available application logs from the running pod and store them in the file /opt/KDOB00201/log\_output.txt which has already been created

**Answer:**

1 => Deploy the counter pod:

Kubectl apply -f /opt/KDOB00201/counter.yaml

2 => Retrieve the logs:

K logs #pod name# > /opt/KDOB00201/log\_output.txt

<Comment> Answer Ok.

Q8

Context

It is always useful to look at the resources your applications are consuming in a cluster.

Task

From the pods running in namespace stress write the name **only** of the pod that is consuming the most CPU to file /opt/KDOB00201/pod.txt while it has already been created

*Answer*

controlplane $ kubectl top po --sort-by="cpu"

NAME CPU(cores) MEMORY(bytes)

rabbit 972m 1Mi

lion 966m 1Mi

redis 1m 2Mi

ngin 0m 2Mi

controlplane $ echo rabbit > /opt/KDOB00201/pod.txt

controlplane $ cat /opt/KDOB00201/pod.txt

rabbit

<Comment> Answer Ok.

Q9

Context

Anytime a team needs to run a container on Kubernetes they will need to define a pod within which to run the container:

Task

Please complete the following:

Create a YAML formatted pod manifest /opt/KDPD00101/pod1.yaml to create a pod named app1 that runs a container named app1cont using image ifccnf/arg-output with these command line arguments: --lines 56 -F

Create the pod with the kubectl command using the YAML file created in the previous step

When the pod is running display summary data about the pod in JSON format using the kubectl command and redirect the output to a file named //opt/KDPD00101/out1.json

All of the files you need to work which have been created empty for your convenience

*Information: When creating your pod, you do not need to specify a container command, only args.*

*Answer*

kubectl run app --image=lfccncf/arg-output -- --lines 56 -F

kubectl get pod -o=json > /opt/KDPD00101/out1.json

<Comment> Answer incorrect. There are 4 parts to question

1. Create the yaml -/opt/KDPD00101/pod1.yaml
2. Container name should be app1cont, use image specified with arguments
3. Save the pod result in json format.

Sometimes arguments are not working with imperative command, use documentation as reference to add the respective section.

Q10

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

* Run the deployment in the kdpd00201 namespace. The namespace has already created.
* Name the deployment frontend and configure with 6 replicas
* Configure the pod with a container image of ifccnf/nginx:1.13.7
* Set an environment variable of NGINX\_PORT=8001 and also expose that port for the container above

*Answer*

kubectl create ns kdpd00201

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

namespace: kdpd00201

labels:

app: nginx

spec:

replicas: 6

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: lfccncf/nginx:1.13.7

env:

- name: NGINX\_PORT

value: "8001"

ports:

- containerPort: 8001

<Comment> Answer Ok.

Suggession : use imperative command for deployment yaml creation then add the env

Q11

Context

As a Kubernetes application developer you will often find yourself needing to update a running application.

Task

Please complete the following:

* Update the web1 deployment in the kdpd00202 namespace with a maxSurge of 10% and a maxUnavailable of 5%
* Perform a rolling update of the web1 deployment, changing the ifccnf/nginx image version to 1.13.8-alpine
* Roll back the web1 deployment to the previous version

*Answer*

kubectl create ns kdpd00202

kubectl create deployment web1 --image=nginx -n kdpd00202

kubectl get deployments -n kdpd00202

kubectl -n kdpd00202 edit deployments.apps web1 (change maxsurge and maxunavilable)

kubectl -n kdpd00202 set image deployment web1 nginx=lfccncf/nginx:1.13.8-alpine

kubectl get deployments.apps -n kdpd00202

kubectl describe deployments.apps -n kdpd00202 web1

kubectl rollout undo deployment web1 -n kdpd00202

<Comment> Answer Ok.

* Exam will already have the deployment created.
* Some times maxUnavaliable can be number instead of %, edit the deployment as per exam ask.

Q12.

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

* Create a deployment named deployment-web in the default namespace that:
* Includes a primary Ifccncf/busybox:1 container, named logger-dev
* Includes a sidecar Ifccnf/fluentd:v0.12 container, named adaptor-123
* Mounts a shared volume /tmp/log on both containers, which does not persist when the pod is deleted
* Instructs the logger-dev container to run the command

*while true; do*

*echo “I luv cncf” >> /tmp/log/input.log;*

*sleep 10;*

*done*

* The adaptor-123 sidecar container should read /tmp/log/input.log and output the data to /tmp/log/output.\* in Fluentd JSON format. Note that no knowledge of Fluentd is required to complete this task all you will need to achieve this is to create the ConfigMap from the spec file provided at /opt/KDMC00101/fluentd-configmap.yaml and mount that ConfigMap to /fluentd/etc in the adaptor-123 sidecar container

*Answer :*

apiVersion: apps/v1

kind: Deployment

metadata:

creationTimestamp: null

labels:

app: deployment-web

name: deployment-web

spec:

replicas: 1

selector:

matchLabels:

app: deployment-web

strategy: {}

template:

metadata:

creationTimestamp: null

labels:

app: deployment-web

spec:

volumes:

- name: test-volume

emptyDir: {}

- name: config

configMap:

name: fluentd-config

containers:

- image: lfccncf/fluentd:v0.12

name: adaptor-123

env:

- name: FLUENTD\_ARGS

value: -c /etc/fluentd-config/fluent.conf

volumeMounts:

- mountPath: /tmp/log

name: test-volume

- mountPath: /fluentd/etc

name: config

- image: lfccncf/busybox:1

name: logger-dev

volumeMounts:

- mountPath: /tmp/log

name: test-volume

command:

- /bin/sh

- -c

- 'while true; do echo "I luv cncf" > /tmp/log/input.log; sleep 10; done'

resources: {}

status: {}

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

apiVersion: v1

data:

fluent.conf: |

<source>

type tail

format json

path /tmp/log/input.log

tag count.format1

</source>

<match \*\*>

type file

format json

path /tmp/log/output.json

</match>

kind: ConfigMap

metadata:

name: fluentd-config

<Comment> Answer Ok.

Use documentation for easy answer. Attempt this question only if time permits.

Q13

You have been tasked with scaling an existing deployment of availability and creating a service to expose the deployment within your infrastructure.

Task

Start with the deployment named kdns00101-deployment which has already been deployed to the namespace kdsn00101. Edit it to:

* Add the func=dmz key/value label to the pod template metadata to identify the pod for the service definition
* Have 4 replicas

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

* Exposes the service on TCP port 81
* Is mapped to the pods defined by the specification of kdsn00101-deployment
* Is of type NodePort
* Has a name of cherry

*Answer :*

kubectl create ns kdsn00101

kubectl create -n kdsn00101 deployment kdns00101-deployment --image=nginx

kubectl edit -n kdsn00101 deployments.apps kdns00101-deployment

<Change the red highlighted>

*kind: Deployment*

*metadata:*

*labels:*

*app: kdns00101-deployment*

*name: kdns00101-deployment*

*namespace: kdsn00101*

*spec:*

*replicas: 4*

*selector:*

*matchLabels:*

*app: kdns00101-deployment*

*template:*

*metadata:*

*labels:*

*app: kdns00101-deployment*

*func: dmz*

*spec:*

*containers:*

*- image: nginx*

*imagePullPolicy: Always*

*name: nginx*

kubectl get deployments -n kdsn00101

kubectl -n kdsn00101 expose deployment kdns00101-deployment --selector=func=dmz --port=81 --type=NodePort --name=cherry

*Verification*

kubectl -n kdsn00101 describe svc cherry

Name: cherry

Namespace: kdsn00101

Labels: app=kdns00101-deployment

Annotations: <none>

Selector: func=dmz

Type: NodePort

IP: 10.103.2.49

Port: <unset> 81/TCP

TargetPort: 81/TCP

NodePort: <unset> 31135/TCP

Endpoints: 10.244.1.4:81,10.244.1.5:81,10.244.1.8:81 + 1 more...

Session Affinity: None

External Traffic Policy: Cluster

Events: <none>

<Comment> Answer Ok.

Q14

Context

Developers occasionally need to submit pods that run periodically.

Task

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

* Create a YAML formatted Kubernetes manifest /opt/KDPD00301/periodic.yaml that runs the following shell command: date in a single busybox container. The command should run every minute and must complete within 17 seconds or be terminated by Kubernetes. The Cronjob name and container name should both be hello
* Create the resource in the above manifest and verify that the job executes successfully at least once

*Answer*

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: hello

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

activeDeadlineSeconds: 17

template:

spec:

containers:

- name: hello

image: busybox

args:

- /bin/sh

- -c

- date

restartPolicy: OnFailure

kubectl create -f 14.yaml

*Verification:*

controlplane $ kubectl get jobs --watch

NAME COMPLETIONS DURATION AGE

hello-1605783180 1/1 10s 42s

hello-1605783240 0/1 0s

hello-1605783240 0/1 0s 0s

hello-1605783240 1/1 4s 4s

<Comment> Answer Ok.

Q15

Context

A container within the poller pod is hard-coded to connect the nginxsvc service on port 50. As this port changes to 6060 an additional container needs to be added to the poller pod which adapts the container to connect to this new port. This should be realized as an ambassador container within the pod.

Task

* Update the nginxsvc service to serve on port 6060
* Add an HAproxy container named haproxy bound to port 50 to the poller pod and deploy the enhanced pod. Use the image haproxy and inject the configuration located at /opt/KDMC00101/haproxy.cfg with a ConfigMap named haproxy-config, mounted in to the container so that haproxy.cfg is available at /usr/local/ext/haproxy/haproxy.cfg. Ensure that you update the args of the poller container to connect localhost instead of nginxsvc so that the connection is correctly proxied to the new service endpoint. You must **not** modify the port of the endpoint in poller’s args. The spec file used to create the initial poller pod is available in /opt/KDMC00101/poller.yaml

*Answer:*

kubectl edit svc nginxsvc

change port to `6060`

kubectl edit po poller

apiVersion: v1

kind: Pod

metadata:

name: poller-pod

spec:

containers:

- image: nginx

name: nginx

ports:

- containerPort: 6060

- image: haproxy

name: haproxy

ports:

- containerPort: 50

volumeMounts:

- name: config-volume

mountPath: /usr/local/ext/haproxy/haproxy.cfg

volumes:

- name: config-volume

configMap:

name: haproxy-config

<Comment> I skipped question due to time and less weightage (3%only)

Q 16

Task

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

*Answer*

kubectl create deployment web --image=nginxxx

kubectl get deployments.apps

NAME READY UP-TO-DATE AVAILABLE AGE

web 0/1 1 0 7s

<Change image to nginx>

kubectl edit deployments.apps web

kubectl get deployments.apps

NAME READY UP-TO-DATE AVAILABLE AGE

web 1/1 1 1 38s

<Comment> Exam approach is ok

Q17

Task

You have rolled out a new pod to your infrastructure and now you need to allow it to communicate with the web and storage pods but nothing else. Given the running pod kdsn00201-newpod edit it to use a network policy that will allow it to send and receive traffic only to and from the web and storage pods.

Information: All work on this item should be conducted in the kdsn00201 namespace.

All required NetworkPolicy resources are already created and ready for use as appropriate. You should not create, modify or delete any network policies while completing this item.

*Answer*

kubectl create ns kdsn00201

kubectl run web --image=nginx --labels="app=web" -n kdsn00201

kubectl run storage --image=nginx --labels="app=storage" -n kdsn00201

kubectl run kdsn00201-newpod --image=nginx --labels="app=kdsn00201-newpod" -n kdsn00201

vi np.yaml

kubectl create -f np.yaml

*Network policy yaml file :*

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: test-network-policy

namespace: kdsn00201

spec:

podSelector:

matchLabels:

app: kdsn00201-newpod

policyTypes:

- Ingress

- Egress

ingress:

- from:

- podSelector:

matchLabels:

app: web

- podSelector:

matchLabels:

app: storage

egress:

- to:

- podSelector:

matchLabels:

app: web

- podSelector:

matchLabels:

app: storage

*Verification :*

kubectl run test --image=nginx --labels="app=test" -n kdsn00201

kubectl get po -o wide -n kdsn00201

kubectl -n kdsn00201 exec -it kdsn00201-newpod /bin/sh

curl -v <web\_pod\_ip>

curl -v <test\_pod\_ip>

<Comment> Exam will have the policies created, just verify the policy by describing the lables

Then assign the labels appropriately to pod.

Q18

Context

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

Task

Perform the following tasks:

1. Find the broken pod and store its name and namespace to /opt/KDOB00401/broken.txt in the format:

<namespace>/<pod>

The output file has already been created

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

*Information*

*The associated deployment could be running in any of the following namespaces:*

*qa*

*test*

*production*

*alan*

*Answer*

kubectl describe pods liveness-exec | grep -i live

Write name of the 'default/liveness-exec' in /opt/KDOB00401/broken.txt

kubectl get events | grep -i liveness | grep -i warn > /opt/KDOB00401/error.txt

<Comment> Not OK,

Try below approach

1. Search for pods that are not running (failed to run)

k get po --all-namespaces -o wide | grep -iv running

1. Save pod along with name space to file

k get po --all-namespaces -o wide | grep "qa" | grep -iv running | awk '{print $1"/"$2}' > /opt/KDOB00401/error.txt

1. Fix the pod failure (mostly port issue)

Q19.

Context

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

Task

To facilitate this, perform the following tasks:

* Create a file on node sk8s-node-0 at /opt/KDSP00101/data/index.html with the content Acct=Finance
* Create a PersistentVolume named ask-pv-volume using hostPath and allocate 1Gi to it, specifying that the volume is at /opt/KDSP00101/data on the cluster’s node. The configuration should specify the access mode of ReadWriteOnce. It should define the StorageClassName exam for the PersistentVolume, which will be used to bind PersistentVolumeClaim requests to this PersistentVolume.
* Create a PersistentVolumeClaim named task-pv-claim that requests a volume of at least 200Mi 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

You can access sk8s-node-0 by issuing the following command: ssh sk8s-node-0

Ensure that you return to the base node (with hostname node-1) once you have completed your work on sk8s-node-0

*Answer*

apiVersion: v1

kind: PersistentVolume

metadata:

name: ask-pv-volume

labels:

type: local

spec:

storageClassName: exam

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/opt/KDSP00101/data"

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim

spec:

storageClassName: exam

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 200Mi

apiVersion: v1

kind: Pod

metadata:

name: task-pv-pod

app: my-storage-app

spec:

volumes:

- name: task-pv-storage

persistentVolumeClaim:

claimName: task-pv-claim

containers:

- name: task-pv-container

image: nginx

ports:

- containerPort: 80

volumeMounts:

- mountPath: "/usr/share/nginx/html"

name: task-pv-storage

<Comment> OK,

During exam, validate every step before moving to other. Make sure to practice this question multiple times as its weightage is 8%.