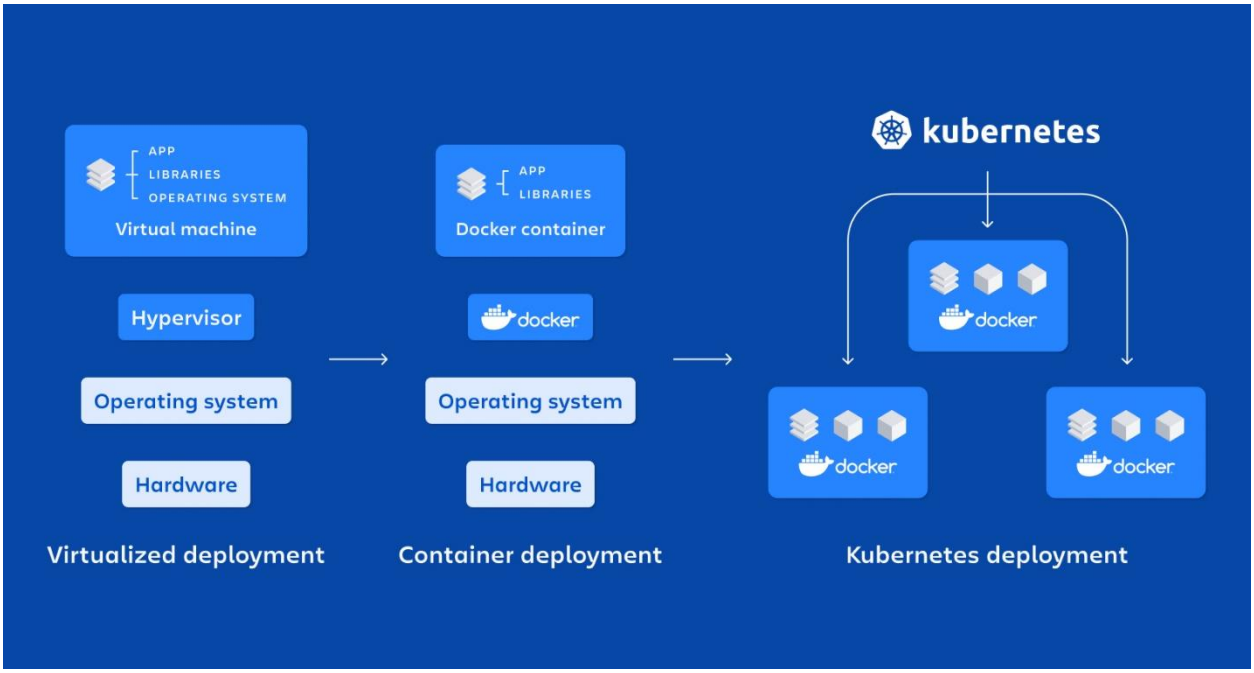
Docker – Docker is a container runtime

Kubernetes – Kubernetes is a platform for running and managing containers from many container runtimes. Kubernetes supports numerous container runtimes including Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).



Helpful tools to use with Kubernetes

Kubectl – The Kubernetes command line tool which allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

Minkube - minikube runs a single-node Kubernetes cluster on your personal computer (including Windows, macOS and Linux PCs) so that you can try out Kubernetes, or for daily development work. Another such tool is kind, kind needs docker installed and configured.

Kubeadm – it is used to create and manage Kubernetes clusters.

Helm – Helm is a package manager for Kubernetes. It helps define, install and upgrade applications running on Kubernetes. Helm and Kubernetes work like a client/server application. The helm client pushes resources to the Kubernetes cluster.

Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on.

CKAD Curriculum

20% - Application Design and Build

- Define, build and modify container images
- Understand Jobs and CronJobs
- Understand multi-container Pod design patterns (e.g. sidecar, init and others)
- Utilize persistent and ephemeral volumes

20% - Application Deployment

- Use Kubernetes primitives to implement common deployment strategies (e.g. blue/ green or canary)
- Understand Deployments and how to perform rolling updates
- Use the Helm package manager to deploy existing packages

15% - Application observability and maintenance

- Understand API deprecations
- Implement probes and health checks
- Use provided tools to monitor Kubernetes applications
- Utilize container logs
- Debugging in Kubernetes

25% - Application Environment, Configuration and Security

- Discover and use resources that extend Kubernetes (CRD)
- Understand authentication, authorization and admission control
- Understanding and defining resource requirements, limits and quotas
- Understand ConfigMaps • Create & consume Secrets
- Understand ServiceAccounts
- Understand SecurityContexts

20% - Services & Networking

- Demonstrate basic understanding of NetworkPolicies
- Provide and troubleshoot access to applications via services
- Use Ingress rules to expose applications

Know your key topics well

- **Core Concepts**: *Pod, Replica Set, Deployment, Namespaces*
- **Configuration**: *Environment Variables, Command Arguments, Config map, Secrets, Volumes, Security Contexts, Node Affinity, Taint and tolerations, Service Account*
- **Multi-Container Pods**: *Sidecar, Ambassador, Adapter*
- **Observability**: *Liveness Probe, Readiness Probe, Logging, Monitoring*
- **Pod Design**: *Labels, Selectors, Job, Cron Job, Deployment Strategy*
- **Services & Networking**: *Network Policies, Services, Ingress*
- **State Persistence**: *Persistent Volume, PersistentVolumeClaim, Volume Mount, Storage Class, Stateful Sets*

Arsenal

- **Imperative commands**
- **VIM**
- **Documentation Kubernetes.io/docs, Heml.sh/docs, Github.com/Kubernetes**
- **Kubectl explain** ex: kubectl explain pod.spec.containers

Plan of action

- ✓ Complete Benjamin Muschko CKAD Guide with tests
- ✓ https://github.com/dgkanatsios/CKAD-exercises
- ✓ Killercoda scenarios
- ✓ 2 lightening labs on kodecloud
- ✓ 2 Mock exams on Kodecloud
- ✓ Killercoda mockexam

Docker Hub

Username: keywestrooster

Password: Docker@1982

PSI Customer Service

1800-367-1565 X 7040

Git Hub

Username: triplecrownct

Password: Github@1982

1 day before exam

- ✓ Review answers to chapter questions before exam from the book Benjamin Muschko CKAD Guide with tests
  Study guide:
  https://learning.oreilly.com/certifications/guides/Certified-Kubernetes-Application-Developer-(CKAD)/0636920820758/

  Answers: https://learning.oreilly.com/library/view/certified-kubernetes-application/9781492083726/app01.html

Exam Must do #1

Set the context and namespace before answering each question to set the Kubernetes cluster and namespace

Kubectl config use-context cluster-name - - namespace namespace-1

And then set alias for kubectl

Alias k=kubectl

Export do=”-o yaml - -dry-run=client” ex: k run nginx –image=nginx $do > nginx.yaml

Export now=”- -grace-period=0 - -force” ex: k delete pod pod1 $now

Exam Must do #2

Open Kubernetes cheat sheet and Kubernetes API in the browser

Exam Must do #3

Internalize shortcuts - Kubectl api-resources

Persistent Volume Claim = pvc, Config maps = cm, Endpoints = ep, Namespaces = ns, Persistence volumes = pv, Pods = po, Replicationcontrollers = rc, Resource quotas = quota, Service accounts = sa, Services = svc, Stateful sets = sts , Networkpolicies = netpol, Storage classes = sc

Exam Must Do #4

In case of mistakes and I want to delete objects and re-answer a question,

Execute a kill command

Kubectl delete pod ngnix $now

Exam Must Do #5

Combine kubectl commands with other commands using pipe and grep -C

Kubectl describe pods –namespace=kube-system | grep -C 10 labels:

Exam Must Do #6

Use **kubectl create - -help** or use **kubectl explain pods**

Exam Must Do #7

Use imperative commands

Use kubectl run or kubectl create to create objects on the fly. You can create a manifest file and then edit it.

$ **Kubectl run frontend - -image=ngnix - - restart=never - -port=80 $do > pod.yaml**

$ vim pod.yaml

$ kubectl create -f pod.yaml

$ kubectl describe pod front-end

Exam Must Do #8

Use apply for creating/editing resources

Kubectl apply -f pod.yaml

Exam Point #9

Executing a command inside a container

K exec -it wordpress - - ls -l

For connecting to a certain IP

wget --spider --timeout=1 10.244.1.2

wget -O- www.google.com

Exam Point #10

After running commands to create a resource, always check if it's created or not before running further commands.

Use, **k get pod pod-name -o wide** to get IP and node name along with status

Exam Point#11

When a service is created, labels cannot be passed imperatively. They need to be set in the yaml.

Also, ports are mentioned using format - - tcp = 9000:8080

Solution- Create a deployment first, add labels and then expose

it as a service.

Exam Point#12

On the exam use Ctrl+shift+C to copy and Control+Shift+V to paste

Get Containers in a pod

kubectl get pods POD_NAME_HERE -o jsonpath='{.spec.containers[*].name}'

crictl ps

instead of deleting and recreating a pod,

k replace -f pod.yaml - - force

Config Maps

Config maps are key value pairs use to externalize environment variables from application. They can be passed in 4 ways.

1) Values themselves
2) In an environment file
3) In a file
4) In a folder

Kubectl create configmap db-config - -from-literal=db=staging

Kubectl create configmap db-config - -from-env-file=config.env

Kubectl create configmap db-config - -from-file=config.txt

Kubectl create configmap db-config - -from-file=directory-name

K create configmap config –from-literal=foo=lala –from-literal=foo=lolo

Consuming a config map as environment variable

Reference a config map named backend-config in a pod and inject key value pairs as environment variables. It is done under image in spec using envFrom.configMapRef = backend-config

Or change the values using **env.valueFrom.configMapKeyRef**

To check the configured values at run time inside the container,

Kubectl exec pod-name - - env

Mounting configmap as a volume

Spec

Containers:

-image: ngnix:1.17.0

Name: app

volumeMounts:

-name: config-volume

mountPath: /etc/config

Volumes:

-name: configVolume

configMap:

name: backend-config

Creating a secret

Same as configMap it can be created in 4 ways. An additional sub-command needs to be passed after secret – generic, tls or docker-registry. Secret needs to be base64 encoded if yaml file is created declaratively.

Kubectl create secret generic db-creds - -from-literal=password=!@spectre

Kubectl create secret generic db-creds - -from-env-file=db=secret.env

Kubectl create secret generic db-creds ssh-key - -from-file=id_rsa=~/.ssh/id_rsa

Consuming a secret as environment variable

Reference a secret named db-creds in a pod and inject key value pairs as environment variables. It is done under image in spec using envFrom.secretRef = db-creds

Mounting a secret

Same as configMap but instead of configMap.name in volumes it's called secret.secretName

Security Context

Security context defines privilege and access control for a pod or a container. Container level settings take precedence over pod level settings.

At the container level it can be set as a child of containers.
Containers.securityRootContext.runAsNonRoot or for pod as a child of spec
spec.securityContext.fsGroup

Resource Quota

Namespaces do not enforce quotas for resources like CPUs, disk space and memory. As a result, Kubernetes can consume unlimited resources until a maximum capacity limit is reached.

Using a resource quota establishes maximum amount of resources per namespace.

Hard resource limits can be defined by using ResourceQuota object kind as a child of spec. spec.hard

Kubectl create quota myreqo –hard=cpu=1,memory=1G,pods=2 –dry-run=client -o yaml

Service Accounts

Kubernetes administrator assigns roles to service accounts using RBAC(Role Based Access Control).

We use a service account to communicate with Kubernetes API server.

If a service account is not assigned, a pod uses a default SA.

Default SA has same permissions as an unauthenticated user. So, it cannot view or modify cluster settings.

K get sa –all-namespaces

Creating and assigning custom Service Accounts

Node Affinity or Node Selector

```
kubectl label nodes <your-node-name> accelerator=nvidia-tesla-p100
kubectl get nodes --show-labels

apiVersion: v1
kind: Pod
metadata:
  name: cuda-test
spec:
  containers:
```

```
    - name: cuda-test
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
  nodeName: # add this
    accelerator: nvidia-tesla-p100 # the selection label
```

Multi-container pods

Init container

Init containers provide initialization logic that needs to be run before the main app even starts. For init containers K specifies a special section – spec.initContainers. The files setup by init container can be shared to the main container through a volume.

Get logs of init container using **-c flag for container name**

Kubectl logs business-app -c configure

Side car

In Kubernetes world, the container that provides helper functionality to the primary application is called side car. They usually operate asynchronously and are used for logging, monitoring and synchronization. The file exchange between main app and sidecar happens through a volume.

Adapter Pattern

The adapter pattern transforms the output produced by the application to make it consumable by external applications or another part of the application. Sidecar can use adapter pattern.

Ambassador Pattern

Ambassador pattern provides a proxy for communicating with external services. Overarching goal of ambassador pattern is to hide the complexity of interaction with different parts of the system.

Use cases include re-try logic, authentication and authorization, monitoring latency or resource usage.

Observability

Making sure that app is running years after deployment using health probing.

Health Probes

Periodic processes that check for certain conditions.

Readiness Probe

This probe checks if the app is ready to serve requests.

Liveness Probe

This probe checks if the app is up or not.

Startup Probe

Some apps take long time to run. This probe is setup to wait for a predefined amount of time before the liveness probe is allowed to check on the application.

Each probe offers 3 different methods to check the health of app.

1) Custom command
2) HTTP GET request
3) TCP Socket Connection

Search on Kubernetes page for details during exam

Debugging in Kubernetes

Debugging pods

KubeYAML can be used outside the exam.

Kubectl describe pod pod-name

Kubectl get events

Kubectl logs pod-name

-f flag – streams the logs live

-- previous flag gets the previous run of the container, helpful if the container got restarted.

Kubectl exec pod-name -it -- /bin/sh

Debugging service

Tip#1

If you can't reach a pod that should map to a service check to make sure that both have same labels.

Kubectl describe service service-name

Kubectl get pod pod-name - -show-labels

Tip#2

Query the endpoints of the service

Kubectl get endpoints myservice

Tip#3

If service type is clusterIP, pod can only be reached through the service, if queried from the same node inside the same cluster.

If clusterIP is the proper type, open an interactive terminal from a temporary pod inside the same cluster and run curl or wget

Kubectl get services

Get IP and then run,

Kubectl run tmp –image=busybox -it – wget -o- IPAddressFromAbove:port

Tip#4

Check if the port mapping of the service is same as port of pod.

Pod Design

Pod design has the most weight in ckad exam.

Labels

Assigning a label

Imperatively using

K run label-test –image=ngnix –labels=tier=backend,env=dev

Or declaratively using metadata.labels

Retrieving a label

Kubectl describe pod pod-name| grep Labels:

For all pods

K get pods - -show-labels

Specific pod,

K get pod pod-name - -show-labels

Modifying labels for a live object

Using label command, this can be done. We can add, delete or modify labels.

K label pod pod-name region=au

K label pod pod-name region=us - - overwrite

K labelpod pod-name region-

Label Selectors

Labels become powerful only when combined with label selectors.

Labels can be selected imperatively using -l or - -selector and by using ==, = or != operators. They can be separated using a comma or an AND.

Labels can also be filtered based on set based operations in, notin and exists.

Get labels

k get pods -l ‘team in (shiny,legacy)’, app=V1.1.24 - - show-labels

Assign labels

k label deployment nginx version=v1

Change labels

K label pod nginx2 app=v2 –overwrite

Show label column

K get pods -L app

Filter by label

k get pods -l app=v2

Add a new label to pods with a label app=v1 or app=v2

K label pod -l ‘app in (v1,v2)’ tier=web

Remove label

K label pod pod1 app-

Label Selection In Manifest

Label selection in a manifest is done using spec.podSelector.matchLabels

Annotations

There is no imperative way to assign annotation using k run command. It is used to add metadata using metadata.annotations. We can use describe or get commands with grep to search for annotations.

But once the pod is created it can be annotated using annotate command

Kubectl annotate pod annotated-pod author=’Swapna karkala’

Kubectl annotate pod annotated-pod author=’Silpi Thota’ - - overwrite

Kubectl annotate pod annotated-pod author-

Check the annotations

K annotate pod nginx1 - -list

Deployments

Deployments are used to create replica sets as duplicates of pod so that if one goes down other can be brought up preventing app downtime which is the main selling point of Kubernetes.

Kubectl create deployment my-deployment –image=ngnix - -replicas=2

Get the replicaset that was created by above –

K describe deploy my-deployment

K get rs nginx-7ff4df89cd -o yaml

Get yaml for one of the pods created by the deployment

K get pods -l app=nginx

k get pod nginx-7ff4df89cd-zbwpv -o yaml

Deployments make heavy use of labels.

Deployments, Pods and Replica Sets are tied using same label.

Pod is controlled by a replicaset and replicaset is controlled by a deployment.

Deployment – parent

Replica set – child

Pod – child of replica set

K get deployments,pods,replicasets

K rollout history deployment my-deploy

K set image deployment my-deploy wordpress=wordpress:latest -> rolling out a new version or changing image to another one.

K rollout status deployment my-deploy

K rollout history deployment my-deploy - - revision=2

Different deployment strategies can be used

Rolling update, recreate, blue green, canary but this is out of scope for the exam

Rolling back to a previous version

K rollout undo deployment my-deploy - -to-revision=1

And verify that the pods have old deployment –

K describe pod pod-name | grep image

Manually scale a deployment

K scale deployment my-deploy - -replicas=5

Auto-scale a deployment

Auto scaling of a deployment is done using horizontal pod autoscalers and vertical pod autoscalers.

HPA – scales pods based on CPU and memory thresholds

VPA – scales based on historic metrics

Both autoscalers use the metrics server. HPA is a standard feature of K whereas VPA is an add-on. Only HPA is part of the exam.

Horizontal Pod Autoscaler

A deployment can autoscaled using the command autoscale deployment

K autoscale deployment my-deploy –cpu-percent=70 –min=2, --max=8

K get hpa

K describe hpa my-deploy

K delete hpa my-deploy

Pause deploy –

K rollout pause deploy nginx

Resume deploy –

K rollout resume deploy nginx

Jobs

Pod is meant for continuous operation where as job is meant for one time operation.

Job uses a single pod by default to perform the work. This is what K calls a non-parallel job.

K create job counter - - image=wordpress - - /bin/sh -c 'counter=1; done;'

Job can be created imperatively using command above or declaratively using kind as job.

Internally a non-parallel job is tracked using spec.template.spec.completions and spec.template.spec.parallelism and both these are 1 for a non-parallel job.

K describe job counter | grep Completions

Completions = 1

Parallelism = 1

If we want the job to complete more than 1 time, increase completions value.

If we want the job to run parallel threads concurrently increase parallelism to more than 1

Default number of restarts if a job fails is 6. Spec.backoffLimit

A job's manifest needs to explicitly declare the restart policy of a job using,

Spec.template.spec.restartPolicy. For a job it can only be onFailure or never.

onFailure – restarts container on job failure

Never – starts a new pod on job failure.

CronJobs

CronJob is not a one-time job, it's a job that runs periodically based on a scheduler.

K create cronjob current-date –schedule=" * * * * *" - - image=wordpress - - /bin/sh -c 'echo "Current Date: $(date)"'

By default, cronjob retains last 3 failed job history and last 1 successful pod history.

To configure job history retention limits to different value, set new values for spec.successfulJobHistoryLimit and spec.failedJobHistoryLimit.

Cronjob should be deleted if it dosen't run even after waiting for 15 sec

Spec.startingDeadlineSecond: 17

```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── hour (0 - 23)
# │ │ ┌──────────── day of the month (1 - 31)
# │ │ │ ┌──────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                    7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * command to execute
```

Services and Networking

We will focus on exposing the pods inside and outside of the cluster.

Network policies define the rules for incoming and outgoing traffic to and from the pods.

Services

Services provide discoverable names and load balancing for pod replicas.

If the label selector in the service matches with the label on the pod, the pod receives traffic.

Every service needs to define its type. Depending on it's type it exposes the matching pods.

Service Types

Cluster IP – exposes service on the cluster's internal IP. Does not expose pod outside the container, only internal.

NodePort – exposes service on each node's IP address using a static port. Exposes pod outside the container

LoadBalancer – exposes pod outside the cluster using an external cloud provider's load balancer.

ExternalName – Maps the service to a DNS name.

For the exam study nodeport and cluster IP.

Network Policies

Network policy combines couple of different attributes which form rules, and together it is called a policy.

podSelector – selects pod to apply the networkpolicy to

policyTypes – defines the type of traffic the network policy applies to – ingress or egress

Ingress – ports and from section

Egress – ports and to section

Taints and tolerations

K label nodes node01 size=large

`kubectl taint node <Node_Name> <key=value:TAINT_EFFECT>`

And then create a deployment and add toleration to the pods in the deployment

K create deployment my-svc - -image=nginx - -name=my-svc - -port=80 - -target-port=80 - - type: NodePort

There is 2 ways to add tolerations

`tolerations: - key: size operator: "Equal" value: large effect: NoSchedule`

Equal: If we specify operator as Equal, then we have to specify all the key, value, and effect option.

Exists: If we specify operator as Exists then it's not compulsory to mention key, value, and effect option.

If you want to allow your pod to tolerate every node taint then inside the pod toleration's part, you should mention only the operator: "Exists".

If you want to allow your pod to tolerate only the particular node taint then inside the pod toleration's part, you should mention only the operator: "Equal" and the label an value.

By defining this your pod able will tolerate every taint which was applied on the node.

1- **NoSchedule**

2- **PreferNoSchedule**

3- **NoExecute**

Creating Services

Create a service

K create service clusterip ngnix-service –tcp=80:80

Expose a new pod or a deployment

K run wordpress - -image=wordpress - -port=80 –expose

Expose an existing deployment

k expose deployment my-deploy - - type=NodePort- -port=80 - -target-port=80 - -name=my-service

Expose an existing pod

K expose pod pod-name –type=clusterip –port=3333 –target-port=80 –name=service-name

1) Access one pod from another pod in the same cluster using clusterIP by it's internal IP address
2) Access one pod on node's IP address using nodePort.

Network Policies

In a cluster any pod can talk to any other pod without restrictions even across namespaces. This poses security risks and does not define a mental model of the architecture.

Network policy defines whether to allow or disallow traffic.

Ingress – traffic coming into the pod

Egress – traffic going out of the pod.

Label selectors play an important role in applying network policies to pods.

Creating network Policies

Network policies cannot be created using imperative approach. They need to be created declaratively.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: payment-processor
      role: api
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: coffeeshop
```

K get networkpolicy

K describe networkpolicy api-allow

Isolating all pods in a namespace

The safest approach to defining network policy is to disallow all ingress and egress and loosen restrictions gradually based on need.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

K create -f default-deny-all.yaml

Restricting access to pods

If not specified by a network policy all ports are accessible. By specifying ports in network policy we disallow all other ports from being connected to ?

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: port-allow
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
```

```
        app: frontend
  ports:
  - protocol: TCP
    port: 8080
```

State Persistence

Container's temporary filesystem is not persisted beyond pod restart. Data needs to be persisted to volumes to prevent data loss.

Volume is a directory that's sharable between different containers of a pod.

The volumes have to be mounted into a container and that's done using persistence volume claims.

Pod needs to claim the persistent volume and mount it to an available directory path using PVC.

Volume Types

Volume type is the medium that backs the volume. Ex: AzureDisk, AWSElasticBlockStore etc.

Volume types relevant to the exam are

emptyDir – live only when pod is alive

config maps and secrets

hostpath – file or directory from host node's filesystem

nfs – network file system

pvc –

Creating and accessing volumes

PV needs to be creates using spec. No imperative commands.

It needs to define storage capacity using spec.capacity and access mode via spec.accessmodes

Access modes can be 3 types

ReadWriteOnce – read write by one node

ReadOnlyMany – RO by many nodes

ReadWriteMany – RW by many nodes

Persistence Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-pv
```

```
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/db
```

kubectl get pv db-pv

Persistence Volume Claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 512m
```

PVC says give me a PV that satisfies the criteria 512m and access mode readwriteonce. The binding to the appropriate pv happens automatically.

When PV is created it's status is AVAILABLE but when PVC is created it's status is set as BOUND.

Once PV and PVC are defined, PVC binds to PV, the PVC needs to be mounted to a pod that wants to consume it.

Mounting PVC in a pod

This is done by defining spec.volumes.persistancevolumeclaim in pod yaml.

K describe pvc pvc-name

Should now show the pod it is mounted to.

Custom Resource Definitions

Container runtimes like CRICTL and Docker, Blue green and Canary, rolling updates using helm package manager, rbac, role, role binding, cluster role, cluster role binding, limit ranges, fix api deprecations, multi container pods, networking between nodes and pods, why pods are not created, path or host based routing, json path, security contexts.

Common questions

1) List all namespaces in the cluster and redirect the output to a file
2) Create a pod with specified container name, image and labels
3) Create a secret and mount it as environment variables into a pod
4) Create a deployment with X image and Y tag. Change the Y tag to Z and record the changes
5) Set memory request for a pod to XMi and limit to half of maximum allowed for pods of that namespace
6) RBAC: ClusterRole, ClusterRoleBindings, Role, RoleBinding, Service Accounts
7) Add a sidecar container that prints the logs of main container. You need to create emptyDir and mount to both the containers. Command to print logs will be given
8) Create a pod with 2 containers. Image names and other info will be provided.
9) Create a network policy that allows all pods with the label key: value to allow traffic from only Y and Z pods in the same namespace. Allpw Egress traffic on port 53 also.
10) Setting CPU and memory limits to containers of a pod.
11) An ingress route is not working. Investigate and fix the issue. Debug service selector etc
12) Set security context on container to disable privilege escalation and run as user 2000
13) Create a canary deployment that route only 20% traffic
14) Create a pod and expose via node port 32000
15) Create a cronjob that runs every 5 mins, pods to deplete after 10s. 3 completions and3 failed.
16) Sort all pods by order of their creation
17) Write all namespaced resources to a file
18) Filter pods by certain replicas and write the results to a file
19) Deploy a helm chart with certain replicas
20) Upgrade the heml chart to the latest version
21) Rollback helm release
22) Build container image from a dockerfile using docker or podman. Push them to the registry.
23) A mounted service account is unable to list the PVSs in the given namespace. Fix it.
24) Copy service account token to a file.
25) Migrate a deployment and its service from X namespace to Y.
26) Create a job that runs an image and a command inside it. The job should complete successfully 3 times. You can create multiple pods to finish the job faster.
27) Find the Heml release which is in pending state.
28) Add readiness probe to the container to get http response from endpoint /health on port 5432. Probe should initially wait for 5 sec and periodically execute every 10 sec.

29) A deployment is using deprecated API. Fix it.
30) A pod template is fiven. Create a deployment of 3 replicas from this template.
31) Create a PV and PVC. Mount this PVC at /tmp of the container.
32) Create a TLS secret or a generic secret. Cert and key are provided.
33) Create a configmap from a file and mount it as volume.
34) Endpoints of a service are empty. Investigate and fix.
35) Change the service type of a service from clusterIP to nodeport using kubectl patch. Write the command to a file.
36) Create a ResourceQuota with the given specifications. Create a pod with the resources give,
37) Use an existing service account X with an existing deployment.