1] Create a new ClusterRole named deployment-clusterrole,which only allows to create the following resource types:

Deployment

StatefulSet

DeamonSet Create a new ServiceAccount named cicd-token in the existing namespace app-team1. Bind the new ClusterRole deployment-clusterrole to the new ServiceAccount cicd-token, limit to the namespace app-team1

Answer:

kubectl create clusterrole deployment-clusterrole --verb=create --resource=deployments,statefulsets,daemonsets

kubectl create serviceaccount cicd-token --namespace=app-team1

kubectl create rolebinding deployment-clusterrole --clusterrole=deployment-clusterrole --serviceaccount=default:cicd-token --namespace=app-team1

2] Set the node labelled with name=ek8s-node-1 as unavailable and reschedule all the pods running on it.

Answer:

kubectl cordon ek8s-node-1

kubectl drain ek8s-node-1 --delete-local-data --ignore-daemonsets --force

3]Given an existing Kubernetes cluster running version 1.18.8, upgrade all of the Kubernetes control plan and node components on the master node only to version 1.19.0. You are also expected to upgrade kubelet and kubectl on the master node.

Answer:

kubectl cordon k8s-master

kubectl drain k8s-master --delete-local-data --ignore-daemonsets --force

apt-get install kubeadm=1.19.0-00 kubelet=1.19.0-00 kubectl=1.19.0-00 --disableexcludes=kubernetes

kubeadm upgrade apply 1.19.0 --etcd-upgrade=false

systemctl daemon-reload

systemctl restart kubelet

kubectl uncordon k8s-master

4]Create a snapshot of the existing etcd instance running at https://127.0.0.1:2379 saving the snapshot to /srv/data/etcd-snapshot.db

Next, restore an existing, previous snameshot located at /var/lib/backup/etcd-snapshot-previous.db.

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

CA certificate: /opt/KUIN00601/ca.crt Client certificate: /opt/KUIN00601/etcd-client.crt Clientkey:/opt/KUIN00601/etcd-client.key

Answer:

#backup

ETCDCTL\_API=3 etcdctl --endpoints="https://127.0.0.1:2379" --cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --key=/opt/KUIN000601/etcd-client.key snapshot save /etc/data/etcd-snapshot.db

#restore

ETCDCTL\_API=3 etcdctl --endpoints="https://127.0.0.1:2379" --cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --key=/opt/KUIN000601/etcd-client.key snapshot restore /var/lib/backup/etcd-snapshot-previoys.db

5] Create a new NetworkPolicy name allow-port-from-namespace that allows Pods in the existing namespace internal to connect to port 9000 of other Pods in the same namespace, Ensure that the new NetworkPolicy:

does not allow access to Pods not listening on port 9000

does not allow access from Pods not in namespace internal

Answer:

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: all-port-from-namespace

namespace: internal

spec:

podSelector:

matchLabels: {}

ingress:

- from:

- namespaceSelector:

matchLabels:

name: namespacecorp-net

- podSelector: {}

ports:

- port: 9000

6]Reconfigure the existing deployment front-end and add a port specification named http exposing port 80/tcp of the existing container nginx

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

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

Answer:

kubectl expose deployment front-end --name=front-end-svc --port=80 --targetport=80 --type=NodePort

7] Create a new nginx Ingress resource as follows:

Name: pong

Namespace: ing-internal

Exposing service hi on path /hi using service port 5678 The availability of service hi can be checked using the following command, which should return hi:

Answer: curl -kL <INTERNAL\_IP>/hi

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: ping

namespace: ing-internal

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- http:

paths:

- path: /hello

pathType: Prefix

backend:

service:

name: hello

port:

number: 5678

8] Scale the deployment loadbalancer to 6 pods.

Answer:

kubectl scale deploy loadbalancer --replicas=6

9] Schedule a pod as follow:

Name: nginx-kusc00401

Image: nginx

Node selector: disk=spinning

Answer: kubectl run nginx –image=nginx –dry-run=client -oyaml edit

apiVersion: v1

kind: Pod

metadata:

name: nginx-kusc00401

labels:

role: nginx-kusc00401

spec:

nodeSelector:

disk: spinning

containers:

- name: nginx

image: nginx

10] Check to see how many nodes are ready (not including nodes tainted NoSchedule) and write the number to /opt/nodenum

Answer:

kubectl get node | grep -i ready

kubectl describe nodes <nodeName> | grep -i taints | grep -i noSchedule

相减，写入/opt/nodenum

11] Create a pod named kucc1 with a single container for each of the following images running inside (there may be between 1 and 4 images specified): nginx + redis + memcached + consul

Answer: kubectl run kucc1 –image=nginx –dry-run=client -oyaml edit kubectl apply

apiVersion: v1

kind: Pod

metadata:

name: kucc1

spec:

containers:

- image: nginx

name: nginx

- image: redis

name: redis

- image: memchached

name: memcached

- image: consul

name: consul

12] Creae a persistent volume with name app-config of capacity 1Gi and access mode ReadWriteOnce. The type of volume is hostPath and its location is /srv/app-config

Answer: refer to: https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/#create-a-persistentvolume

apiVersion: v1

kind: PersistentVolume

metadata:

name: app-config

labels:

type: local

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/srv/app-config"

13] Create a new PVC:

Name: pv-volume

Class: csi-hostpath-sc

Capacity: 10Mi Create a new Pod which mounts the PVC as a volume:

Name: web-server

Image: nginx

Mount path: /usr/share/nginx/html Configure the new Pod to have ReadWriteOnce access on the volume. Finally, using kubectl edit or kubectl path expand the PVC to a capacity 70Mi and record that change.

Answer:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pv-volume

spec:

accessModes:

- ReadWriteOnce

volumeMode: Filesystem

resources:

requests:

storage: 10Mi

storageClassName: csi-hostpath-sc

apiVersion: v1

kind: Pod

metadata:

name: web-server

spec:

containers:

- name: nginx

image: nginx

volumeMounts:

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

name: pv-volume

volumes:

- name: pv-volume

persistentVolumeClaim:

claimName: pv-volume

kubectl edit pvc pv-volume –save-config

14] Monitor the logs of pod foobar and:

Extract log lines corrsponding to error unable-to-access-website

Write them to /opt/KUTR00101/foobar

Answer:

kubectl logs foobar |grep unable-to-access-website > /opt/KUTR00101/foobar

cat /opt/KUTR00101/foobar

15] Without changing its existing containers, an existing Pod needs to be integrated into Kubernetes’s built-in logging architecture(e.g. kubectl logs). Adding a streaming sidecar container is a good and common way to accomplish this requirement.

Add a busybox sidecar container to the existing Pod legacy-app. The new sidecar container has to run the following command: /bin/sh -c tail -n+1 -f /var/log/legac-appp.log

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

Don’t modify the existing container.

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

Answer: kubectl get pod xxx -oyaml

apiVersion: v1

kind: Pod

metadata:

name: podname

spec:

containers:

- name: count

image: busybox

args:

- /bin/sh

- -c

- >

i=0;

while true;

do

echo "$(date) INFO $i" >> /var/log/legacy-ap.log;

i=$((i+1));

sleep 1;

done

volumeMounts:

- name: logs

mountPath: /var/log

- name: count-log-1

image: busybox

args: [/bin/sh, -c, 'tail -n+1 -f /var/log/legacy-ap.log']

volumeMounts:

- name: logs

mountPath: /var/log

volumes:

- name: logs

emptyDir: {}

#验证： kubectl logs <pod\_name> -c <container\_name>

16] From the pod label name=cpu-user,find pods running high CPU workloads and write the name of the pod consuming most CPU to the fule /opt/KUT00401/KUT00401.txt (which already exists).

Answer:

kubectl top -l name=cpu-user -A

echo 'pod name' >> /opt/KUT00401/KUT00401.txt

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

Answer:

sudo -i

systemctl status kubelet

systemctl start kubelet

systemctl enable kubelet

18]There seems to be an issue with the kubelet not running on cluster3-node1. Fix it and confirm that cluster has node cluster3-node1 available in Ready state afterwards. You should be able to schedule a Pod on cluster3-node1 afterwards.

ans:

ssh cluster3-node1

service kubelet status

service kubelet start

service kubelet status

/usr/local/bin/kubelet

whereis kubelet

vim /etc/systemd/system/kubelet.service.d/10-kubeadm.conf # fix

systemctl daemon-reload

systemctl restart kubelet

19] Use a JSON PATH query to identify the context configured for the aws-user in the my-kube-config context file and store the result in /opt/outputs/aws-context-name.

Ans:

kubectl config view --kubeconfig=my-kube-config -o jsonpath="{.contexts[?(@.context.user=='aws-user')].name}" > /opt/outputs/aws-context-name

1] Control ServiceAccount permissions using RBAC

There are existing Namespaces ns1 and ns2 .

Create ServiceAccount pipeline in both Namespaces.

These SAs should be allowed to view almost everything in the whole cluster. You can use the default ClusterRole view for this.

These SAs should be allowed to create and delete Deployments in Namespaces ns1 and ns2 .

Verify everything using kubectl auth can-i

=

- k create serviceaccont pipeline -n ns1 -n ns2

- k create clusterrolebinding pipeline-view --clusterrole view --serviceaccount ns1:pipeline --serviceaccount ns2:pipeline

- k create clusterrole pipeline-deployment-manager --verb create,delete --resource deployments

- k -n ns1 create rolebinding pipeline-deployment-manager --clusterrole pipeline-deployment-manager -- serviceaccount ns1:pipeline

- k -n ns2 create rolebinding pipeline-deployment-manager --clusterrole pipeline-deployment-manager --serviceaccount ns2:pipeline

- k auth can-i delete deployments --as system:serviceaccount:ns1:pipeline -n ns1 # YES

2] Control User permissions using RBAC

There is existing Namespace applications .

User smoke should be allowed to create and delete Pods, Deployments and StatefulSets in Namespace applications.

User smoke should have view permissions (like the permissions of the default ClusterRole named view ) in all Namespaces but not in kube-system .

Verify everything using kubectl auth can-i .

=

- k create role role-user --verb=create,delete --resource=Deployments,StatefulSets,Pods -n applications

- k create rolebinding role-user-binding --role=role-user --user=smoke -n applications

- k get ns

- k create rolebinding application-view --clusterrole=view --user=smoke -n applications

- k create rolebinding application-view --clusterrole=view --user=smoke -n kube-public

- k create rolebinding application-view --clusterrole=view --user=smoke -n kube-nodes-lease

- k create rolebinding application-view --clusterrole=view --user=smoke -n default

3] Scheduling Priority:

Delete highest priority pod:

>Find the Pod with the highest priority in Namespace management and delete it.

=

k describe pod -n management (delete the pod which have a high prioruty number in priority: )

k delete pod xxx -n management

Create Pod with higher priority:

>In Namespace lion there is one existing Pod which requests 1Gi of memory resources.

That Pod has a specific priority because of its PriorityClass.

Create new Pod named important of image nginx:1.21.6-alpine in the same Namespace. It should request 1Gi memory resources.

Assign a higher priority to the new Pod so it's scheduled instead of the existing one.

Both Pods won't fit in the cluster.

=

k run important --image=nginx:1.21.6-alpine -n lion --dry-run=client -o yaml > imp.yaml

kubectl create priorityclass high-priority --value=300000000

vi imp.yaml

containers:

resources:

requests:

memory: 1Gi

priorityClassName: high-priority

4] NetworkPolicy Namespace Selector:

There are existing Pods in Namespace space1 and space2 .

We need a new NetworkPolicy named np that restricts all Pods in Namespace space1 to only have outgoing traffic to Pods in Namespace space2 . Incoming traffic not affected.

We also need a new NetworkPolicy named np that restricts all Pods in Namespace space2 to only have incoming traffic from Pods in Namespace space1 . Outgoing traffic not affected.

The NetworkPolicies should still allow outgoing DNS traffic on port 53 TCP and UDP.

=

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: np

namespace: space1

spec:

podSelector: {}

policyTypes:

- Egress

egress:

- to:

- namespaceSelector:

matchLabels:

kubernetes.io/metadata.name: space2

- ports:

- protocol: TCP

port: 53

- protocol: UDP

port: 53

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

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: np

namespace: space2

spec:

podSelector: {}

policyTypes:

- Ingress

ingress:

- from:

- namespaceSelector:

matchLabels:

kubernetes.io/metadata.name: space1

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

5] Ingress Create:

Create Services for existing Deployments:

There are two existing Deployments in Namespace world which should be made accessible via an Ingress.

First: create ClusterIP Services for both Deployments for port 80 . The Services should have the same name as the Deployments.

=

k get deploy -n world

k expose deployment asia -n world --port=80

k expose deployment europe -n world --port=80

Create Ingress for existing Services:

The Nginx Ingress Controller has been installed.

Create a new Ingress resource called world for domain name world.universe.mine . The domain points to the K8s Node IP via /etc/hosts .

The Ingress resource should have two routes pointing to the existing Services:

http://world.universe.mine:30080/europe/

and

http://world.universe.mine:30080/asia/

=

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: world

namespace: world

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

ingressClassName: nginx

rules:

- host: "world.universe.mine"

http:

paths:

- path: /europe

pathType: Prefix

backend:

service:

name: europe

port:

number: 80

- path: /asia

pathType: Prefix

backend:

service:

name: asia

port:

number: 80

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

=

- kubectl create serviceaccount pvviewer

- k create clusterrole pvviewer-role --verb=list --resource=PersistentVolumes

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

7] create pod set

securityContext:

capabilities:

add: ["NET\_ADMIN", "SYS\_TIME"]

then set system time by executing the command 'sudo date +%T -set "16:00:00"

and store output in file /opt/dile.txt

=

kubectl exec -it pod-add1 -- date +%T --set "16:00:00" > /opt/data/vishal.txt

k run front-end-helper --image=nginx -it --rm --restart=Never -- /bin/sh -c 'echo binary de' > front.txt

k get node -o json | jq ".items[] | {name:.metadata.name, taint:.spec.taints}"

8] Create a new ClusterRole named demo-clusterrole in demo-namespace. Any resource associated with the cluster role should be able to create the following resources:

Deployment

StatefulSet

DaemonSet

Create a ServiceAccount named demo-token and bind the ClusterRole with the ServiceAccount.

=

kubectl create ns demo-namespace

kubectl create serviceaccount demo-token -n demo-namespace

kubectl create clusterrole demo-clusterrole –resource=deployments,statefulsets,daemonsets –verb=create

kubectl create clusterrolebinding demo-clusterrole-binding –clusterrole=demo-clusterrole serviceaccount=demo-namespace:demo-token

9] Create three pods, pod name and their image name is given below:

Pod Name Image Name & commands

nginx nginx

busybox1 busybox1, sleep 3600

busybox2 busybox2, sleep 1800

Make sure only busybox1 pod should be able to communicate with nginx pod on port 80. Pod busybox2 should not be able to connect to pod Nginx.

=

kubectl run nginx –image=nginx

kubectl run busybox1 –image=busybox — sleep 3600

kubectl run busybox2 –image=busybox — sleep 1800

vi network.yaml

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: networkpolicy

namespace: default

spec:

podSelector:

matchLabels:

run: nginx

policyTypes:

– Ingress

ingress:

– from:

– podSelector:

matchLabels:

run: busybox1

ports:

– protocol: TCP

port: 80

10] Create a pod “cka-pod” using image “nginx”, with a hard limit of 0.5 CPU and 20 Mi memory in “cka-exam” namespace.

=

kubectl create ns cka-exam

vi pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: cka-pod

namespace: cka-exam

spec:

containers:

– name: cka-pod

image: nginx

resources:

limits:

cpu: “0.5”

memory: 20Mi

11] Create a pod by name “readonly-pod” using image “alpine”. The process running inside pod should only have ReadOnly access on container’s filesystem. Create a volume by the name “my-volume” and mount it at /data ( inside the container). The process running inside the container should have read & write access on the mounted volume. Also, run “sleep 3600” inside the container.

Image name: alpine

Pod name: readonly-pod

MountPath inside container: /data

command: sleep 3600

Access on :

Container filesystem: Read only

montedVolume: Read & Write

=

apiVersion: v1

kind: Pod

metadata:

labels:

run: readonly-pod

name: readonly-pod

spec:

containers:

– image: alpine

name: readonly-pod

command: [‘sh’, ‘-c’, ‘sleep 3600’]

securityContext:

readOnlyRootFilesystem: true

volumeMounts:

– name: my-volume

mountPath: /data

readOnly: false

volumes:

– name: my-volume

emptyDir: {}

kubectl exec -it readonly-pod — touch /tmp/test.txt

12] Create a pod by name “pod-add-settime-capability”, using image “alpine”. Add capability called “SYS\_TIME” and “NET\_ADMIN” to the container and command “sleep 3600”. Set system time by executing the command “sudo date +%T –set “16:00:00”.

Once pod is up and runnimg, check system time and store the output under /tmp/cka/pod-add-settime-capability.txt

=

apiVersion: v1

kind: Pod

metadata:

name: pod-add-settime-capability

spec:

containers:

– name: pod-add-settime-capability

image: alpine

command: [‘sh’, ‘-c’, ‘sleep 3600’]

securityContext:

capabilities:

add: [“SYS\_TIME”, “NET\_ADMIN”]

kubectl exec -it pod-add-settime-capability — date +%T –set ’16:00:00′

echo `kubectl exec -it pod-add-settime-capability — date` > /tmp/cka/pod-add-settime-capability.txt

13] Create a multi-pod container by name “multi-container-pod” with below-mentioned details:

First container

name: main

Image name: busybox:1.28

args: – /bin/sh – -c – > i=0; while true; do echo “$i: $(date)” >> /var/log/cka-exam.log; i=$((i+1));

sleep 1; done

volumeMount: /var/log

volumeName: main-volume

Second container

name: sidecar

Image name: busybox:1.28

args: [/bin/sh, -c, ‘tail -n+1 -F /var/log/cka-exam.log’]

volumeMount: /var/log

volumeName: main-volume

check the sidecar container’s log and store it’s output under /tmp/output.txt

=

apiVersion: v1

kind: Pod

metadata:

name: multi-container-pod

spec:

containers:

– name: main

image: busybox:1.28

args:

– /bin/sh

– -c

– >

i=0;

while true;

do

echo “$i: $(date)” >> /var/log/cka-exam.log;

i=$((i+1));

sleep 1;

done

volumeMounts:

– name: main-volume

mountPath: /var/log

– name: sidecar

image: busybox:1.28

args: [/bin/sh, -c, ‘tail -n+1 -F /var/log/cka-exam.log’]

volumeMounts:

– name: main-volume

mountPath: /var/log

volumes:

– name: main-volume

emptyDir: {}

14] You have got a project which has three running pods by below given names:

Pod Name Image Namespace

WebServer devopstitan/cka-exam:webserver-p80 web

appserver devopstitan/cka-exam:appserver-p8080 app

dbserver devopstitan/cka-exam:dbserver-p6379 db

Create a network policy by name “cka-networkpolicy” to allow only one way connection from Webserver pod to appserver and DB server . The appserver is running on the TCP port 8080 and dbserver is running on the TCP port 6379

=

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: webserver-networkpolicy

namespace: web

spec:

podSelector:

matchLabels:

run: webserver

policyTypes:

– Egress

– Ingress

egress:

– to:

– podSelector:

matchLabels:

run: dbserver

– namespaceSelector:

matchLabels:

kubernetes.io/metadata.name: db

ports:

– protocol: TCP

port: 6379

– to:

– podSelector:

matchLabels:

run: appserver

– namespaceSelector:

matchLabels:

kubernetes.io/metadata.name: app

ports:

– protocol: TCP

port: 8080

15] Switch to context “kubernetes-admin@kubernetes” and Create a deployment by name cka-deployment, as per the below requirements:

Replicas: 4

Image : nginx:1.14

namespace: cka-exam

Implement strategy for rolling update with maxUnavailable: 50%

Update deployment :

Image: nginx:1.15

Update Deployment once again:

Image: nginx:1.99

Check the rollout status & rollout history.

After 2nd rollout, if pods are not up then roll back the deployment to its initial version ( image=nginx:1.14)

=

kubectl config use-context kubernetes-admin@kubernetes

kubectl create ns cka-exam

apiVersion: apps/v1

kind: Deployment

metadata:

name: cka-deployment

namespace: cka-exam

labels:

app: nginx

spec:

replicas: 4

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 50%

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

– name: nginx

image: nginx:1.14

ports:

– containerPort: 80

kubectl create -f cka-deployment.yaml

kubectl set image deployment cka-deployment nginx=nginx:1.15 -n cka-exam

kubectl rollout status deployment cka-deployment -n cka-exam

kubectl set image deployment.v1.apps cka-deployment nginx=nginx:1.99 -n cka-exam

kubectl rollout status deployment cka-deployment -n cka-exam

kubectget deployment -n cka-exam

kubectrollout status deployment cka-a-deployment -n cka-exam

kubectl rollout history deployment cka-deployment -n cka-exam

kubectl rollout undo deployment cka-deployment -n cka-exam –to-revision=1

kubectl get deployment -n cka-exam -o wide

16] to check certicate details,name ,issure name

-

cat /etc/kunernets/manifest/kube-api-server.yaml

openssl x509 -in <file\_path>.crt -text -noout

17]Create a secret object with the credentials required to access the registry.

Name: private-reg-cred

Username: dock\_user

Password: dock\_password

Server: myprivateregistry.com:5000

Email: dock\_user@myprivateregistry.com

=

kubectl create secret docker-registry private-reg-cred --docker-username=dock\_user --docker-password=dock\_password --docker-server=myprivateregistry.com:5000 [--docker-email=dock\_user@myprivateregistry.com](mailto:--docker-email=dock_user@myprivateregistry.com)

18] Take a backup of the etcd cluster and save it to /opt/etcd-backup.db.

= export ETCDCTL\_API=3

etcdctl snapshot save --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 \

/opt/etcd-backup.db

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

=kubectl run redis-storage --image=redis:alpine --dry-run=client -oyaml > redis-storage.yaml

apiVersion: v1

kind: Pod

metadata:

creationTimestamp: null

labels:

run: redis-storage

name: redis-storage

spec:

containers:

- image: redis:alpine

name: redis-storage

volumeMounts:

- mountPath: /data/redis

name: temp-volume

volumes:

- name: temp-volume

emptyDir: {}

20] Create a new pod called super-user-pod with image busybox:1.28. Allow the pod to be able to set system\_time.The container should sleep for 4800 seconds

= creqate pod

conatiners:

securityContext:

capabilities:

add: ["SYS\_TIME"]

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

mountPath: /data

persistentVolumeClaim Name: my-pvc

=- Add a persistentVolumeClaim

- create pod

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

Deployment : nginx-deploy. Image: nginx:1.16

Image: nginx:1.16

Task: Upgrade the version of the deployment to 1:17

Task: Record the changes for the image upgrade

= kubectl create deployment nginx-deploy --image=nginx:1.16 --dry-run=client -o yaml > deploy.yaml

kubectl apply -f deploy.yaml --record

kubectl rollout history deployment nginx-deploy

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

kubectl rollout history deployment nginx-deploy

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

Important Note: As of kubernetes 1.19, the CertificateSigningRequest object expects a signerName

= use documnetattion

---

apiVersion: certificates.k8s.io/v1

kind: CertificateSigningRequest

metadata:

name: john-developer

spec:

signerName: kubernetes.io/kube-apiserver-client

request: <find that>

usages:

- digital signature

- key encipherment

- client auth

- kubectl certificate approve john-developer

kubectl create role developer --resource=pods --verb=create,list,get,update,delete --namespace=development

kubectl create rolebinding developer-role-binding --role=developer --user=john --namespace=development

kubectl auth can-i update pods --as=john --namespace=development

24]Create a nginx pod called nginx-resolver using image nginx, expose it internally with a service called nginx-resolver-service. Test that you are able to look up the service and pod names from within the cluster. Use the image: busybox:1.28 for dns lookup. Record results in /root/CKA/nginx.svc and /root/CKA/nginx.pod

= kubectl run nginx-resolver --image=nginx

kubectl expose pod nginx-resolver --name=nginx-resolver-service --port=80 --target-port=80 --type=ClusterIP

kubectl run test-nslookup --image=busybox:1.28 --rm -it --restart=Never -- nslookup nginx-resolver-service > /root/CKA/nginx.svc

kubectl get pod nginx-resolver -o wide

kubectl run test-nslookup --image=busybox:1.28 --rm -it --restart=Never -- nslookup <P-O-D-I-P.default.pod> > /root/CKA/nginx.pod

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

Use /etc/kubernetes/manifests as the Static Pod path for example.

=kubectl run nginx-critical --image=nginx --dry-run=client -o yaml > static.yaml

root@controlplane:~# scp static.yaml node01:/root/

root@controlplane:~# kubectl get nodes -o wide

root@controlplane:~# ssh node01

root@node01:~# mkdir -p /etc/kubernetes/manifests

root@node01:~# vi /var/lib/kubelet/config.yaml

root@node01:~# cp /root/static.yaml /etc/kubernetes/manifests/

root@node01:~# exit

logout

root@controlplane:~# kubectl get pods

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

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

=

27] List the InternalIP of all nodes of the cluster. Save the result to a file /root/CKA/node\_ips.

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

=kubectl get nodes -o jsonpath='{.items[\*].status.addresses[?(@.type=="InternalIP")].address}'> /root/CKA/node\_ips

28] Create a pod called multi-pod with two containers.

Container 1, name: alpha, image: nginx

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

Environment Variables:

container 1:

name: alpha

Container 2:

name: beta

= apiVersion: v1

kind: Pod

metadata:

name: multi-pod

spec:

containers:

- image: nginx

name: alpha

env:

- name: name

value: alpha

- image: busybox

name: beta

command: ["sleep", "4800"]

env:

- name: name

value: beta

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

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

Important: Don't delete any current objects deployed

=apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: ingress-to-nptest

namespace: default

spec:

podSelector:

matchLabels:

run: np-test-1

policyTypes:

- Ingress

ingress:

- ports:

- protocol: TCP

port: 80

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

key: env\_type, value: production, operator: Equal and effect: NoSchedule

=kubectl taint node node01 env\_type=production:NoSchedule

kubectl run dev-redis --image=redis:alpine

kubectl get pods -o wide

-- Solution manifest file to deploy new pod called prod-redis with toleration to be scheduled on node01 worker node.

apiVersion: v1

kind: Pod

metadata:

name: prod-redis

spec:

containers:

- name: prod-redis

image: redis:alpine

tolerations:

- effect: NoSchedule

key: env\_type

operator: Equal

value: production

kubectl get pods -o wide | grep prod-redis

31] Create a pod called hr-pod in hr namespace belonging to the production environment and frontend tier .

image: redis:alpine

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

=

kubectl create namespace hr

kubectl run hr-pod --image=redis:alpine --namespace=hr --labels=environment=production,tier=frontend

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

=Verify host and port for kube-apiserver are correct.

Open the super.kubeconfig in vi editor.

Change the 9999 port to 6443 and run the below command to verify:

kubectl cluster-info --kubeconfig=/root/CKA/super.kubeconfig

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

=kubectl scale deploy nginx-deploy --replicas=3

kubectl get pods -n kube-system

sed -i 's/kube-contro1ler-manager/kube-controller-manager/g' /etc/kubernetes/manifests/kube-controller-manager.yaml

kubectl get deploy

34]Create a service messaging-service to expose the messaging application within the cluster on port 6379.

Use imperative commands.

Service: messaging-service

Port: 6379

Type: ClusterIp

Use the right labels

ans:- kubectl create deploy messaging-service --image=<massaging pod image>

- kubectl expose deployment messaging-service --type ClusterIP --port 6379 --labels=<massaging pod labels>

- kubectl expose pod messaging --port=6379 --name messaging-service

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

ans:- kubectl get nodes -o json > /opt/outputs/nodes-z3444kd9.json

36]Use JSON PATH query to retrieve the osImages of all the nodes and store it in a file /opt/outputs/nodes\_os\_x43kj56.txt.

The osImages are under the nodeInfo section under status of each node.

ans: - kubectl get nodes -o jsonpath='{.items[\*].status.nodeInfo.osImage}' > /opt/outputs/nodes\_os\_x43kj56.txt

37] Allow the pod to be able to set system\_time

ans: spec:

containers:

securityContext:

capabilities:

add: ["SYS\_TIME"]