(Certified Kubernetes Administrator - CKA)

Created By – Jaswinder Kumar

Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to create or think about namespaces at all.

Kubernetes starts with four initial namespaces:

default - The default namespace for objects with no other namespace

kube-system - The namespace for objects created by the Kubernetes system

kube-public - This namespace is created automatically and is readable by all users (including those not authenticated). This namespace is mostly reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster. The public aspect of this namespace is only a convention, not a requirement.

kube-node-lease - This namespace for the lease objects associated with each node which improves the performance of the node heartbeats as the cluster scales.

You can run this command to get all namespaces:

kubectl get ns

```
WKMIN7290299:jkumar jaskumar$ kubectl get ns
NAME              STATUS   AGE
default           Active   17h
dev               Active   17h
kube-node-lease   Active   17h
kube-public       Active   17h
kube-system       Active   17h
WKMIN7290299:jkumar jaskumar$
```

In case you did not specify the namespace by default kubernetes will create the resources (not all) in default namespace.

Not all the namespace resources itself is created in namespace. To see what resources are created in namespace and what are not run the command:

kubectl api-resources --namespaced=true

```
WKMIN7290299:jkumar jaskumar$ kubectl api-resources --namespaced=true
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
configmaps                        cm                                          true         ConfigMap
endpoints                         ep                                          true         Endpoints
events                            ev                                          true         Event
limitranges                       limits                                      true         LimitRange
persistentvolumeclaims            pvc                                         true         PersistentVolumeClaim
pods                              po                                          true         Pod
podtemplates                                                                  true         PodTemplate
replicationcontrollers            rc                                          true         ReplicationController
resourcequotas                    quota                                       true         ResourceQuota
secrets                                                                       true         Secret
serviceaccounts                   sa                                          true         ServiceAccount
services                          svc                                         true         Service
controllerrevisions                            apps                           true         ControllerRevision
daemonsets                        ds           apps                           true         DaemonSet
deployments                       deploy       apps                           true         Deployment
replicasets                       rs           apps                           true         ReplicaSet
statefulsets                      sts          apps                           true         StatefulSet
localsubjectaccessreviews                      authorization.k8s.io           true         LocalSubjectAccessReview
horizontalpodautoscalers          hpa          autoscaling                    true         HorizontalPodAutoscaler
cronjobs                          cj           batch                          true         CronJob
jobs                                           batch                          true         Job
backendconfigs                    bc           cloud.google.com               true         BackendConfig
leases                                         coordination.k8s.io            true         Lease
endpointslices                                 discovery.k8s.io               true         EndpointSlice
ingresses                         ing          extensions                     true         Ingress
pods                                           metrics.k8s.io                 true         PodMetrics
frontendconfigs                                networking.gke.io              true         FrontendConfig
managedcertificates               mcrt         networking.gke.io              true         ManagedCertificate
ingresses                         ing          networking.k8s.io              true         Ingress
networkpolicies                   netpol       networking.k8s.io              true         NetworkPolicy
updateinfos                       updinf       nodemanagement.gke.io          true         UpdateInfo
poddisruptionbudgets              pdb          policy                         true         PodDisruptionBudget
rolebindings                                   rbac.authorization.k8s.io      true         RoleBinding
roles                                          rbac.authorization.k8s.io      true         Role
volumesnapshots                                snapshot.storage.k8s.io        true         VolumeSnapshot
```

kubectl api-resources --namespaced=false

```
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
componentstatuses                 cs                                          false        ComponentStatus
namespaces                        ns                                          false        Namespace
nodes                             no                                          false        Node
persistentvolumes                 pv                                          false        PersistentVolume
mutatingwebhookconfigurations                  admissionregistration.k8s.io   false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io   false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io           false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io         false        APIService
tokenreviews                                   authentication.k8s.io          false        TokenReview
selfsubjectaccessreviews                       authorization.k8s.io           false        SelfSubjectAccessReview
selfsubjectrulesreviews                        authorization.k8s.io           false        SelfSubjectRulesReview
subjectaccessreviews                           authorization.k8s.io           false        SubjectAccessReview
certificatesigningrequests        csr          certificates.k8s.io            false        CertificateSigningRequest
nodes                                          metrics.k8s.io                 false        NodeMetrics
storagestates                                  migration.k8s.io               false        StorageState
storageversionmigrations                       migration.k8s.io               false        StorageVersionMigration
runtimeclasses                                 node.k8s.io                    false        RuntimeClass
podsecuritypolicies               psp          policy                         false        PodSecurityPolicy
clusterrolebindings                            rbac.authorization.k8s.io      false        ClusterRoleBinding
clusterroles                                   rbac.authorization.k8s.io      false        ClusterRole
priorityclasses                   pc           scheduling.k8s.io              false        PriorityClass
volumesnapshotclasses                          snapshot.storage.k8s.io        false        VolumeSnapshotClass
volumesnapshotcontents                         snapshot.storage.k8s.io        false        VolumeSnapshotContent
csidrivers                                     storage.k8s.io                 false        CSIDriver
csinodes                                       storage.k8s.io                 false        CSINode
storageclasses                    sc           storage.k8s.io                 false        StorageClass
volumeattachments                              storage.k8s.io                 false        VolumeAttachment
```

You can create a namespace with command:

kubectl create ns <name>

Example:

kubectl create ns dev

You can also set the default namespace permanently for all kubectl commands with command:

kubectl config set-context --current --namespace=dev

You can validate it with command:

kubectl config view --minify | grep namespace:

Now if you spin-up a pod you will notice by default it will spin-up under **dev** namespace

```
WKMIN7290299:IAC jaskumar$ kubectl describe pod nginx
Name:         nginx
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Wed, 15 Jul 2020 13:24:46 +0530
Labels:       app=nginx
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx","namespace":"dev"},"spec":{"container...
Status:       Running
IP:           10.4.1.6
Containers:
  nginx:
    Container ID:   docker://88caa56f044f00633c07245423339732d881e05c3f76b70789543d73cc611b9c
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:8ff4598873f588ca9d2bf1be51bdb117ec8f56cdfd5a81b5bb0224a61565aa49
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 15 Jul 2020 13:24:52 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
```

Pods

Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. A Pod is a group of one or more containers with shared storage/network, and a specification for how to run the containers.

We can use this command to create a pod:

kubectl run --generator=run-pod/v1 nginx --image=nginx --image-pull-policy=always -o yaml --dry-run > nginx-pod.yaml

You can also attach labels to a pod with command:

kubectl run --generator=run-pod/v1 nginx --image=nginx --image-pull-policy=always --labels=app=nginx,env=dev -o yaml --dry-run > nginx-pod.yaml

To define the resource limits use below command:

kubectl run --generator=run-pod/v1 nginx --image=nginx --image-pull-policy=always --limits=cpu= 200m,memory=512Mi -o yaml --dry-run > nginx-pod.yaml

You can use below command to define the resource requests:

kubectl run --generator=run-pod/v1 nginx --image=nginx --image-pull-policy=always –requests=cpu= 200m,memory=512Mi -o yaml --dry-run > nginx-pod.yaml

In case you want to execute a command in pod run the command:

kubectl run --generator=run-pod/v1 nginx --image=nginx --command sleep 2000 -o yaml --dry-run > nginx-pod.yaml

You can also specify the service account for a pod with command:

kubectl run --generator=run-pod/v1 nginx --image=nginx --serviceaccount=service-account-1 -o yaml --dry-run > nginx-pod.yaml

To set environment variables in pod run the command:

kubectl run --generator=run-pod/v1 nginx --image=nginx --command sleep 2000 --env="ENV=dev" --env="APP=nginx" -o yaml --dry-run > nginx-pod.yaml

Scheduling pods on specific nodes

One option is to use **nodeScheduler** in pods but for that we first need to attach label to the nodes to which you want to schedule the pods.

Lets attach a label team=platform on one specific node

```
kubectl label nodes gke-cluster-1-default-pool-440e2077-kshp team=platform
```

Next, you can validate the node if label is attached or not with command:

```
kubectl get nodes --show-labels
```

```
WKMIN7290299:jkumar jaskumar$ kubectl get nodes --show-labels
NAME                                       STATUS   ROLES    AGE   VERSION          LABELS
gke-cluster-1-default-pool-d74ac0e8-86sn   Ready    <none>   17h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-86sn,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,topology.kube
rnetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
gke-cluster-1-default-pool-d74ac0e8-c8cq   Ready    <none>   17h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-c8cq,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,team=platform
,topology.kubernetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
gke-cluster-1-default-pool-d74ac0e8-tprj   Ready    <none>   17h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-tprj,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,topology.kube
rnetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
WKMIN7290299:jkumar jaskumar$
```

Example: nodeSelector

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
  nodeSelector:
   team: platform
```

Second option is to use **nodeName** in pod template file.

Example - nodeName

```
apiVersion: v1
kind: Pod
metadata:
  labels:
```

```
    app: nginx
  name: nginx
spec:
  nodeName: master
  containers:
    - image: nginx
        name: nginx
```

Nodes

Nodes can be worker node or master node. You can list all the nodes with command:

kubectl get nodes -o wide

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o wide
NAME                                     STATUS   ROLES    AGE   VERSION          INTERNAL-IP   EXTERNAL-IP      OS-IMAGE                             KERNEL-VERSION   CONTAINER-RU
NTIME
gke-cluster-1-default-pool-d74ac0e8-86sn   Ready    <none>   18h   v1.17.6-gke.11   10.128.0.4    34.72.110.14     Container-Optimized OS from Google   4.19.112+        docker://19.
3.6
gke-cluster-1-default-pool-d74ac0e8-c8cq   Ready    <none>   18h   v1.17.6-gke.11   10.128.0.2    104.154.201.41   Container-Optimized OS from Google   4.19.112+        docker://19.
3.6
gke-cluster-1-default-pool-d74ac0e8-tprj   Ready    <none>   18h   v1.17.6-gke.11   10.128.0.3    34.68.238.80     Container-Optimized OS from Google   4.19.112+        docker://19.
3.6
```

To list all nodes with the labels attached to them:

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes --show-labels
NAME                                     STATUS   ROLES    AGE   VERSION          LABELS
gke-cluster-1-default-pool-d74ac0e8-86sn   Ready    <none>   18h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-86sn,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,topology.kube
rnetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
gke-cluster-1-default-pool-d74ac0e8-c8cq   Ready    <none>   18h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-c8cq,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,team=platform
,topology.kubernetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
gke-cluster-1-default-pool-d74ac0e8-tprj   Ready    <none>   18h   v1.17.6-gke.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=e2-medium,beta.kubernetes.io/os=li
nux,cloud.google.com/gke-nodepool=default-pool,cloud.google.com/gke-os-distribution=cos,failure-domain.beta.kubernetes.io/region=us-central1,failure-domain.beta.kubernetes.io/zone=u
s-central1-c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cluster-1-default-pool-d74ac0e8-tprj,kubernetes.io/os=linux,node.kubernetes.io/instance-type=e2-medium,topology.kube
rnetes.io/region=us-central1,topology.kubernetes.io/zone=us-central1-c
```

In case we just need to extract the node name from the output we can use **JSONPATH** as output.

Example – fetch only node names

kubectl get nodes -o jsonpath='{.items[*].metadata.name}'

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o jsonpath='{.items[*].metadata.name}'
gke-cluster-1-default-pool-d74ac0e8-86sn gke-cluster-1-default-pool-d74ac0e8-c8cq gke-cluster-1-default-pool-d74ac0e8-tprj
```

Example – fetch os images

kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.osImage}'

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.osImage}'
Container-Optimized OS from Google Container-Optimized OS from Google Container-Optimized OS from Google
```

Example – fetch InternalIP

kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type == "InternalIP")].address}'

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type == "InternalIP")].address}'
10.128.0.4 10.128.0.2 10.128.0.3WKMIN7290299:IAC jaskumar$
```

Example – fetch ExternalIP

kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type == "ExternalIP")].address}'

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type == "ExternalIP")].address}'
34.72.110.14 104.154.201.41 34.68.238.80WKMIN7290299:IAC jaskumar$
WKMIN7290299:IAC jaskumar$
```

Example – fetch kernalVersion

kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.kernelVersion}'

```
WKMIN7290299:IAC jaskumar$ kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.kernelVersion}'
4.19.112+ 4.19.112+ 4.19.112+WKMIN7290299:IAC jaskumar$
```

ConfigMap

A ConfigMap is an API object used to store non-confidential data in key-value pairs. **Pods** can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a **volume**.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

There are three way to create the config map:

Using --from-literal

kubectl create configmap config1 --from-literal=app=nginx --from-literal=env=dev -o yaml --dry-run > config.yaml

Using --from-file

kubectl create configmap config2 --from-file=/tmp/app.txt -o yaml --dry-run > config-file.yaml

Where /tmp/app.txt contains the following:
app=nginx

Using --from-env-file

kubectl create configmap config3 --from-env-file=/tmp/config.txt -o yaml --dry-run > config-env-file.yaml

Where /tmp/config.txt contains the following:
app=nginx
env=prod

When passing **--from-env-file** multiple times to create a ConfigMap from multiple data sources, only the last env-file is used

There are multiple ways to access config map in pods:

Exposing as environment variable

Option-1:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      command: ["cat","/etc/config/app.txt"]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: config2
  nodeSelector:
    team: platform
```

Option-2:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      command: ["cat","/etc/config/app.txt"]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: config2
  nodeSelector:
    team: platform
```

Mounting as Volume

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      command: ["cat","/etc/config/app.txt"]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: config2
  nodeSelector:
       team: platform
```

Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in an image. Users can create secrets and the system also creates some secrets. There are three way to create the secrets:

Using --from-literal

kubectl create secret generic secret1 --from-literal=username=root --from-literal=password=password123 -o yaml --dry-run > secret.yaml

Using --from-file

kubectl create secret generic secret2 --from-file=/tmp/secret.txt -o yaml --dry-run > secret-file.yaml

Where /tmp/secret-file.txt contains the following:
username=root

Using --from-env-file

kubectl create secret generic secret2 --from-file=/tmp/secret.txt -o yaml --dry-run > secret-env-file.yaml

Where /tmp/secret.txt contains the following:
username=root
password=password123

When passing **--from-env-file** multiple times to create a secret from multiple data sources, only the last env-file is used

If a field, such as **username**, is specified in both data and stringData, the value from stringData is used. For example, the following Secret definition:

You can access secrets in container via two options:

Using environment variables

Option-1:

```
apiVersion: v1
kind: Pod
```

```
  metadata:
    labels:
      app: nginx
    name: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
        command: ["echo","==> username name
  is","$(username)"]
        envFrom:
          - secretRef:
              name: secret1
    nodeSelector:
      team: platform
```

Option-2:

```
  apiVersion: v1
  kind: Pod
  metadata:
    labels:
      app: nginx
    name: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
        command: ["echo","==> username name
  is","$(username)"]
        envFrom:
          - secretRef:
              name: secret1
    nodeSelector:
      team: platform
```

Mounting volumes

```
  apiVersion: v1
  kind: Pod
  metadata:
```

```
    labels:
      app: nginx
    name: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
        command: ["cat","/etc/secrets/secret-file.txt"]
        volumeMounts:
          - name: volume1
            mountPath: /etc/secrets
    volumes:
      - name: volume1
        secret:
          secretName: secret-2
    nodeSelector:
         team: platform
```

Authentication

All Kubernetes clusters have two categories of users: service accounts managed by Kubernetes, and normal users.

Normal users are assumed to be managed by an outside, independent service. An admin distributing private keys, a user store like Keystone or Google Accounts, even a file with a list of usernames and passwords. In this regard, *Kubernetes does not have objects which represent normal user accounts.* Normal users cannot be added to a cluster through an API call.

Even though normal user cannot be added via an API call, but any user that presents a valid certificate signed by the cluster's certificate authority (CA) is considered authenticated.

In contrast, service accounts are users managed by the Kubernetes API. They are bound to specific namespaces, and created automatically by the API server or manually through API calls.

To create basic authentication using static file with password follow under mentioned steps:

1. Create a static user and password file at **/tmp/users.csv** with below contents:
   - password123,user1,u0001,group1
   - password456,user2,u0002,group2

2. Modify the api-server static pod at /etc/kubernetes/manifests/kube-apiserver.yaml file to add the following switch under command:

   --basic-auth-file=/tmp/users/user-details.csv

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --authorization-mode=Node,RBAC
    <content-hidden>
    - --basic-auth-file=/tmp/users/user-details.csv
```

3. Create required roles and role bindings for the users using manifest as shown below:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

---
# This role binding allows "jane" to read pods in the "default
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: user1 # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or C
  apiGroup: rbac.authorization.k8s.io
```

4. You can authenticate with command:

curl -v -k https://localhost:6443/api/v1/pods -u "user1:password123"

Creating User

There are a few steps are required in order to get normal user to be able to authenticate and invoke API. First, this user must have certificate issued by the Kubernetes Cluster, and then present that Certificate into the API call as the Certificate Header, or through the kubectl. To allow a normal user say **jaskuma**r to authenticate with kubernetes to follow under mentioned steps:

1. Create a private key with command:

   openssl genrsa -out jaskumar.key 2048

2. Create CSR certificate with command:

   openssl req -new -key jaskumar.key -out jaskumar.csr -subj "/CN=jaskumar/O=platform"

   Assuming user jaskumar is part of platform group.

3. Encode the certificate with command:

   cat jaskumar.csr | base64 | tr -d "\n"

4. Create a **CertificateSigningRequest** template which looks like the one shown below:

```
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: jaskumar-k8s-csr-request
spec:
  groups:
  - system:authenticated
  request:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ
2F6Q0NBVk1DQVFBd0pqRVJNQThHQTFVRUF3d0lhbUZ6YTNWdFlYSX
hFVEFQQmdOVkJBb01DSEJzWVhSbQpiM0p0TUlJQklqQU5CZ2txaGt
pRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dLQ0FRRUFySzhvWEhIMUpN
NEVLdlhMCldLRHE3MjdEYnovbzdoL1JyYnIzcnpoMWNYMFUrR1RSM
HFrdUdodktyTWMyTkhHQ1FraDJzUzdxVVlpSjNxN20KOHBLVFliY2
9kZi8zWVdUMWVDczRWcHVtM3lJRzFVN3FmNjErbVNwcDJnYlhhSG9
POUR1cVZVNkszSFdsay9HUgpoTmQxWUNYTGdVOXFkVGxRVDNCY3lU
SHEzb3p5V3lkb0tDKzNYNWtNUEZYZWF2VjNHdWhNejZiNkJFTmY3d
jIzCk1NTXBYSlp1ZEVQY0ZlVzErQ01reVpTdHllczI2ZWdFMkhiM2
VIdjN3MUlSNGl0YndxcjAyc1VmWmIrT1NMYWgKeGRySjB2bC9mcTR
pVkNTdmgrL1puLzdleXp1MU5GNy9kbUFCTVBpN2wrdDdaazh5bGgz
RWZzVk85amxhUlNRdApBeUxlOVFJREFRQUJvQUF3RFFZSktvWklod
mNOQVFFTEJRQURnZ0VCQUdLNmJRUGZ6MXhaR282ZkM4WVJvOG90Cl
phSGE2Wjg1K2NENG9IdVo2WW4zY3lWTTBjcWVJbzMrNFVrYTNZbVd
4b1VhSUNLUkNseW1kb3FKN2J5MTVHT2gKdzNsQ3h0M3BOSnFVdlpV
```

```
TldZSC9NT1d4cVVCZmdZSWg2UGl4dmxJN2pzVENFREU3Sm9OTk13T
mxaRnlOQlI2NgpLYVZ2MXRvcUFFWnU2WTZSdFBLRzZXclJSNEU1NE
xzMkp2K0NGRy9rbnpkREUrKzBNUHFXWjJpMUMvYkpacmwxCi9TUkd
ySVdTcGp1d09QYkU3M1pIT0ZlWldSWmNadFBmeUJBSDBIUFdVZkJP
amtZM1FBMXkzT1B3L3pYSFQ1WVAKMTV2dUUxYzlWYUVpNXRMK0I4Q
k5qbU8yQUtRMHdWaExWbkExUFNHS1hrMXJrN09MSEJVTnZ0RVRpQ0
xEc0xRPQotLS0tLUVORCBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0
K
  usages:
    - client auth
```

5. Under **request** you have to paste the output of step 3.
6. Apply the template with command:
   kubectl apply -f csr.yaml

```
WKMIN7290299:kubernets jaskumar$ kubectl apply -f csr.yaml
certificatesigningrequest.certificates.k8s.io/jaskumar-k8s-csr-request created
```

7. You can validate the CSR with command:

   kubectl get csr

```
WKMIN7290299:kubernets jaskumar$ kubectl get csr
NAME                        AGE   REQUESTOR                          CONDITION
jaskumar-k8s-csr-request    68s   [redacted]@gmail.com               Pending
WKMIN7290299:kubernets jaskumar$
```

8. As you can see the certificate is in pending condition and therefore we need to approve it with command:

   kubectl certificate approve jaskumar-k8s-csr-request

```
WKMIN7290299:kubernets jaskumar$ kubectl certificate approve jaskumar-k8s-csr-request
certificatesigningrequest.certificates.k8s.io/jaskumar-k8s-csr-request approved
WKMIN7290299:kubernets jaskumar$
```

9. Now you can validate the certificate again with the command:

   kubectl get csr

```
WKMIN7290299:kubernets jaskumar$ kubectl get csr
NAME                        AGE     REQUESTOR                          CONDITION
jaskumar-k8s-csr-request    4m23s   [redacted]gmail.com                Approved,Issued
WKMIN7290299:kubernets jaskumar$
```

Now it is approved.

10. Next you can get the certificate with command:

    kubectl get csr jaskumar-k8s-csr-request -o yaml

    The Certificate value is in Base64-encoded format under status.certificate

11. To retrieve this certificate we will run the command:

    kubectl get csr jaskumar-k8s-csr-request -o jsonpath='{.status.certificate}' | base64 --decode > jaskumar.crt

Create Role

Next we need to create a role with command:

kubectl create role platform --verb=get,list,update,delete,create,watch --resource=pods

Considering we want to grant access only on pods.

Create Role Binding

To create role binding run the command:

kubectl create rolebinding platform-role-binding --role=platform --user=jaskumar

Add user in kubeconfig file

Lastly we have to add this user in kubeconfig file. W can do that with command:

kubectl config set-credentials jaskumar --client-key=/Users/jaskumar/kubernets/jaskumar.key --client-certificate=/Users/jaskumar/kubernets/jaskumar.crt --embed-certs=true

Set the context with command:

kubectl config set-context jaskumar --cluster=gke_bamboo-strata-280313_us-central1-c_cluster-1 --user=jaskumar

To validate this change the context with command:

kubectl config use-context jaskumar

As we granted permission only on pods and therefore if you try to run the command on any other resource you will get the error as shown in below:

```
WKMIN7290299:kubernets jaskumar$ kubectl get ns
Error from server (Forbidden): namespaces is forbidden: User "jaskumar" cannot list resource "namespaces" in API group "" at the cluster scope
WKMIN7290299:kubernets jaskumar$
```

You can list the pods as shown below:

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods
NAME                                                READY   STATUS             RESTARTS   AGE
nginx                                               0/1     CrashLoopBackOff   606        2d2h
nginx-pod-gke-cluster-1-default-pool-d74ac0e8-c8cq  1/1     Running            0          139m
nginx-static-pod-gke-cluster-1-default-pool-d74ac0e8-86sn  1/1     Running            0          34h
```

Get the group name for which CSR is raised

You can get the details of the group for which CSR is raised with the command:

kubectl get certificate -o yaml > certificate.yaml

This manifest should look like the one shown below:

```yaml
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: jaskumar-k8s-csr-request
spec:
  groups:
  - system:authenticated
  request: <Certificate>
  usages:
  - client auth
```

Here CSR is raised for group **system:authenticated**

Reject the CSR

You can either approve or deny the CSR as an admin. To deny any CSR request use the command:

kubectl certificate deny <CSR_NAME>

Delete a CSR

You can delete a CSR with command:

kubectl delete csr <CSR_NAME>

API Groups

You can start a proxy to the kubernetes API with command:

kubectl proxy &

Now you can access the api-server with localhost as shown below:

curl http://localhost:8001

```
WKMIN7290299:kubernets jaskumar$ curl http://localhost:8001
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/admissionregistration.k8s.io",
    "/apis/admissionregistration.k8s.io/v1",
    "/apis/admissionregistration.k8s.io/v1beta1",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1",
    "/apis/apiextensions.k8s.io/v1beta1",
    "/apis/apiregistration.k8s.io",
    "/apis/apiregistration.k8s.io/v1",
    "/apis/apiregistration.k8s.io/v1beta1",
    "/apis/apps",
    "/apis/apps/v1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/autoscaling/v2beta1",
    "/apis/autoscaling/v2beta2",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v1beta1",
```

Service Accounts

You can use below command to create the service account:

kubectl create serviceaccount dev-svc

Note – this is not in scope of CKA exam

TLS Certificates

You can find the detail of all the certificates and private keys used by different control plane components in their corresponding yaml file available at location **/etc/kubernetes/manifests**.

Few of the Key points about TLS certificates:

- Default TLS certificate location on master

  /etc/kubernetes/pki

- Certificates used by apiserver are:

  api-server
  - /etc/kubernetes/pki/apiserver.crt
  - /etc/kubernetes/pki/apiserver.key
  - /etc/kubernetes/pki/apiserver-etcd-client.crt
  - /etc/kubernetes/pki/apiserver-etcd-client.key
  - /etc/kubernetes/pki/apiserver-kubelet-client.crt
  - /etc/kubernetes/pki/apiserver-kubelet-client.key
  - /etc/kubernetes/pki/etcd/ca.crt

  etcd
  - /etc/kubernetes/pki/etcd/server.crt
  - /etc/kubernetes/pki/etcd/server.key
  - /etc/kubernetes/pki/etcd/ca.crt

  controller-manager
  - /etc/kubernetes/pki/ca.crt
  - / etc/kubernetes/pki/ca.key

Validate common names in certificates

We can use the below command to get the certificate details:

openssl x509 -in <CRT_FILE> -text -noout

```
master $ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1602484644455389325 (0x163d2b4b36c8188d)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = kubernetes
        Validity
            Not Before: Jul 21 14:44:22 2020 GMT
            Not After : Jul 21 14:44:22 2021 GMT
        Subject: CN = kube-apiserver
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
```

Get CA name who issued the certificate

Check for **Issuer: CN** text in output as shown in below screenshot:

```
master $ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1602484644455389325 (0x163d2b4b36c8188d)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = kubernetes
        Validity
            Not Before: Jul 21 14:44:22 2020 GMT
            Not After : Jul 21 14:44:22 2021 GMT
        Subject: CN = kube-apiserver
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
```

Get SAN list of certificate

```
                    8c:7f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:master, DNS:kubernetes, DNS:kubernetes.default, DNS:kubernetes.default.svc,
dress:10.96.0.1, IP Address:172.17.0.35
    Signature Algorithm: sha256WithRSAEncryption
         e3:ba:ce:ec:91:e1:61:00:d3:60:2e:5f:94:50:ab:ee:43:a4:
         53:2c:1d:c0:93:a7:03:46:a1:ae:2f:3e:47:50:32:08:40:59:
         dd:19:47:30:e8:a5:b3:5c:5b:01:ca:c3:62:b4:86:64:5e:b8:
```

authorization modes configured on the cluster.

```
   Host Port:      <none>
   Command:
     kube-apiserver
     --advertise-address=172.17.0.94
     --allow-privileged=true
     --authorization-mode=Node,RBAC
     --client-ca-file=/etc/kubernetes/pki/ca.crt
     --enable-admission-plugins=NodeRestriction
     --enable-bootstrap-token-auth=true
     --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
     --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
     --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
     --etcd-servers=https://127.0.0.1:2379
     --insecure-port=0
```

Kubeconfig

Kubeconfig files to organize information about clusters, users, namespaces, and authentication mechanisms. The kubectl command-line tool uses kubeconfig files to find the information it needs to choose a cluster and communicate with the API server of a cluster.

By default, kubectl looks for a file named config in the **$HOME/.kube** directory. You can specify other kubeconfig files by setting the KUBECONFIG environment variable or by passing the **--kubeconfig** flag.

So the default kubeconfig file is **$HOME/.kube/config**.

kubectl config view

It will display the default config file as shown in below:

```
WKMIN7290299:kubernets jaskumar$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://35.239.132.102
  name: gke_bamboo-strata-280313_us-central1-c_cluster-1
contexts:
- context:
    cluster: gke_bamboo-strata-280313_us-central1-c_cluster-1
    user: gke_bamboo-strata-280313_us-central1-c_cluster-1
  name: gke_bamboo-strata-280313_us-central1-c_cluster-1
current-context: gke_bamboo-strata-280313_us-central1-c_cluster-1
kind: Config
preferences: {}
users:
- name: gke_bamboo-strata-280313_us-central1-c_cluster-1
  user:
```

To view some other kubeconfig file run the command:

kubectl config view --kubeconfig myConfig

You can set the context using command:

kubectl config set-context --user=<USER> --cluster=<CLUSTER> --namespace=<NAMESPACE> --current

Example:

```
kubectl config set-context --user=jaskumar --cluster=cluster-1 --
namespace=default --current
```

Security Context

A security context defines privilege and access control settings for a Pod or Container. Security context settings include, but are not limited to:

- Discretionary Access Control: Permission to access an object, like a file, is based on UID and GID
- **Security Enhanced Linux (SELinux):** Running as privileged or unprivileged
- **Linux Capabilities:** Give a process some privileges, but not all the privileges of the root user
- **Seccomp:** Filter a process's system calls

To specify security settings for a Pod, include the **securityContext** field in the Pod specification. The securityContext field is a **PodSecurityContext** object. The security settings that you specify for a Pod apply to all Containers in the Pod.

To specify security settings for a Container, include the securityContext field in the Container manifest. Security settings that you specify for a Container apply only to the individual Container, and they override settings made at the Pod level when there is overlap. Container settings do not affect the Pod's Volumes.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-2
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - name: sec-ctx-demo-2
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
```

The output shows that the processes are running as user 2000. This is the value of runAsUser specified for the Container. It overrides the value 1000 that is specified for the Pod.

You can use Linux Capabilities to grant certain privileges to a process without granting all the privileges of the root user. To add or remove Linux capabilities for a Container, include the capabilities field in the securityContext section of the Container manifest.

Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
  - name: sec-ctx-4
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
```

Network Policy

A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints. NetworkPolicy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.

Network policies are implemented by the network plugin. To use network policies, you must be using a networking solution which supports NetworkPolicy. Creating a NetworkPolicy resource without a controller that implements it will have no effect

By default, pods are non-isolated; they accept traffic from any source. Pods become isolated by having a NetworkPolicy that selects them. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy.

The NetworkPolicy resource contains the following mandatory field:

- apiVersion
- kind
- metadata
- spec

NetworkPolicy spec section contains the following:

- **podSelector –** It is used to select the pods to which the policy is applied. An empty podSelector selects all the pods in the namespace.
- **policyTypes –** policy types can be **Ingress, Egress** or both. If no policy type is specified then default is **Ingress**.

There are four kinds of selectors that can be specified in an ingress from section or egress to section:

- podSelector
- namespaceSelector
- namespaceSelector and podSelector
- ipBlock

AND condition example:

```
ingress:
  - from:
    - namespaceSelector:
        matchLabels:
```

```
            name: dev
        podSelector:
          matchLabels:
            app: frontend
```

It contains a single **from** element allowing connections from Pods with the label **app=frontend** in namespaces with the label **name=dev**. So both the conditions should match.

OR condition example:

```
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            name: dev
      - podSelector:
          matchLabels:
           app: frontend
```

It contains two elements in the **from** array, and allows connections from Pods in the local Namespace with the label **app=frontend**, *or* from any Pod in any namespace with the label **name=dev**.

The sample NetworkPolicy manifest looks like the one shown below:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
  namespace: default
spec:
  podSelector:
    matchLabels:
      color: blue
  ingress:
  - from:
    - podSelector:
        matchLabels:
          color: red
    ports:
```

```
    - port: 80
```

As per this policy incoming traffic to pod with label **color: blue** is allowed in case it is coming from pod with label **color: red**, on port 80.

Consider another example:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
  namespace: default
spec:
  podSelector:
    matchLabels:
      color: blue
  ingress:
  - from:
    - podSelector:
        matchLabels:
          color: red
      namespaceSelector:
        matchLabels:
          shape: square
    ports:
    - port: 80
```

Here this network policy is allowing incoming traffic to pod with label **color:blue** in case it is coming from pod with label **color: red** and namespace with label **shape: square,** on port 80.

Deny default ingress

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: policy-demo
spec:
  podSelector:
```

```
    matchLabels: {}
  policyTypes:
  - Ingress
```

Deny default egress

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: policy-demo
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
```

You can create a network policy with command:

kubectl create networkpolicy default-deny

To get list of network policy run command:

kubectl get networkpolicies

```
master $ kubectl get networkpolicies
NAME             POD-SELECTOR   AGE
payroll-policy   name=payroll   33s
master $
```

Hostname	Namespace	Type	Root	IP Address
web-service	apps	svc	cluster.local	10.107.37.188
10-244-2-5	apps	pod	cluster.local	10.244.2.5
10-244-1-5	default	pod	cluster.local	10.244.1.5

Troubleshooting

We may have to troubleshoot at three levels:

1. Application Level
2. Control Planes running on master
3. Worker nodes

Application Troubleshooting

To troubleshoot application issue at high level we should perform the following steps:

- Check all relevant application containers are running fine in default or specified namespace. You can use **kubectl get pods -n <namespace>** command.
- Check services are pointing to write application pod via matchLabels. The label should match
- Validate the pods by describing them with **kubectl describe pod <pod-name>** and check for events
- Validate all the mentioned ports and IP's properly mapped with service via endpoints
- Check logs with command **kubectl logs -f <pod> -p <pod-name>**
- In case pods are deployed as deployments. Validate deployment as well from environment and replicas perspective.

Control Plane Troubleshooting

To troubleshoot control plane components at master node we can follow under mentioned high level steps:

- Check all pods are running fine in **kube-system** namespace with command: **kubectl get pods -n kube-system**
- In case control plane components are deployed as service we have to check the service status. We should validate the following services:
    - kube-apiserver
    - kube-controller-manager
    - kube-scheduler
- Validate the configurations at **/etc/kubernetes/manifests** file to figure out the root cause. Usually check mounted volumes path, binary name, certificates path or name in the configuration file of failed component
- Check certificates validation and expiry date etc with **openssl x509 -in <certificate> -text** command

Worker Node Troubleshooting

- Validate nodes health with command: **kubectl get nodes**
- Try to describe the node with command: **kubectl describe node <node-name>**
- In case did not find anything suspicious we have to ssh to the faulty node
- Check **kubelet** service status with command: **service kubelet status -l**
- If did not find anything run command: **journalctl -u kubelet** and check the logs.
- Validate /var/lib/kubelet/config.conf file
- Validate /etc/kubernetes/kubelet.conf
- Validate api-server URL with command: **kubelet cluster-info**
- Check staticPod path in /var/lib/kubelet/config.yaml

Taint & Toleration

Taints are used to repel Pods from specific nodes. This is quite similar to the node anti-affinity feature. However, taints and tolerations take a slightly different approach. Instead of applying the label to a node, we apply a taint that tells a scheduler to repel Pods from this node if it does not match the taint. Only those Pods that have a *toleration* for the taint can be let into the node with that taint.

Use cases for taint and tolerations

In general, taints and tolerations support the following use cases:

1. **Dedicated nodes**. Users can use a combination of node affinity and taints/tolerations to create dedicated nodes. For example, you can limit the number of nodes onto which to schedule Pods by using labels and node affinity, apply taints to these nodes, and then add corresponding tolerations to the Pods to schedule them on those particular nodes. We'll show how to implement this use case in detail at the end of the article.

2. **Nodes with special hardware**. if you have nodes with special hardware (e.g GPUs) you want to repel Pods that do not need this hardware and attract Pods that do need it. This can be done by tainting the nodes that have the specialized hardware (e.g. **kubectl taint nodes <nodename> special=true:NoSchedule**) and adding corresponding toleration to Pods that must use this special hardware.

3. **Taint-based Evictions**. New Kubernetes versions allow configuring per-Pod eviction behavior on nodes that experience problems. Taint-based evictions will be discussed in detail below.

You can taint a node with command:

kubectl taint node gke-cluster-1-default-pool-96cbbcce-v8tt team=platform:NoSchedule

```
WKMIN7290299:kubernets jaskumar$ kubectl taint node gke-cluster-1-default-pool-96cbbcce-v8tt team=platform:NoSchedule
node/gke-cluster-1-default-pool-96cbbcce-v8tt tainted
WKMIN7290299:kubernets jaskumar$
```

You can verify the taint with command:

kubectl describe node gke-cluster-1-default-pool-96cbbcce-v8tt

```
WKMIN7290299:kubernets jaskumar$ kubectl describe node gke-cluster-1-default-pool-96cbbcce-v8tt
Name:               gke-cluster-1-default-pool-96cbbcce-v8tt
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/instance-type=e2-medium
                    beta.kubernetes.io/os=linux
                    cloud.google.com/gke-nodepool=default-pool
                    cloud.google.com/gke-os-distribution=cos
                    failure-domain.beta.kubernetes.io/region=us-central1
                    failure-domain.beta.kubernetes.io/zone=us-central1-c
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=gke-cluster-1-default-pool-96cbbcce-v8tt
                    kubernetes.io/os=linux
Annotations:        container.googleapis.com/instance_id: 3067237049640574398
                    node.alpha.kubernetes.io/ttl: 0
                    node.gke.io/last-applied-node-labels: cloud.google.com/gke-nodepool=default-pool,cloud.google.com
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Sun, 19 Jul 2020 11:40:37 +0530
Taints:             team=platform:NoSchedule
Unschedulable:      false
Conditions:
```

The taint has the format <taintKey>=<taintValue>:<taintEffect> . Thus, the taint we just created has the key "**team**", the value "**platform**", and the taint effect NoSchedule. A taint's key and value can be any arbitrary string and the taint effect should be one of the supported taint effects such as NoSchedule , NoExecute , and PreferNoSchedule .
Taint effects define how nodes with a taint react to Pods that don't tolerate it. For example, the NoSchedule taint effect means that unless a Pod has matching toleration, it won't be able to schedule onto the host1 . Other supported effects include PreferNoSchedule and NoExecute . The former is the "soft" version of NoSchedule . If the PreferNoSchedule is applied, the system will *try not* to place a Pod that does not tolerate the taint on the node, but it is not required. Finally, if the NoExecute effect is applied, the node controller will immediately evict all Pods without the matching toleration from the node.

If you don't need a taint anymore, you can remove it like this:

kubectl taint node gke-cluster-1-default-pool-96cbbcce-v8tt team:NoSchedule-

Adding Tolerations

As we know, taints and tolerations work together. Without a toleration, no Pod can be scheduled onto a node with a taint. That's not what we trying to achieve! Let's now create a Pod with a toleration for the taint we created above. Tolerations are specified in the PodSpec as shown in below manifest file:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
```

```
    labels:
      app: frontend
  spec:
    containers:
    - name: nginx
      image: nginx
    tolerations:
    - key: "team"
      operator: "Equal"
      value: "platform"
      effect: "NoSchedule"
```

As you see, the Pod's toleration has the key "**team**", the value "**platform**", and the effect "**NoSchedule**", which exactly matches the taint we applied earlier. Therefore, this Pod *can* be scheduled onto the gke-cluster-1-default-pool-96cbbcce-v8tt . However, this does not mean that the Pod will be scheduled onto that exact node because we did not use node affinity or nodeSelector .

The second Pod below can be also scheduled onto a tainted node although it uses the operator "**Exists**" and does not have the key's value defined.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-1
  labels:
    app: frontend
spec:
  containers:
  - name: nginx-1
    image: nginx
  tolerations:
  - key: "team"
    operator: "Exists"
    effect: "NoSchedule"
```

This demonstrates the following rule: if the operator is Exists the toleration matches the taint if keys and effects are the same (no value should be specified). However, if the operator is Equal , the toleration's and taint's values should be also equal.

Consider the below pod manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-1
  labels:
    app: frontend
spec:
  containers:
  - name: nginx-1
    image: nginx
  tolerations:
  - operator: "Exists"
```

As you see, the key , value , and effect fields of this Pod's toleration are empty. We just used the operator: Exists . This toleration will match all keys, values, and effects. In other words, it will tolerate any node.

NoExecute Effect

The taint with the NoExecute effect results in the eviction of all Pods without a matching toleration from the node. When using the toleration for the NoExecute effect you can also specify an optional **tolerationSeconds** field. Its value defines how long the Pod that tolerates the taint can run on that node before eviction and after the taint is added. Let's look at the manifest below:

Consider the below manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: frontend
spec:
  containers:
  - name: nginx
    image: nginx
  tolerations:
  - key: "team"
    operator: "Equal"
```

```
    value: "platform"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

If this Pod is running and a matching taint is added to the node, it will stay bound to the node for 3600 seconds. If the taint is removed by that time, the Pod won't be evicted.

In general, the following rules apply for the NoExecute effect:

- Pods with no tolerations for the taint(s) are evicted immediately.
- Pods with the toleration for the taint but that do not specify tolerationSeconds in their toleration stay bound to the node forever.
- Pods that tolerate the taint with a specified tolerationSeconds remain bound for the specified amount of time.

Node Affinity

There are four ways to schedule the pods on specific nodes:

1. nodeName
2. nodeSelector
3. Taint & Tolerations
4. Node Affinity

The Node affinity is the advanced feature and provides advanced capabilities to limit pod placement on specific nodes. Affinity is of 3 types such as **node affinity**, **pod affinity** and **anti-affinity**. It's important to read these as properties of a pod.

Node affinity is conceptually similar to nodeSelector -- it allows you to constrain which nodes your pod is eligible to be scheduled on, based on labels on the node.

There are currently two types of node affinity:

- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution.

What DuringScheduling means?

- The pod is NOT yet created and going to be created for the first time.
- Usually when the pod is created the affinity rules will be applied to place the pod in the right nodes.

What DuringExecution means?

- In this state pod has been running and the change is made in the environment that affects nodeAffinity such as a change in the label of the node.

You can think of **requiredDuringSchedulingIgnoredDuringExecution** as "hard" and **preferredDuringSchedulingIgnoredDuringExecution** as "soft" respectively, in the sense that the former specifies rules that *must* be met for a pod to be scheduled onto a node, while the latter specifies *preferences* that the scheduler will try to enforce but will not guarantee.

The "**IgnoredDuringExecution**" part of the names means that, similar to how nodeSelector works, if labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod will still continue to run on the node.

How to apply nodeAffinity

To apply node affinity we need to follow under mentioned steps:

1. We first need to apply the label to the node with command:

kubectl label node gke-cluster-1-default-pool-96cbbcce-t5sc team=platform

2. Next we need to specify the node affinity as field **nodeAffinity** of field **affinity** in the PodSpec. Have a look on below manifest:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  affinity:
    nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "team"
                operator: "Exists"
```

Exists operator will simply check if the label **"team"** exists on the nodes, and you don't need the values section for that, because it does not check the values.

3. You can apply the above manifest with command:

kubectl apply -f node-affinity.yaml

```
WKMIN7290299:kubernets jaskumar$ kubectl apply -f node-affinity.yaml
pod/nginx created
WKMIN7290299:kubernets jaskumar$ vi node-affinity.yaml
```

4. You can validate if pod scheduled on the node which meets the nodeAffinity with command:

kubectl get pod nginx -o wide

```
WKMIN7290299:kubernets jaskumar$ kubectl get pod nginx -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP         NODE                                        NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          17m   10.4.1.4   gke-cluster-1-default-pool-96cbbcce-t5sc   <none>           <none>
WKMIN7290299:kubernets jaskumar$
```

The new node affinity syntax supports the following operators: In, NotIn, Exists, DoesNotExist, Gt, Lt. You can use NotIn and DoesNotExist to achieve node anti-affinity behavior, or use **node taints** to repel pods from specific nodes.

Now suppose you are trying to create a pod by using node affinity but the node does not have any label? What will happen?

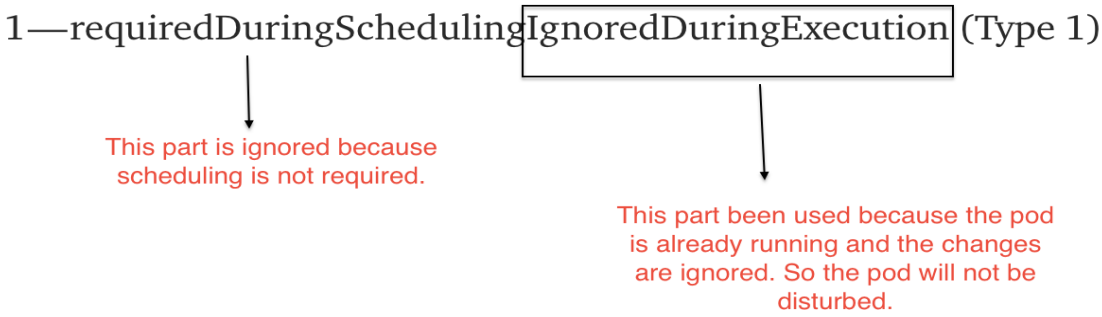
Case-1:

If you have used the above method, then the pod will not be placed on the node because the node does not have the labels.

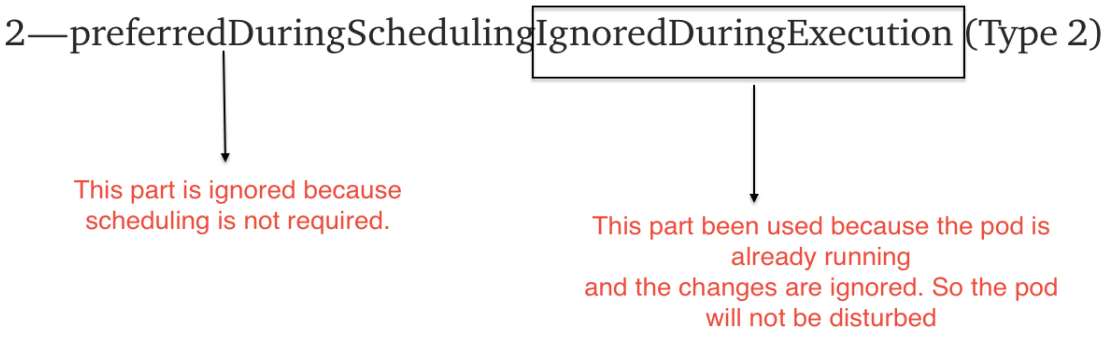
If you have used the above method, then even though the node does not have the label, the pod will be placed.

Case-2:

Assume that the given pod is running on the node and someone removed the label “team” on the node. Now, what will happen to the PODs that are running on the node?

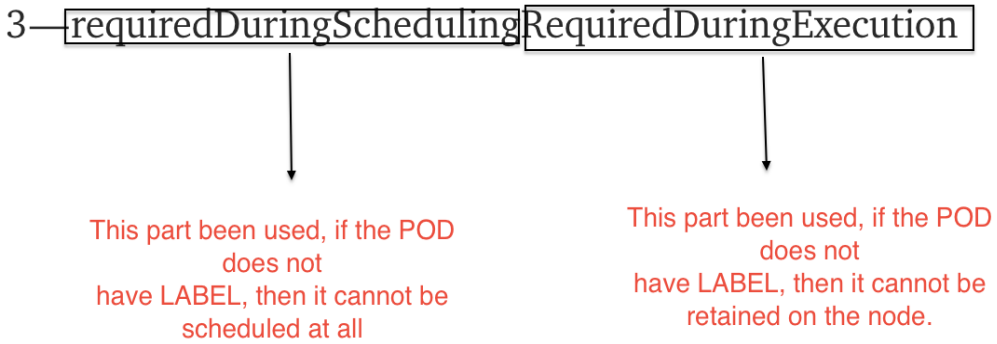


If you have used the above affinity type, then the pod will still continue to run and the new changes will not impact the pod in anyway.



If you have used the above affinity type, then the pod will still continue to run and the new changes will not impact the pod in any ways. Ideally both provide same results.
Because if you notice the second part in both type of affinity, it says **"IgnoredDuringExecution"** which means, "Pod will continue to run and any changes in node affinity will not impact them once they are scheduled".

There is one Node Affinity type which is not available right now but may come in future Kubernetes releases, please refer below type,



In this type, the pod will be removed from the node which does not have a label.

InitContainers

InitContainers are specialized containers that run before app containers in a pod. Init containers can contain utilities or setup scripts not present in an app image.

In case you have multiple InitContainers kubelet execute them in sequence and all the InitContainers should start before the pod.

initContainers does not support lifecycle, livenessProbe, readyProbe & startupProbe
Example:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  initContainers:
    - image: nginx
      name: nginx
      command: ["cat","/etc/secrets/secret-file.txt"]
      volumeMounts:
        - name: volume1
          mountPath: /etc/secrets
  volumes:
    - name: volume1
      secret:
        secretName: secret-2
  nodeSelector:
    team: platform
```

Ephemeral Containers

It is a special type of container that runs temporarily in an existing pod to accomplish user-initiated actions such as troubleshooting. You use ephemeral containers to inspect services rather than to build applications.

Ephemeral containers was introduced as alpha feature in kubernetes 1.16. One of the use case of ephemeral container is to troubleshoot your live application. Think how many times have you wished your base docker image you've built your application image on top of has curl, dig, or even ping in some cases…

As per the best practices, we've kept our docker images as slim as possible, and removed as much attack surface area as possible. This usually means all the useful diagnostic tools are missing.

Ephemeral Containers are great. We can now keep a diagnostic Docker image handy with all the tools we need and live insert a diagnostic container into a running pod to troubleshoot when the time arises.

When an Ephemeral Container runs, it executes within the namespace of the target pod. So you'll be able to access, for example, the filesystems and other resources that containers in the pods have.

By default ephemeral containers are disabled in kubernetes and you need kubernetes 1.16 or higher to enable it. You also need following two pod features:

- **EphemeralContainers** – disabled by default in 1.16 as it's alpha.
- **PodShareProcessNamespace** – for sharing the process namespace in a pod (enabled by default in 1.16 as it's a beta feature).

To enable ephemeral containers we need to add the following under **command** section:

--feature-gates=EphemeralContainers=true

in following files:

- /etc/kubernetes/manifests/kube-apiserver.yaml
- /etc/kubernetes/manifests/kube-scheduler.yaml

Let's create a new pod using manifest as shown below:

```
apiVersion: v1
```

```
kind: Pod
metadata:
  labels:
    app: my-nginx
  name: nginx
spec:
  shareProcessNamespace: true
  containers:
  - image: nginx
    name: nginx
    ports:
    - name: http
      containerPort: 80
```

Check the highlighted **shareProcessNamespace** attribute. Next create ephemeral container manifest in JSON format as the one shown below:

```
{
    "apiVersion": "v1",
    "kind": "EphemeralContainers",
    "metadata": {
            "name": "nginx"
    },
    "ephemeralContainers": [{
        "command": [
            "bash"
        ],
        "image": "debug-tools:latest",
        "imagePullPolicy": "Always",
        "name": "diagtools",
        "stdin": true,
        "tty": true,
        "terminationMessagePolicy": "File"
    }]
}
```

Apply the changes with command:

kubectl -n default replace --raw /api/v1/namespaces/default/pods/nginx/ephemeralcontainers -f ./ephemeral-diagnostic-container.json

You can validate the ephemeral container

kubectl get pod nginx

Next you can attach a container to a pod with command:

kubectl attach -it nginx -c diagtools

Remember that with Ephemeral Containers:

- Cannot have ports, so fields such as **ports**, **livenessProbe**, **readinessProbe** are not able to be used.
- Setting resources is disallowed as pod resources are immutable. They will disappear if a pod is deleted/re-scheduled.

Pause Containers

```
gke-cluster-1-default-pool-96cbbcce-t5sc /home/jaskumar # docker ps
CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS              PORTS
6e6238a7eb67        nginx                                  "/docker-entrypoint.…"   2 hours ago         Up 2 hours
b9-43a5-bd38-978f4a2ae75d_0
1c3f859d1ee1        k8s.gcr.io/pause:3.1                   "/pause"                 2 hours ago         Up 2 hours
-43a5-bd38-978f4a2ae75d_0
2065b0eee8e2        k8s.gcr.io/prometheus-to-sd            "/monitor --source=k…"   5 hours ago         Up 5 hours
d994f-t5t9n_kube-system_cbac100c-060d-4889-bfe4-44c6d6998068_0
ef5b092aa554        k8s.gcr.io/k8s-dns-sidecar-amd64       "/sidecar --v=2 --lo…"   5 hours ago         Up 5 hours
9n_kube-system_cbac100c-060d-4889-bfe4-44c6d6998068_0
15d2a77ed933        k8s.gcr.io/k8s-dns-dnsmasq-nanny-amd64 "/dnsmasq-nanny -v=2…"   5 hours ago         Up 5 hours
9n_kube-system_cbac100c-060d-4889-bfe4-44c6d6998068_0
464fa999a9a5        gke.gcr.io/prometheus-to-sd            "/monitor --stackdri…"   5 hours ago         Up 5 hours
-exporter-gke-6c9d8bd8d8-4wkz6_kube-system_1fa764fa-7e17-4787-9b8b-5e5083360994_0
359462eeb8d3        gke.gcr.io/prometheus-to-sd            "/monitor --stackdri…"   5 hours ago         Up 5 hours
td-gke-svh2m_kube-system_55711484-3cf5-4372-86a6-2d32b9057280_0
10508ff44119        k8s.gcr.io/k8s-dns-kube-dns-amd64      "/kube-dns --domain=…"   5 hours ago         Up 5 hours
9n_kube-system_cbac100c-060d-4889-bfe4-44c6d6998068_0
e41ac9fd78aa        gcr.io/gke-release/gke-metrics-agent   "/otelsvc --config=/…"   5 hours ago         Up 5 hours
gent-8tm25_kube-system_b77f0ef5-bf67-40bd-890c-400188d8b9ff_0
bed5fcc3985e        k8s.gcr.io/event-exporter              "/event-exporter '-s…"   5 hours ago         Up 5 hours
ke-6c9d8bd8d8-4wkz6_kube-system_1fa764fa-7e17-4787-9b8b-5e5083360994_0
d275c30d4c40        8c865b401705                           "/entrypoint.sh /usr…"   5 hours ago         Up 5 hours
ube-system_55711484-3cf5-4372-86a6-2d32b9057280_0
5df3f726741f        k8s.gcr.io/prometheus-to-sd            "/monitor --source=k…"   5 hours ago         Up 5 hours
d-rcv6h_kube-system_7895f016-b896-4ee1-9408-422dd313e979_0
e24f8bf6f7c         k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
m_55711484-3cf5-4372-86a6-2d32b9057280_0
8147510bf00         k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
be-system_cbac100c-060d-4889-bfe4-44c6d6998068_0
78ec950905d         k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
-system_b77f0ef5-bf67-40bd-890c-400188d8b9ff_0
b277cb87243         k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
8-4wkz6_kube-system_1fa764fa-7e17-4787-9b8b-5e5083360994_0
08a2ea1d5b1         k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
system_7895f016-b896-4ee1-9408-422dd313e979_0
f7ab19ca37d4        9fd6006e8a72                           "/bin/sh -c 'exec ku…"   5 hours ago         Up 5 hours
er-1-default-pool-96cbbcce-t5sc_kube-system_93c44356ce6cec2f898a25944d14da42_0
43b07bd31ec4        k8s.gcr.io/pause:3.1                   "/pause"                 5 hours ago         Up 5 hours
```

Static Pods

Static Pods are managed directly by the kubelet daemon on a specific node, without the **API server** observing them. Unlike Pods that are managed by the control plane; instead, the kubelet watches each static Pod (and restarts it if it crashes).

Static Pods are always bound to one **Kubelet** on a specific node.

The kubelet automatically tries to create a **mirror pod** on the Kubernetes API server for each static Pod. This means that the Pods running on a node are visible on the API server, but cannot be controlled from there.

To create a static pod follow under mentioned steps:
1. Login to specific node or master on which you want to create static pod.
2. By default kubelet read the **/etc/kubernetes/manifests** directory
3. Create pod template file with some meaningful name under **/etc/kubernetes/manifests**
4. Restart kubelet service with command

   service kubelet restart

Other than this you can also create a directory of your choice say /etc/kubelet.d

Create template file under **/etc/kubelet.d** directory and add **--pod-manifest-path** in **/etc/kubernetes/kubelet** service

Finally restart kubelet service with command:

service kubelet restart

I have create a file at location **/etc/kubernetes/manifests/nginx-static-pod.yaml** file with below contents:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-pod
spec:
  containers:
    - name: nginx-static
      image: nginx
```

Now when you run **kubectl get pods** command you will see the below output:

```
WKMIN7290299:jkumar jaskumar$ kubectl get pods
NAME                                                  READY   STATUS             RESTARTS   AGE
nginx                                                 0/1     CrashLoopBackOff   194        16h
nginx-static-pod-gke-cluster-1-default-pool-d74ac0e8-86sn   1/1     Running            0          19s
```

Another option is to validate the **/var/lib/kubelet/config.yaml** file on the node on which to create the static pod for key **staticPodPath**. In case it is defined we need to get its value. Usually it's value is **/etc/kubernetes/manifests**. In case it is not defined we can set it's value to a directory of our choice. You can also set it's value to **/etc/kubernetes/manifests**. We just need to create the pod template file in this directory and restart the kubelet service.

Volumes

On-disk files in a Container are ephemeral, which presents some problems for non-trivial applications when running in Containers. First, when a Container crashes, kubelet will restart it, but the files will be lost - the Container starts with a clean state. Second, when running Containers together in a Pod it is often necessary to share files between those Containers. The Kubernetes Volume abstraction solves both of these problems.

An **emptyDir** volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node. As the name says, it is initially empty. Containers in the Pod can all read and write the same files in the **emptyDir** volume, though that volume can be mounted at the same or different paths in each Container. When a Pod is removed from a node for any reason, the data in the emptyDir is deleted forever.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      volumeMounts:
        - name: volume1
          mountPath: /tmp/nginx
  volumes:
    - name: volume1
      emptyDir: {}
```

hostPath volume type

A hostPath volume mounts a file or directory from the host node's filesystem into your Pod. This is not something that most Pods will need, but it offers a powerful escape hatch for some applications.

For example, some uses for a hostPath are:

```
apiVersion: v1
```

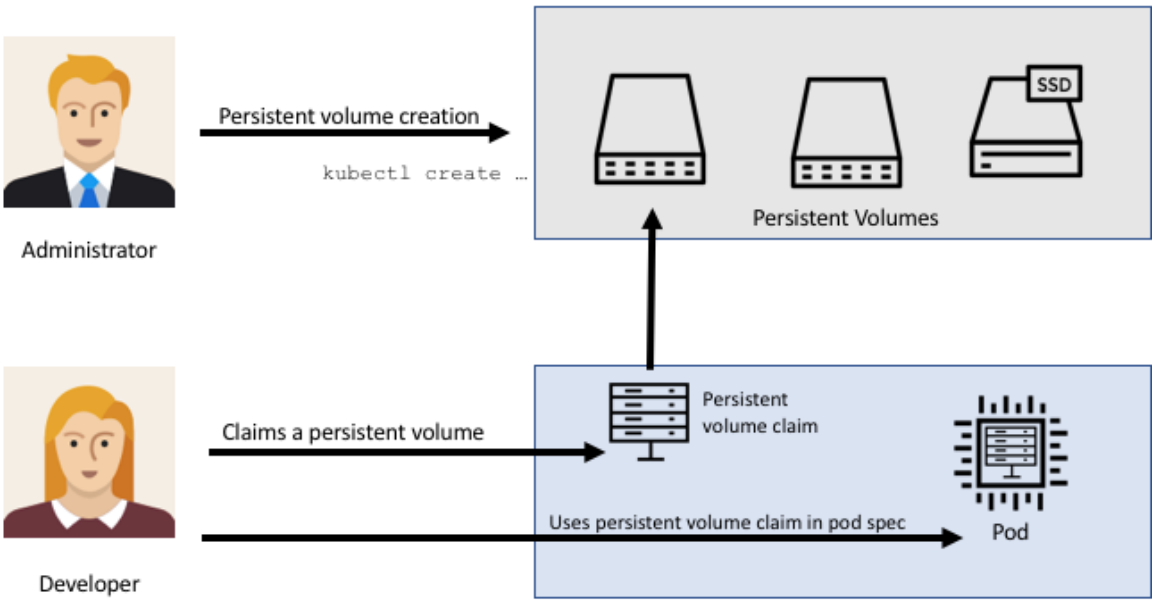
```yaml
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
      volumeMounts:
        - name: volume1
          mountPath: /tmp/nginx
  volumes:
    - name: volume1
      hostPath:
        path: /etc/nginx
        type: Directory
```

Persistent Volume

Kubernetes persistent volumes are administrator provisioned volumes. These are created with a particular filesystem, size, and identifying characteristics such as volume IDs and names.

A Kubernetes persistent volume has the following attributes

- It is provisioned either dynamically or by an administrator
- Created with a particular filesystem
- Has a particular size
- Has identifying characteristics such as volume IDs and a name



Kubernetes Volumes vs Persistent Volumes

There are currently two types of storage abstracts available with Kubernetes: Volumes and Persistent Volumes. A Kubernetes volume exists only while the containing pod exists. Once the pod is deleted, the associated volume is also deleted. As a result, Kubernetes volumes are useful for storing temporary data that does not need to exist outside of the pod's lifecycle. Kubernetes persistent volumes remain available outside of the pod lifecycle – this means that the volume will remain even after the pod is deleted. It is available to claim by another pod if required, and the data is retained.

You can use the below syntax to create the persistent volume:

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: pv3
spec:
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/opt/volume"
```

You can also attach a label to a persistent volume with command:

kubectl label pv pv-2 app=nginx

Now you can list all persistent volumes with attached labels with command:

kubectl get pv --show-labels

```
WKMIN7290299:IAC jaskumar$ kubectl get pv --show-labels
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE   LABELS
pv1    2Gi        RWO            Retain           Available                                   94m   <none>
pv2    1Gi        RWO            Retain           Available                                   93m   app=nginx
pv3    3Gi        RWO            Retain           Available                                   93m   <none>
WKMIN7290299:IAC jaskumar$
```

Persistent Volume Claim

A *PersistentVolumeClaim* (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes.

This is how the persistentVolumeClaim looks like:

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

You can consume claim as a volume in pod. This is shown below:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
      - mountPath: "/data"
        name: volume-1
  volumes:
    - name: volume-1
      persistentVolumeClaim:
        claimName: task-pv-volume
```

DaemonSet

A *DaemonSet* ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a cluster storage daemon on every node
- running a logs collection daemon on every node
- running a node monitoring daemon on every node

Since kubernetes 1.16 DaemonSets by default are not scheduled on master node. DaemonSets are similar to Deployments. You can create a daemonset using the template shown below:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: ckads
  name: cka-daemonset
spec:
  selector:
    matchLabels:
      app: ckads
  template:
    metadata:
      labels:
        app: ckads
    spec:
      containers:
      - image: nginx
        name: cka-daemonset
```

Where we are trying to run nginx as a daemonset on all the nodes. You can list all the daemonsets with command:

kubectl get daemonsets

```
WKMIN7290299:IAC jaskumar$ kubectl get daemonsets -o wide
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE   CONTAINERS      IMAGES   SELECTOR
cka-daemonset   3         3         3       3            3           <none>          79m   cka-daemonset   nginx    app=ckads
WKMIN7290299:IAC jaskumar$
```

To delete a daemonset run the command:

kubectl delete daemonset cka-daemonset

In case we need to run daemonset on master node as well we have to use the below configuration:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: ckads
  name: cka-daemonset
spec:
  selector:
    matchLabels:
      app: ckads
  template:
    metadata:
      labels:
        app: ckads
    spec:
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
          operator: "Exists"
      containers:
      - image: nginx
        name: cka-daemonset
```

We need to add the selected stuff in case we have to schedule it on master node.

Deployment

A *Deployment* provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

You can use this command to create the deployment:

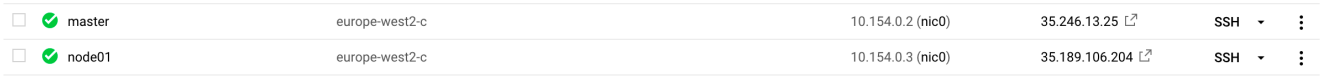
kubectl create deployment cka-deployment --image=nginx:1.16.1 > /opt/D407/cka-dep.yaml

Now you can edit the file **/opt/D407/cka-dep.yaml** file with the needed labels and replicas. This is how the template syntax looks like:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cka-dep
  name: cka-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cka-dep
  strategy: {}
  template:
    metadata:
      labels:
        app: cka-dep
    spec:
      containers:
      - image: nginx:1.16.1
        name: nginx
```

Cluster Installation – kubeadm way

Though you will get two or more nodes already spin-up in exam and you have to be asked to setup kubernetes cluster using kubeadm approach. To practice the same I have spin-up two nodes on GCP having **ubuntu 16.04** as shown below:



To setup master I have followed under mentioned steps:

1. Login to master node with command:

   gcloud beta compute ssh --zone "europe-west2-c" "master" --project "bamboo-strata-280313"

2. Make sure that the **br_netfilter** module is loaded. You can validate this with command:

   lsmod | grep br_netfilter

3. In case #2 did not return anything we have to load the module with command:

   modprobe br_netfilter

   You can again validate it with same command mentioned in #2 and the output is:

```
root@master:~# modprobe br_netfilter
root@master:~# lsmod | grep br_netfilter
br_netfilter           24576  0
bridge                155648  1 br_netfilter
root@master:~#
```

4. Next we will run the below commands:

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

```
root@master:~# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
root@master:~# sudo sysctl --system
* Applying /etc/sysctl.d/10-console-messages.conf ...
kernel.printk = 4 4 1 7
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
kernel.kptr_restrict = 1
* Applying /etc/sysctl.d/10-link-restrictions.conf ...
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

5. Next install container runtime. There are multiple options available for that but we will install docker as container runtime. For all the steps refer: https://kubernetes.io/docs/setup/production-environment/container-runtimes/

Ubuntu 16.04+ | CentOS/RHEL 7.4+

```
# (Install Docker CE)
## Set up the repository:
### Install packages to allow apt to use a repository over HTTPS
apt-get update && apt-get install -y \
  apt-transport-https ca-certificates curl software-properties-common gnupg2
```

```
# Add Docker's official GPG key:
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
```

```
# Add the Docker apt repository:
add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

```
# Install Docker CE
apt-get update && apt-get install -y \
  containerd.io=1.2.13-2 \
  docker-ce=5:19.03.11~3-0~ubuntu-$(lsb_release -cs) \
  docker-ce-cli=5:19.03.11~3-0~ubuntu-$(lsb_release -cs)
```

6. Validate docker service status with command:

   service docker status

7. Next run the following commands to install kubeadm, kubelet and kubectl:

   ```
   sudo apt-get update && sudo apt-get install -y apt-transport-https curl
   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
   cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
   deb https://apt.kubernetes.io/ kubernetes-xenial main
   EOF
   sudo apt-get update
   sudo apt-get install -y kubelet kubeadm kubectl
   sudo apt-mark hold kubelet kubeadm kubectl
   ```

8. Next initialize the cluster with kubeadm init command. This needs to be performed on master node not on worked nodes

   kubeadm init --pod-network-cidr=10.155.0.0/16 --apiserver-advertise-address=10.154.0.2

```
root@master:~# kubeadm init --pod-network-cidr=10.155.0.0/16 --apiserver-advertise-address=10.154.0.2
W0723 16:24:36.764600   10348 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]
[init] Using Kubernetes version: v1.18.6
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [master kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [1
.2]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [master localhost] and IPs [10.154.0.2 127.0.0.1 ::1]


Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.154.0.2:6443 --token ckynfb.5s01e5hwxmte1gga \
    --discovery-token-ca-cert-hash sha256:09756cf8f6250dcf1f2b673e5a7054de5eece4e50d45c833973afc77a7e535d3
```

If look at the message in above screenshot kubectl is asking to switch to regular user and run the below commands:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

9. Save the below output somewhere as it is needed to join the worker nodes:

```
kubeadm join 10.154.0.2:6443 --token ckynfb.5s01e5hwxmte1gga \
    --discovery-token-ca-cert-hash
  sha256:09756cf8f6250dcf1f2b673e5a7054de5eece4e50d45c833973afc
  77a7e535d3
```

10. Install autocomplete command on master:

```
sudo apt-get install bash-completion

echo 'source <(kubectl completion bash)' >>~/.bashrc

source ~/.bashrc
```

11. Run kubectl get nodes command on master node

```
jaskumar@master:~$ kubectl get nodes
NAME      STATUS     ROLES    AGE     VERSION
master    NotReady   master   6m25s   v1.18.6
jaskumar@master:~$
```

12. Install pod network add-on. We are going with weave-net option.

```
kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=$(kubectl
version | base64 | tr -d '\n')
```

13. Next run kubectl get nodes command again:

```
jaskumar@master:~$ kubectl get nodes
NAME      STATUS   ROLES    AGE   VERSION
master    Ready    master   12m   v1.18.6
jaskumar@master:~$
```

14. Follow steps #1 to #7 on worker node.

15. Finally run the below command on worker node to join the master node:

```
kubeadm join 10.154.0.2:6443 --token ckynfb.5s01e5hwxmte1gga \
    --discovery-token-ca-cert-hash
sha256:09756cf8f6250dcf1f2b673e5a7054de5eece4e50d45c833973afc77
a7e535d3
```

```
root@node01:~# kubeadm join 10.154.0.2:6443 --token ckynfb.5s01e5hwxmte1gga \
>     --discovery-token-ca-cert-hash sha256:09756cf8f6250dcf1f2b673e5a7054de5eece4e50d45c833973afc77a7e535d3
W0723 16:43:24.889350    8763 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane settings will be ignored when control-plane flag is not set.
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

16. Finally go to master node and run the command:

    kubectl get nodes

```
root@master:~# kubectl get nodes
NAME     STATUS   ROLES    AGE   VERSION
master   Ready    master   19m   v1.18.6
node01   Ready    <none>   93s   v1.18.6
root@master:~#
root@master:~#
root@master:~#
```

17. Validate pods running on master with command:

    kubectl get pods -n kube-system

```
root@master:~# kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-66bff467f8-ddkxq            1/1     Running   0          20m
coredns-66bff467f8-xnq2d            1/1     Running   0          20m
etcd-master                         1/1     Running   0          20m
kube-apiserver-master               1/1     Running   0          20m
kube-controller-manager-master      1/1     Running   0          20m
kube-proxy-8v4sd                    1/1     Running   0          20m
kube-proxy-mlkj2                    1/1     Running   0          2m28s
kube-scheduler-master               1/1     Running   0          20m
weave-net-44gb4                     2/2     Running   0          9m56s
weave-net-7jmsj                     2/2     Running   1          2m28s
root@master:~#
```

JSONPATH

At least we should practice the following jsonpath queries:

1. Build jsonpath query to fetch node names

   kubectl get nodes -o jsonpath={.items[*].metadata.name}

2. Build jsonpath query to fetch node OS image

   kubectl get nodes -o jsonpath='{.items[*].status.nodeInfo.osImage}'

3. Build jsonpath query to fetch user names from kube config file. You can take your own kube config file example.

   kubectl config view --kubeconfig=/root/my-kube-config -o=jsonpath={.users[*].name}

4. Build jsonpath query to sort the persistent volumes on the basis of capacity

   kubectl get pv --sort-by={.spec.capacity.storage} --no-headers -o=custom-columns='NAME:metadata.name, CAPACITY:spec.capacity.storage'

5. Build jsonpath query to sort the persistent volumes on the basis of capacity and store as NAME and CAPACITY using custom-columns

   kubectl get pv --sort-by={.spec.capacity.storage} -o=custom-columns='NAME:metadata.name,CAPACITY:spec.capacity.storage'

6. Build jsonpath query to fetch the context name for user **aws-user**

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

7. Build a jsonpath query to fetch the DEPLOYMENT NAME, CONTAINER IMAGE, READY REPLICAS and NAMESPACE. It should be displayed as shown below:

   DEPLOYMENT CONTAINER_IMAGE READY_REPLICAS NAMESPACE

   Also sort it on the basis of deployment name and use custom-columns.

```
kubectl get deploy -n admin2406 --sort-by={.metadata.name} -o=custom-
columns=DEPLOYMENT:.metadata.name,CONTAINER_IMAGE:.spec.templ
ate.spec.containers[*].image,READY_REPLICAS:.spec.replicas,NAMESPAC
E:.metadata.namespace
```

ETCD Backup & Restore

To take etcd backup follow under mentioned steps:

1. Login to kubernetes master node
2. Set environment variable

   export ETCDCTL_API=3

3. Get the details of cert, key, cacert and endpoints from

   /etc/kubernetes/manifests/etcd.yaml

4. Suppose you want to create the backup at location /var/lib/etcd/etcd-backup
5. Run command to initiate the snapshot

   etcdctl snapshot save /tmp/snapshot.db –cert=<cert_file> --key=<key_file> --cacert=<ca_file> --endpoints=<endPoint>

Upgrade Cluster – Master & Worker Nodes

To upgrade the cluster note the following:

- You can upgrade only one major version at a time
- Upgrade master node first
- Version number should exactly match with the version number asked in exam

To upgrade the cluster follow under mentioned steps:

1. Run cluster upgrade plan command:

   kubeadm upgrade plan

```
master $ kubeadm upgrade plan
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.16.0
[upgrade/versions] kubeadm version: v1.17.0
I0725 06:19:30.237653    8523 version.go:251] remote version is much newer: v1.18.6; falling back to: stable-1.17
[upgrade/versions] Latest stable version: v1.17.9
[upgrade/versions] Latest version in the v1.16 series: v1.16.13
```

2. Drain the master node for maintenance with command:

   kubectl drain master --igonore-daemonsets

3. Next upgrade kubeadm (one major version at a time). Suppose we need to upgrade cluster from 1.16.0 to 1.18.0. We first need to upgrade kubeadm from 1.16.0 to 1.17.0 with command:

   apt-get upgrade -y kubeadm=1.17.0-00

4. Upgrade the cluster with command:

   kubeadm upgrade apply 1.17.0

5. Next upgrade the kubelet with same version number:

   apt-get upgrade -y kubelet=1.17.0-00

6. Restart kubelet service

   service kubelet restart

Same steps with need to follow to upgrade to 1.18.0

Questions & Solutions

Question-1: print persistent volume names in sorting order with capacity and write the output to the file /opt/D407/pv.txt

To sort persistent volume names on the basis of capacity follow under mentioned steps:

1. List all persistent volumes with command:

   kubectl get pv

```
WKMIN7290299:IAC jaskumar$ kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv1    2Gi        RWO            Retain           Available                                   100m
pv2    1Gi        RWO            Retain           Available                                   98m
pv3    3Gi        RWO            Retain           Available                                   98m
WKMIN7290299:IAC jaskumar$
```

2. Run the below command:

   kubectl get pv --no-headers --sort-by={.spec.capacity.storage}

```
WKMIN7290299:IAC jaskumar$ kubectl get pv --no-headers --sort-by={.spec.capacity.storage}
pv2   1Gi   RWO   Retain   Available                   109m
pv1   2Gi   RWO   Retain   Available                   110m
pv3   3Gi   RWO   Retain   Available                   108m
WKMIN7290299:IAC jaskumar$
```

3. Now to fetch only names as per capacity sorting run command:

   kubectl get pv --no-headers --sort-by={.spec.capacity.storage} -o=custom-columns='NAME:metadata.name'

```
WKMIN7290299:IAC jaskumar$ kubectl get pv --no-headers --sort-by={.spec.capacity.storage} -o=custom-columns='NAME:metadata.name'
pv2
pv1
pv3
WKMIN7290299:IAC jaskumar$
```

4. Finally redirect the output in file /opt/D407/pv.txt

   kubectl get pv --no-headers --sort-by={.spec.capacity.storage} -o=custom-columns='NAME:metadata.name' > /opt/D407/pv.txt

5. Validate the file contents with command:
   cat /opt/D407/pv.txt

Question-2: grep the specific error <error> from container logs and redirect to file /opt/D407/container.txt

To redirect specific error from the container logs follow the steps:

1. Fetch the logs from container with command:

   kubectl logs <pod_name> -c <container_name> | grep '<error_keyword>' > /opt/D407/container.txt

2. Validate the log contents

   cat /opt/D407/container.txt

Question-3: create a persistent volume with hostPath /mnt/data, capacity 2Gi and access mode **ReadWriteOnce**.

To create persistent volume with hostPath /mnt/data, capacity 2Gi and access mode **ReadWriteOnce** follow under mentioned steps:

1. Create persistent volume using below template:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: persistent
spec:
  accessMode:
    - ReadWriteOnce
  capacity:
    storage: 2Gi
  hostPath:
      path: /mnt/data
```

2. You need to replace the persistent volume name, capacity, access mode etc as per exam
3. Copy & Paste the step 1 contents in file **pv-latest.yaml**
4. Apply the changes with command

   kubectl apply -f pv.yaml

```
WKMIN7290299:IAC jaskumar$ kubectl apply -f pv-latest.yaml
persistentvolume/persistent created
WKMIN7290299:IAC jaskumar$
WKMIN7290299:IAC jaskumar$
```

5. You can validate the persistent volume with command:

kubectl get pv

```
WKMIN7290299:IAC jaskumar$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
persistent   2Gi        RWO            Retain           Available                                   66s
pv1          2Gi        RWO            Retain           Available                                   171m
pv2          1Gi        RWO            Retain           Available                                   170m
pv3          3Gi        RWO            Retain           Available                                   170m
WKMIN7290299:IAC jaskumar$
```

6. Describe the persistent volume to validate the hostPath with command:

kubectl describe pv persistent

```
WKMIN7290299:IAC jaskumar$ kubectl describe pv persistent
Name:            persistent
Labels:          <none>
Annotations:     kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"PersistentVolume","metadata":{"annotations":{},"name":"persistent"},"spec":{"accessModes":["ReadWriteOnce"],"ca...
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:
Status:          Available
Claim:
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        2Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/data
    HostPathType:
Events:            <none>
WKMIN7290299:IAC jaskumar$
```

Question-4: create a daemonset with name **cka-daemonset** that runs on all the nodes except master node. It should have the label **app=ckads** attached and should be based on nginx image. Once the dammonset is created, copy the daemonset.yaml at /opt/D407/ds_q2.yaml file

The DaemonSet by default is not created on master node. We can follow under mentioned steps to create a daemonset as per the request:

1. Create a daemonset template file /opt/D407/ds_q2.yaml with below contents:

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
```

```
      app: ckads
    name: cka-daemonset
  spec:
    selector:
      matchLabels:
        app: ckads
    template:
      metadata:
        labels:
          app: ckads
      spec:
        containers:
        - image: nginx
          name: cka-daemonset
```

2. Create the daemonset with command:

   kubectl apply -f /opt/D407/ds_q2.yaml

3. Check if daemonset created with command:

   kubectl get daemonset

```
WKMIN7290299:IAC jaskumar$ kubectl get daemonset
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
cka-daemonset   3         3         3       3            3           <none>          107m
```

4. You can also validate on how many nodes it get scheduled with command:

   kubectl describe daemonset cka-daemonset

```
WKMIN7290299:IAC jaskumar$ kubectl describe daemonset cka-daemonset
Name:           cka-daemonset
Selector:       app=ckads
Node-Selector:  <none>
Labels:         app=ckads
Annotations:    deprecated.daemonset.template.generation: 2
                kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"apps/v1","kind":"DaemonSet","metadata":{"annotations":{},"labels":{"app":"ckads"},"name":"cka-daemonset","namespace":"dev"}...
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=ckads
  Containers:
   cka-daemonset:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted pod: cka-daemonset-slqct
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: cka-daemonset-gs7hr
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted pod: cka-daemonset-72r7n
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: cka-daemonset-ntb67
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted pod: cka-daemonset-vtsh8
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: cka-daemonset-dtgw9
WKMIN7290299:IAC jaskumar$
```

Question-5: create a deployment with name **cka-deployment** having 2 replicas based on the image **nginx:1.16.1** and use the labels **app=ckadep** to select the pods.

once the deployment is created, copy the deployment template to **/opt/D407/cka-dep.yaml**

Next without modifying the deployment spec update the image of the container to **nginx:1.17.4**.Then rollback the deployment to previous version.

To create the deployment with name **cka-deployment** and replicas 2 using **nginx:1.16.1** image and having label **app=ckadep** follow under mentioned steps:

1. Run the deployment in dry-run mode with command:

   kubectl create deployment cka-deployment --image=nginx:1.16.1 --dry-run -o yaml > cka-dep.yaml

   It will create the following template:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: cka-deployment
  name: cka-deployment
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cka-deployment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: cka-deployment
    spec:
      containers:
      - image: nginx:1.16.1
        name: nginx
        resources: {}
  status: {}
```

2. In this deployment file change label to **ckadep** and **replicas:2**. So the final template should look like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: ckadep
  name: cka-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ckadep
  template:
    metadata:
      labels:
        app: ckadep
    spec:
      containers:
      - image: nginx:1.16.1
        name: nginx
```

3. Apply these changes with command:

   kubectl apply -f cka-dep.yaml

4. Validate the deployment with command:

   kubectl get deploy

```
WKMIN7290299:IAC jaskumar$ kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
cka-deployment   2/2     2            2           60s
```

5. Validate pods under the deployment are running

   kubectl get pods

```
WKMIN7290299:IAC jaskumar$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
cka-daemonset-dtgw9               1/1     Running   0          10h
cka-daemonset-gs7hr               1/1     Running   0          10h
cka-daemonset-ntb67               1/1     Running   0          10h
cka-deployment-567665bf8d-7fjdm   1/1     Running   0          68s
cka-deployment-567665bf8d-wcdkt   1/1     Running   0          68s
WKMIN7290299:IAC jaskumar$
```

6. Validate pods are using image nginx:1.16.1 image and label app=ckadep is applied

   kubectl describe pod cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl describe pod cka-deployment
Name:           cka-deployment-567665bf8d-7fjdm
Namespace:      dev
Priority:       0
Node:           gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:     Thu, 16 Jul 2020 08:11:28 +0530
Labels:         app=ckadep
                pod-template-hash=567665bf8d
Annotations:    <none>
Status:         Running
IP:             10.4.1.12
Controlled By:  ReplicaSet/cka-deployment-567665bf8d
Containers:
  nginx:
    Container ID:   docker://90c6fb9da769b70f0bc15fde89d38db638f4024edb46dbd2c30156fc3a49fcd6
    Image:          nginx:1.16.1
    Image ID:       docker-pullable://nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf30dd9c
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 08:11:36 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
```

7. Now copy **cka-dep.yaml** at **/opt/D407/cka-dep.yaml**
8. We next have to update the nginx image to nginx:1.17.4 without modifying the template file. For this we can use the below command:

   kubectl set image deployments/cka-deployment nginx=nginx:1.17.4

9. Validate the deployment and containers are using the nginx:1.17.4 image

   kubectl describe pod cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl describe pod cka-deployment
Name:           cka-deployment-85c548f67b-wp5rq
Namespace:      dev
Priority:       0
Node:           gke-cluster-1-default-pool-d74ac0e8-c8cq/10.128.0.2
Start Time:     Thu, 16 Jul 2020 08:21:09 +0530
Labels:         app=ckadep
                pod-template-hash=85c548f67b
Annotations:    <none>
Status:         Running
IP:             10.4.0.10
Controlled By:  ReplicaSet/cka-deployment-85c548f67b
Containers:
  nginx:
    Container ID:   docker://ce8b0568dccb290fa68d919466c0e860a9d64dcc510678279a0ad9b9a8e866f4
    Image:          nginx:1.17.4
    Image ID:       docker-pullable://nginx@sha256:77ebc94e0cec30b20f9056bac1066b09fbdc049401b71850922c63fc0cc1762e
    Port:           <none>
    Host Port:      <none>
```

10. Validate containers are also running

    kubectl get pods

```
WKMIN7290299:IAC jaskumar$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
cka-daemonset-dtgw9                 1/1     Running   0          10h
cka-daemonset-gs7hr                 1/1     Running   0          10h
cka-daemonset-ntb67                 1/1     Running   0          10h
cka-deployment-85c548f67b-wp5rq     1/1     Running   0          2m9s
cka-deployment-85c548f67b-x5qqq     1/1     Running   0          2m16s
```

11. Check how many revisions are deployed with command:

kubectl rollout history deployment cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl rollout history deployment/cka-deployment
deployment.apps/cka-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

12. Next we need to rollback the image back to **nginx:1.16.1** with command

kubectl rollout undo deployment cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl rollout undo deployment/cka-deployment
deployment.apps/cka-deployment rolled back
WKMIN7290299:IAC jaskumar$
```

You can also rollback to a specific revision with command:

kubectl rollout undo deployment cka-deployment --to-revision=1

13. Validate the number of revisions with command:

kubectl rollout history deployment cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl rollout history deployment/cka-deployment
deployment.apps/cka-deployment
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

14. Validate the pods to make sure it is reverted back to **nginx:1.16.1** image

kubectl describe pod cka-deployment

```
WKMIN7290299:IAC jaskumar$ kubectl describe pod cka-deployment
Name:           cka-deployment-567665bf8d-2kjr9
Namespace:      dev
Priority:       0
Node:           gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:     Thu, 16 Jul 2020 08:32:39 +0530
Labels:         app=ckadep
                pod-template-hash=567665bf8d
Annotations:    <none>
Status:         Running
IP:             10.4.1.13
Controlled By:  ReplicaSet/cka-deployment-567665bf8d
Containers:
  nginx:
    Container ID:   docker://ad2d252c692ab71370519511237f967d8408192dc3e52821b7b5fd0ee34ef018
    Image:          nginx:1.16.1
    Image ID:       docker-pullable://nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf30dd9c
    Port:           <none>
    Host Port:      <none>
```

Question-6: Node say node-01 is going through the maintenance, please evacuate all the pods from this node

In case any node is going for maintenance we have to remove it from the list of available nodes and should not be available to schedule the pods.

To achieve this we have to use the **drain** command as mentioned below:

kubectl drain node01 --ignore-daemonsets

Question-7: Create multi-pod with name cka-multipod and using the following images:
- Nginx
- Redis
- Memcached

Once the pod is created, put the spec into the file **/opt/D407/cka-multipod.yaml**

Ok, so to answer this question follow under mentioned steps:

1. Create a pod template with command:

kubectl run --generator=run-pod/v1 cka-multipod --image=nginx --dry-run -o yaml > multipod.yaml

2. Modify multipod.yaml to add remaining two images. It should look like this:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: cka-multipod
```

```
    name: cka-multipod
  spec:
    containers:
    - image: nginx
      name: nginx
    - image: redis
      name: redis
    - image: memcached
        name: Memcached
```

3. Apply the template with command

   kubectl apply -f multipod.yaml

4. Validate the pod with command:

   kubectl get pods

```
WKMIN7290299:IAC jaskumar$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
cka-daemonset-dtgw9                 1/1     Running   0          11h
cka-daemonset-gs7hr                 1/1     Running   0          11h
cka-daemonset-ntb67                 1/1     Running   0          11h
cka-deployment-567665bf8d-2kjr9     1/1     Running   0          34m
cka-deployment-567665bf8d-6zbkd     1/1     Running   0          34m
cka-multipod                        3/3     Running   0          11s
WKMIN7290299:IAC jaskumar$
```

5. Next describe the pod to see the containers and images used

   kubectl describe pod cka-multipod

```
WKMIN7290299:IAC jaskumar$ kubectl describe pod cka-multipod
Name:         cka-multipod
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-86sn/10.128.0.4
Start Time:   Thu, 16 Jul 2020 09:07:11 +0530
Labels:       run=cka-multipod
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"run":"cka-multipod"},"name":"cka-multi
Status:       Running
IP:           10.4.2.7
Containers:
  nginx:
    Container ID:   docker://63d86c63e7c4cecf5d91e57e2f211939b0ffc35224cafefbfdd9590ca9a4d50e
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:8ff4598873f588ca9d2bf1be51bdb117ec8f56cdfd5a81b5bb0224a61565aa49
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 09:07:12 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
  redis:
    Container ID:   docker://38e579dcb02390c66540ae9d8f8934abe653281945df7c2153c8776aade448f7
    Image:          redis
    Image ID:       docker-pullable://redis@sha256:76ff608805ca40008d6e0f08180d634732d8bf4728b85c18ab9bdbfa0911408d
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 09:07:14 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
  memcached:
    Container ID:   docker://d21433a5418b00155aa8b3603eb0663cdf70bcaa357c2d778e0a61e21674084c
    Image:          memcached
    Image ID:       docker-pullable://memcached@sha256:4883efd6cfc85109bebdbca6e121dc2aa8095f5157f28407f86a2e5c2451bed6
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 09:07:15 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
```

Question-8: Create a pod manifest with pod name pod09301130. The pod should be created in default namespace. Use hostPath type volume mounted on /tmp/data on the host and mount this volume inside the init container under /datainit. The name of initContainer should be the init continitpod09301130. The init container should create a file "test_file" in /datainit directory.

Also create a main container inside the pod using "busybox:1.31" image and this container should run the command "sleep 3600" only if the file "test_file" created by init container should exists.

To solve this question follow under mentioned steps:

1. Create pod template with command:

   kubectl run --generator=run-pod/v1 pod09301130 --image=busybox:1.31 --dry-run -o yaml > pod09301130.yaml

2. Modify template **pod09301130.yaml** to incorporate the ask as per exam. This is how it looks like:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod09301130
  name: pod09301130
spec:
  initContainers:
    - image: busybox:1.31
      name: initpod09301130
      command: ["/bin/sh","-c","touch
/datainit/test_file"]
      volumeMounts:
        - name: volume1
          mountPath: /datainit
  containers:
    - image: busybox:1.31
      name: mainpod09301130
      command: ["/bin/sh","-c","test -f
/datainit/test_file && sleep 3600"]
      volumeMounts:
        - name: volume1
          mountPath: /datainit
  volumes:
    - name: volume1
      hostPath:
          path: /tmp/data
```

3. Apply the changes with command:

   kubectl apply -f pod09301130.yaml

4. Validate the pod with command:

   kubectl describe pod pod09301130

```
Events:
  Type    Reason     Age   From                                                  Message
  ----    ------     ----  ----                                                  -------
  Normal  Scheduled  7s    default-scheduler                                     Successfully assigned dev/pod09301130 to gke-cluster-1-default-pool-d74ac0e8-c8cq
  Normal  Pulled     6s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Container image "busybox:1.31" already present on machine
  Normal  Created    6s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Created container initpod09301130
  Normal  Started    6s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Started container initpod09301130
  Normal  Pulled     5s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Container image "busybox:1.31" already present on machine
  Normal  Created    5s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Created container mainpod09301130
  Normal  Started    5s    kubelet, gke-cluster-1-default-pool-d74ac0e8-c8cq  Started container mainpod09301130
```

Question-9: Create a Secret with name **super-secret** with secret **Username: abc** and perform the following:

- Run the first pod in which mount the secret to the empty dir volume
- Run the second pod in which mount the secret as a env variable with name CONFIDENTIAL

To solve the question follow under mentioned steps:

1. Encrypt the secret with base64 as shown in below command:

   echo -n "abc" | base64

   ```
   WKMIN7290299:kubernets jaskumar$ echo -n "abc" | base64
   YWJj
   ```

2. Create a secret with command:

   kubectl create secret generic super-secret --from-literal=Username=YWJj --dry-run -o yaml > super-secret.yaml

   This is how the template look like:

   ```
   apiVersion: v1
   data:
     Username: WVdKag==
   kind: Secret
   metadata:
       name: super-secret
   ```

3. Now apply the secret with command:

   kubectl apply -f super-secret.yaml

4. Validate the secret with command:

   kubectl describe secret super-secret

```
WKMIN7290299:kubernets jaskumar$ kubectl describe secret super-secret
Name:         super-secret
Namespace:    dev
Labels:       <none>
Annotations:
Type:         Opaque

Data
====
Username:  4 bytes
```

5. Let's mount this secret in redis pod as empty directory. We will create a pod template with command:

   kubectl run --generator=run-pod/v1 redis --image=redis --dry-run -o yaml > redis.yaml

6. Modify **redis.yaml** to look like the one below:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: redis
  name: redis
spec:
  containers:
  - image: redis
    name: redis
    volumeMounts:
      - name: volume1
        mountPath: /etc/redis
  volumes:
    - name: volume1
      secret:
          secretName: super-secret
```

7. Describe the pod to validate this

   kubectl describe pod redis

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pod redis
Name:         redis
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Thu, 16 Jul 2020 13:52:04 +0530
Labels:       run=redis
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"run":"redis"},"name":"redis","n
Status:       Running
IP:           10.4.1.15
Containers:
  redis:
    Container ID:   docker://436db5557cc7240a2aab72e26f3de4ee487f75f0aa5309330b2d1a0c7e274f06
    Image:          redis
    Image ID:       docker-pullable://redis@sha256:76ff608805ca40008d6e0f08180d634732d8bf4728b85c18ab9bdbfa0911408d
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 13:52:06 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /etc/redis from volume1 (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  volume1:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  super-secret
    Optional:    false
```

8. Next create another pod where we need to map secret as an environment variable with name **CONFIDENTIAL**. We can use the previous template with slight modification. This is how the template looks like:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: redis
  name: redis1
spec:
  containers:
  - image: redis
    name: redis
    env:
      - name: CONFIDENTIAL
        valueFrom:
          secretKeyRef:
            name: super-secret
              key: Username
```

9. Apply the template with command:

kubectl apply -f redis.yaml

10. You can validate the pod if environment variable is created or not with command:

kubectl describe pod redis1

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pod redis1
Name:         redis1
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Thu, 16 Jul 2020 14:03:32 +0530
Labels:       run=redis
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"run":"redis"},"name":"redis1","namespace":"de
Status:       Running
IP:           10.4.1.17
Containers:
  redis:
    Container ID:   docker://56449f6b2e1b81f04ea4c3feb6dfc4dbfbdf4045656717c49cc45193f3cfe1a4
    Image:          redis
    Image ID:       docker-pullable://redis@sha256:76ff608805ca40008d6e0f08180d634732d8bf4728b85c18ab9bdbfa0911408d
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jul 2020 14:03:33 +0530
    Ready:          True
    Restart Count:  0
    Environment:
      CONFIDENTIAL:  <set to the key 'Username' in secret 'super-secret'>  Optional: false
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

Question-10: Create a pod with name **front-end** and the pod should use the lables "**app=cka-forntend**". The pod should contain a container based on nginx latest image.

Expose this pod as a ClusterIP type service with name as **cka-frontend-service**. The service port and targetPort should be same.

To solve this question follow under mentioned steps:

1. Create a pod template with command:

kubectl run --generator=run-pod/v1 front-end --image=nginx -l=app=cka-frontend --dry-run -o yaml > front-end.yaml

2. The template should look like the one shown below:

```
apiVersion: v1
kind: Pod
metadata:
```

```
    labels:
      app: cka-frontend
    name: front-end
  spec:
    containers:
    - image: nginx
        name: front-end
```

3. Apply the changes with command:

   kubectl apply -f front-end.yaml

4. Validate the pod with command:

   kubectl get pod front-end

```
WKMIN7290299:kubernets jaskumar$ kubectl get pod front-end
NAME        READY   STATUS    RESTARTS   AGE
front-end   1/1     Running   0          58s
WKMIN7290299:kubernets jaskumar$
```

5. Next create a service with command:

   kubectl create service clusterip cka-frontend-service --port=80 --dry-run -o yaml > cka-frontend-service.yaml

6. Next modify the label name as created in pod. This is how template looks like:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cka-frontend
  name: cka-frontend-service
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: cka-frontend
```

```
  type: ClusterIP
```

7. Apply the changes with command:

   kubectl apply -f cka-frontend-service.yaml

8. Validate the service with command:

   kubectl describe svc cka-frontend-service

```
WKMIN7290299:kubernets jaskumar$ kubectl describe svc cka-frontend-service
Name:              cka-frontend-service
Namespace:         dev
Labels:            app=cka-frontend
Annotations:       cloud.google.com/neg: {"ingress":true}
                   kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"creationTimestamp":null,"labels":{"app":"cka-frontend"},"name":"cka-fron...
Selector:          app=cka-frontend
Type:              ClusterIP
IP:                10.8.13.225
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.4.0.16:80
Session Affinity:  None
Events:            <none>
```

   The endpoint IP should be the IP of the front-end pod.

Question-11: create a deployment with name **cka-deployment9** and using image nginx with 2 replicas. Expose the deployment as nodePort service with the service name as **cka-deployment-service**

To solve this follow under mentioned steps:

1. Create deployment template with command:

   kubectl create deployment cka-deployment9 --image=nginx --dry-run -o yaml > cka-deployment9.yaml

2. Modify the template **cka-deployment9.yaml** as per the requirement. It should look like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cka-deployment9
  name: cka-deployment9
spec:
  replicas: 2
  selector:
    matchLabels:
```

```
      app: cka-deployment9
  template:
    metadata:
      labels:
        app: cka-deployment9
    spec:
      containers:
      - image: nginx
        name: nginx
```

3. Run the template with command:

   kubectl apply -f cka-deployment9.yaml

4. Validate the deployment with command:

   kubectl get deploy cka-deployment9

```
WKMIN7290299:kubernets jaskumar$ kubectl get deploy cka-deployment9
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
cka-deployment9    2/2     2            2           17s
WKMIN7290299:kubernets jaskumar$
```

5. Next validate all the pods are running fine with command:

   kubectl get pods

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
cka-daemonset-dtgw9                 1/1     Running   0          17h
cka-daemonset-gs7hr                 1/1     Running   0          17h
cka-daemonset-ntb67                 1/1     Running   0          17h
cka-deployment-567665bf8d-2kjr9     1/1     Running   0          6h16m
cka-deployment-567665bf8d-6zbkd     1/1     Running   0          6h16m
cka-deployment9-547d8679f9-qzlcs    1/1     Running   0          2m5s
cka-deployment9-547d8679f9-x22wd    1/1     Running   0          2m5s
cka-multipod                        3/3     Running   0          5h42m
front-end                           1/1     Running   0          32m
pod09301130                         1/1     Running   1          82m
redis                               1/1     Running   0          57m
redis1                              1/1     Running   0          45m
```

6. Next create the service with command:

kubectl expose deployment cka-deployment9 --name=cka-deployment-service --port=80 --name=NodePort --protocol=TCP

7. Validate the service with command:

kubectl describe svc cka-deployment-service

```
WKMIN7290299:kubernets jaskumar$ kubectl describe svc cka-deployment-service
Name:                     cka-deployment-service
Namespace:                dev
Labels:                   app=cka-deployment9
Annotations:              cloud.google.com/neg: {"ingress":true}
                          kubectl.kubernetes.io/last-applied-configuration:
                            {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"cka-deployment9"},
Selector:                 app=cka-deployment9
Type:                     NodePort
IP:                       10.8.15.150
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32600/TCP
Endpoints:                10.4.0.17:80,10.4.2.8:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Check the endpoint IP. It should match the deployment pods.

Question-12: List the pods running under service cka-deployment-service and print the pod names to file /opt/D407/service-pod.txt

You can use below command to get the list of pods running under a service:

kubectl get endpoints cka-deployment-service -o jsonpath={.subsets[*].addresses[*].targetRef.name} > /opt/D407/service-pod.txt

Post executing the command validate output of file /opt/D407/service-pod.txt

Question-13: select the pod with highest cpu utilization among the pods with label as "name=cpu-utilization" and put the pods names into the file /opt/D407/cka-cpupods.txt

We can run this command to redirect the output in file:

kubectl top pods --selector=name=cpu-utilization --sort-by=cpu --no-headers | head -1 | awk '{print $1}' > /opt/D407/cka-cpupods.txt

Validate the output of file /opt/D407/cka-cpupods.txt.

In case we need to get the pods of highest cpu consumption in all namespaces we have to use the below command:

kubectl top pods -A | sort --reverse --key 3 --numeric | head -3

Question-14: take etcd back to the file /opt/D407/cka-etcdbkp

To solve this question follow under mentioned steps:

1. Set environment variable

   export ETCDCTL_API=3

2. Get the details of cert, key, cacert and endpoints from /etc/kubernetes/manifests/etcd.yaml

3. Run command to initiate the snapshot

   etcdctl snapshot save /opt/D407/cka-etcdbkp --cert=<cert_file> --key=<key_file> --cacert=<ca_file> --endpoints=<endPoint>

Question-15: verify all the nodes in the cluster are in healthy state or not. If needed troubleshoot and bring the nodes to ready status.

To solve this we can perform under mentioned steps:

1. Check the nodes status with command:
   kubectl get nodes

```
WKMIN7290299:kubernets jaskumar$ kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
gke-cluster-1-default-pool-d74ac0e8-86sn    Ready    <none>   43h   v1.17.6-gke.11
gke-cluster-1-default-pool-d74ac0e8-c8cq    Ready    <none>   43h   v1.17.6-gke.11
gke-cluster-1-default-pool-d74ac0e8-tprj    Ready    <none>   43h   v1.17.6-gke.11
WKMIN7290299:kubernets jaskumar$
```

   Check the **STATUS** field. In case for any node if it is not Ready we have to debug the nodes.

2. Validate kubelet service with command on node which is not healthy

   service kubelet status

3. In case service is down we have to bring it up by starting the service

   service kubelet start

Reason may be something else for node unhealthy so we need to troubleshoot that.

Question-16: Increase the number replicas of the deployment cka-deployment9 to 4

To increase the number of replicas from 2 to 4 run the command:

kubectl scale --replicas=4 deployment/cka-deployment9

You can validate the deployment with command:

kubectl get deploy cka-deployment9

```
WKMIN7290299:kubernets jaskumar$ kubectl get deploy cka-deployment9
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
cka-deployment9   4/4     4            4           64m
WKMIN7290299:kubernets jaskumar$
```

Question-17: create a persistent volume with the below given specification.
Volume Name : cka-pv
Storage: 200MB
Access Modes: ReadWriteMany
Host Path: /ckapv
To create a persistent volume as per the request follow under mentioned steps:

1. Create persistent volume template which looks like the below one:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cka-pv
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 200Mi
  hostPath:
    path: /ckapv
```

2. Apply the template with command:

kubectl apply -f cka-pv.yaml

3. Validate the persistent volume with command:

kubectl describe pv cka-pv

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pv cka-pv
Name:            cka-pv
Labels:          <none>
Annotations:     kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"PersistentVolume","metadata":{"annotations":{},"name":"cka-pv"},
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:
Status:          Available
Claim:
Reclaim Policy:  Retain
Access Modes:    RWX
VolumeMode:      Filesystem
Capacity:        200Mi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /ckapv
    HostPathType:
Events:            <none>
WKMIN7290299:kubernets jaskumar$ vi cka-pv.yaml
```

Question-18: create a deployment using a spec file and once the deployment is created, delete the objects and copy the spec file to the location /opt/D407/cka-simpledep.yml

You can give it any name but should be created using nginx image.

To solve this question follow under mentioned steps:

1. Create a deployment template with command:

kubectl create deployment nginx-dep --image=nginx --dry-run -o yaml > nginx-dep.yaml

2. This is how the template looks like

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-dep
  name: nginx-dep
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: nginx-dep
  template:
    metadata:
      labels:
        app: nginx-dep
    spec:
      containers:
      - image: nginx
        name: nginx
```

3. Now you can delete the objects with command:

   kubectl delete deployment nginx-dep

4. Copy the nginx-dep.yaml file at location /opt/D407/cka-simpledep.yml

Question-19: create a pod that starts along with the kubelet running on the node node01. The pod creation should be initiated from kube-apiserver and pod should be based on nginx image

This question is on static pod and for that we need to follow under mentioned steps:

1. Create a pod template with command:

   kubectl run --generator=run-pod/v1 nginx-pod --image=nginx --dry-run -o yaml > nginx-pod.yaml

2. This is how the template looks like the one shown below:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx-pod
  name: nginx-pod
spec:
  containers:
  - image: nginx
    name: nginx-pod
```

3. SSH to node01 with command:

ssh node01

4. Copy the step 2 contents in file **/etc/kubernetes/manifests/nginx-pod.yaml**
5. Restart the kubelet service with command:

   service kubelet restart

6. As kubelet will also create mirror pod and therefore you can validate it on master node with command:

   kubectl get pods -n default

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods -n default
NAME                                                  READY   STATUS             RESTARTS   AGE
nginx                                                 0/1     CrashLoopBackOff   579        2d
nginx-pod-gke-cluster-1-default-pool-d74ac0e8-c8cq    1/1     Running            0          4m29s
nginx-static-pod-gke-cluster-1-default-pool-d74ac0e8-86sn   1/1     Running   0          32h
WKMIN7290299:kubernets jaskumar$
```

   In my case the default namespace is **dev** which I have explicitly set and by default pod is created in **default** namespace if you did not specify the namespace and there we need to pass the namespace name in above command.

Question-20: create a pod with volume that has the life cycle of the pod and mount this volume under **/mnt/data** of container. Create a file **file1** inside the directory and the pod should be based on the **redis** image.

We need to create **emptyDir** volume to have the same lifecycle as pod.

To solve this question follow under mentioned steps:

1. Create pod template with command:

   kubectl run --generator=run-pod/v1 redis --image=redis --dry-run -o yaml > redis.yaml

2. We need to add the volumes and volumeMount section in template. This is how the template should look like:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: redis
```

```
    name: redis
  spec:
    containers:
    - image: redis
      name: redis
      command: ["/bin/sh","-c","touch
  /mnt/data/file1"]
      volumeMounts:
        - name: volume1
          mountPath: /mnt/data
    volumes:
      - name: volume1
          emptyDir: {}
```

3. Apply the template with command:

   kubectl apply -f redis.yaml

4. Though the pod will crash but still you can describe the pod with command:

   kubectl describe pod redis

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pod redis
Name:         redis
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Thu, 16 Jul 2020 20:53:38 +0530
Labels:       run=redis
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"run":"redis"},
Status:       Running
IP:           10.4.1.20
Containers:
  redis:
    Container ID:  docker://0b5456271b7ee64f25173458331e373879dd887b86bcf2144b4630f988201f97
    Image:         redis
    Image ID:      docker-pullable://redis@sha256:76ff608805ca40008d6e0f08180d634732d8bf4728b85c18ab9b
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
      touch /mnt/data/file1 && test -f /mnt/data/file1 && echo exist
    State:          Waiting
      Reason:       CrashLoopBackOff
    Last State:     Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Thu, 16 Jul 2020 20:59:11 +0530
      Finished:     Thu, 16 Jul 2020 20:59:11 +0530
    Ready:          False
    Restart Count:  6
    Environment:    <none>
    Mounts:
      /mnt/data from volume1 (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
Conditions:
  Type              Status
  Initialized       True
```

Question-21: create a namespace name **cka-ns** and create a deployment based on nginx image with 2 replicas in this namespace.

To solve this follow under mentioned steps:

1. Create a namespace with command:

   kubectl create ns cka-ns

2. Create a deployment template with command:

   kubectl create deployment deploy1 --image=nginx --dry-run -o yaml > deploy.yaml

3. Modify the template to look like the one below:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
```

```
      app: deploy1
    name: deploy1
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: deploy1
    template:
      metadata:
        labels:
          app: deploy1
      spec:
        containers:
        - image: nginx
            name: nginx
```

4. Apply the template with command:

   kubectl apply -f deploy.yaml --namespace=cka-ns

5. You can list the deployment with command:

   kubectl get deployment -n cka-ns

```
WKMIN7290299:kubernets jaskumar$ kubectl get deployment -n cka-ns
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
deploy1   2/2     2            2           22s
WKMIN7290299:kubernets jaskumar$
```

6. Validate all pods are running with command:

   kubectl get pods -n cka-ns

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods -n cka-ns
NAME                       READY   STATUS    RESTARTS   AGE
deploy1-5d98f66655-htptq   1/1     Running   0          2m19s
deploy1-5d98f66655-xht7v   1/1     Running   0          2m18s
WKMIN7290299:kubernets jaskumar$
```

Question-22: Deploy a pod on a node having specific label **team=platform**.

This question is based on **nodeSelector** and for that we have two options:

Option-1: You can directly specify the nodeName in template file. Have a look on below template:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  nodeName: gke-cluster-1-default-pool-d74ac0e8-tprj
  containers:
  - image: nginx
    name: nginx
```

But now the question is to deploy a pod on a node having a specific label. For example **team=platform** is the label attached to the node. We need to follow under mentioned steps:

1. Check the node if label is attached or not with command:

   kubectl get nodes --show-labels | grep team=platform | awk '{print $1}'

```
WKMIN7290299:kubernets jaskumar$ kubectl get nodes --show-labels | grep team=platform | awk '{print $1}'
gke-cluster-1-default-pool-d74ac0e8-c8cq
WKMIN7290299:kubernets jaskumar$
```

2. Next create a pod template with command:

   kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml > nginx-node.yaml

3. Modify the template to look like the one shown below:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
```

```
    name: nginx
  spec:
    containers:
    - image: nginx
      name: nginx
    nodeSelector:
        team: platform
```

4. Notice the **nodeSelector** part in above template.
5. Apply the template with command:

   kubectl apply -f nginx-node.yaml

6. Validate it is should on the same node which has the label **team=platform** with command:

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods -o wide | grep nginx-2
nginx-2                       1/1     Running           0          3m    10.4.0.20   gke-cluster-1-default-pool-d74ac0e8-c8cq   <none>           <none>
WKMIN7290299:kubernets jaskumar$
```

Question-23: Create 5 nginx pods in which two of them is labelled env=prod and three of them is labelled as env=dev

To create 5 pods with the labels as mentioned in question run the commands:

kubectl run nginx-1 --image=nginx -l=env=prod
kubectl run nginx-2 --image=nginx -l=env=prod
kubectl run nginx-3 --image=nginx -l=env=dev
kubectl run nginx-4 --image=nginx -l=env=dev
kubectl run nginx-5 --image=nginx -l=env=dev
Validate all the pods with specified labels

```
WKMIN7290299:kubernets jaskumar$ kubectl get pods --selector=env=prod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-1-6cfb85f6dd-vtwn8   1/1     Running   0          65s
nginx-2-7b4cf48765-rzhdw   1/1     Running   0          56s
WKMIN7290299:kubernets jaskumar$ kubectl get pods --selector=env=dev
NAME                     READY   STATUS    RESTARTS   AGE
nginx-3-74c66b4764-2wwwl   1/1     Running   0          55s
nginx-4-75d9bf4fd5-jpl5g   1/1     Running   0          49s
nginx-5-789f477d4b-fqq5r   1/1     Running   0          37s
```

Question-24: Label a node with nodeName=nginxnode and deploy an nginx pod on this node

To solve this question follow under mentioned steps:

1. Apply the label to the node with command:

   kubectl label node gke-cluster-1-default-pool-d74ac0e8-tprj nodeName=nginxnode

```
WKMIN7290299:kubernets jaskumar$ kubectl label node gke-cluster-1-default-pool-d74ac0e8-tprj nodeName=nginxnode
node/gke-cluster-1-default-pool-d74ac0e8-tprj labeled
WKMIN7290299:kubernets jaskumar$
```

2. Create pod template with command:

   kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml > nginx.yaml

3. Modify **nginx.yaml** template to add the **nodeSelector** configuration as shown in below template:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  nodeSelector:
    nodeName: nginxnode
  containers:
  - image: nginx
      name: nginx
```

4. Apply the template with command:

   kubectl apply -f nginx.yaml

5. Validate the pod is deployed on right node with command:

   kubectl get pod nginx -o wide

```
WKMIN7290299:kubernets jaskumar$ kubectl get pod nginx -o wide
NAME    READY   STATUS    RESTARTS   AGE     IP          NODE                                          NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          2m14s   10.4.1.23   gke-cluster-1-default-pool-d74ac0e8-tprj   <none>           <none>
WKMIN7290299:kubernets jaskumar$
```

kubectl describe pod nginx | grep -i Node-Selector

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pod nginx
Name:         nginx
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Fri, 17 Jul 2020 23:07:11 +0530
Labels:       run=nginx
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"run":"nginx
Status:       Running
IP:           10.4.1.23
Containers:
  nginx:
    Container ID:   docker://41ffade71a457f78475748e4d039d29dcb1477653cdb7d4bc733e73ad1eaa4eb
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:8ff4598873f588ca9d2bf1be51bdb117ec8f56cdfd5a81b5
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 17 Jul 2020 23:07:13 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-v6pqq:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-v6pqq
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  nodeName=nginxnode
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
```

Question-25: Create a deployment manifest file called webapp with nginx image and 5 replicas. Don't apply the manifest file and put it in the directory mentioned in exam.

You can create a deployment with name **webapp** and with **nginx** image with command:

kubectl create deployment webapp --image=nginx --dry-run -o yaml > webapp.yaml

This is the manifest file **webapp.yaml** looks like:

```
apiVersion: apps/v1
kind: Deployment
```

```
  metadata:
    labels:
      app: webapp
    name: webapp
  spec:
    replicas: 5
    selector:
      matchLabels:
        app: webapp
    strategy: {}
    template:
      metadata:
        labels:
          app: webapp
      spec:
        containers:
        - image: nginx
          name: nginx
```

We just need to remove not needed information and set the replicas to 5.

Question-26: Create deployment of webapp with nginx:1.17.1 with container port 80 and verify the image version. Once this is done update the image version to 1.17.4. Check rollout history and make sure everything is working fine. Lastly undo the version to previous version and check the image version.

To solve this question follow under mentioned steps:

1. Create deployment manifest with command:

   kubectl create deployment webapp --image=nginx:1.17.1 --dry-run -o yaml > webapp.yaml

2. Add ports section in the manifest file as shown below:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: webapp
  name: webapp
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - image: nginx:1.17.1
        name: nginx
        ports:
          - containerPort: 80
```

3. Apply the manifest with command:

   kubectl apply -f webapp.yaml

4. Validate the image version with command:

   kubectl describe deploy webapp

```
WKMIN7290299:kubernets jaskumar$ kubectl describe deploy webapp
Name:                   webapp
Namespace:              dev
CreationTimestamp:      Fri, 17 Jul 2020 23:43:01 +0530
Labels:                 app=webapp
Annotations:            deployment.kubernetes.io/revision: 1
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"webapp"},"name":"webapp","name
Selector:               app=webapp
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=webapp
  Containers:
   nginx:
    Image:        nginx:1.17.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   webapp-84c76cb8cc (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  14s   deployment-controller  Scaled up replica set webapp-84c76cb8cc to 1
```

5. Next to update the image version to 1.17.4 run the command:

   kubectl set image deployment webapp nginx=nginx:1.17.4

6. Validate image version again with command:

kubectl describe deploy webapp

```
WKMIN7290299:kubernets jaskumar$ kubectl describe deploy webapp
Name:                   webapp
Namespace:              dev
CreationTimestamp:      Fri, 17 Jul 2020 23:43:01 +0530
Labels:                 app=webapp
Annotations:            deployment.kubernetes.io/revision: 2
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"webapp"},"nam
Selector:               app=webapp
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=webapp
  Containers:
   nginx:
    Image:        nginx:1.17.4
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
```

7. To check the rollout history run the command:

kubectl rollout history deployment webapp

```
WKMIN7290299:kubernets jaskumar$ kubectl rollout history deployment webapp
deployment.apps/webapp
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

8. Lastly to undo the deployment to previous version run the command:

kubectl rollout undo --to-revision=1 deployment webapp

9. Describe the deployment again to check the image version with command:

kubectl describe deploy webapp

```
WKMIN7290299:kubernets jaskumar$ kubectl describe deploy webapp
Name:                   webapp
Namespace:              dev
CreationTimestamp:      Fri, 17 Jul 2020 23:43:01 +0530
Labels:                 app=webapp
Annotations:            deployment.kubernetes.io/revision: 3
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"ani
Selector:               app=webapp
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=webapp
  Containers:
   nginx:
    Image:        nginx:1.17.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
```

10. Validate all pods and replica sets is also working fine with command:

    kubectl get pod webapp
    kubectl get rs webapp

Question-27: Create a hostPath persistent volume named **task-pv-volume** with storage 10Gi, access mode as ReadWriteOnce and storageClassName as manual. It should mount to /mnt/data location.
Next create a **persistentVolumeClaim** of at least 3Gi storage and ReadWriteOnce and verify status is Bound.

Lastly create a pod with redis image and configure the above created persistentVolumeClaim on given path **/data/redis** that lasts the lifetime of a pod.

To solve this question we need to follow under mentioned steps:
1. Create the persistent volume manifest file which should look like the one shown below:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
```

```
    name: pv
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  hostPath:
      path: /mnt/data
```

2. Apply the manifest with command:

   kubectl apply -f task-pv-volume.yaml

```
WKMIN7290299:kubernets jaskumar$ kubectl apply -f task-pv-volume.yaml
persistentvolume/task-pv-volume created
```

3. Validate the persistent volume with command:

   kubectl describe pv task-pv-volume

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pv task-pv-volume
Name:            task-pv-volume
Labels:          name=pv
Annotations:     kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"PersistentVolume","metadata":{"annotations":{},"labels":{"name
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    manual
Status:          Available
Claim:
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        10Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/data
    HostPathType:
Events:          <none>
```

4. Next we would like to create the **persistentVolumeClaim** having the template which looks like the one below:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-volume
spec:
  accessModes:
```

```
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
  storageClassName: manual
  selector:
    matchLabels:
        name: pv
```

5. Apply the manifest with command:

   kubectl apply -f pvc.yaml

```
WKMIN7290299:kubernets jaskumar$ kubectl apply -f pvc.yaml
persistentvolumeclaim/task-pv-volume created
```

6. Validate persistentVolumeClaim if it is bound to right PersistentVolume with command:

   kubectl get pvc -o wide

```
WKMIN7290299:kubernets jaskumar$ kubectl get pvc -o wide
NAME             STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE     VOLUMEMODE
task-pv-volume   Bound    task-pv-volume   10Gi       RWO            manual         2m42s   Filesystem
WKMIN7290299:kubernets jaskumar$
```

7. Lastly to create Pod with redis image create the manifest with command:

   kubectl run --generator=run-pod/v1 redis --image=redis --dry-run -o yaml > redis.yaml

8. The manifest should look like the one shown below:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: redis
  name: redis
spec:
  containers:
  - image: redis
    name: redis
    volumeMounts:
```

```
      - name: my-volume
        mountPath: /mnt/redis
  volumes:
    - name: my-volume
      persistentVolumeClaim:
          claimName: task-pv-volume
```

9. Apply the manifest with command:

   kubectl apply -f redis.yaml

10. Validate the pod with command:

    kubectl describe pod redis

Question-28: Create a pod with nginx image and configure the pod with capabilities **NET_ADMIN** and **SYS_TIME**. Verify the capabilities.

To solve this question we have to follow under mentioned steps:

1. Create a pod manifest with the command:

   kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml > nginx.yaml

2. Add the capabilities as asked in question. This is how the manifest looks like:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    securityContext:
      capabilities:
            add: ["NET_ADMIN", "SYS_TIME"]
```

3. Next apply the manifest with command:

kubectl apply -f nginx.yaml

Question-29: Create a NetworkPolicy which denies all ingress traffic

To solve this question we have to create a NetworkPolicy using the manifest as shown below:

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

You can apply the manifest with command:
kubectl apply -f network-policy.yaml

```
WKMIN7290299:kubernets jaskumar$ kubectl apply -f network-policy.yaml
networkpolicy.networking.k8s.io/deny-default-ingress created
WKMIN7290299:kubernets jaskumar$
```

You can validate the network policies with command:

kubectl get networkpolicy

```
WKMIN7290299:kubernets jaskumar$ kubectl get networkpolicy -o wide
NAME                   POD-SELECTOR   AGE
deny-default-ingress   <none>         2m18s
WKMIN7290299:kubernets jaskumar$
```

Question-30: Create an nginx pod with a manifest yaml file and use the label **my-nginx** and expose it to port 80.

To solve this question we will follow under mentioned steps:
1. Create a pod manifest with command:

   kubectl run --generator=run-pod/v1 nginx --image=nginx -l=app=my-nginx --port=80 --dry-run -o yaml > nginx.yaml

2. This is how the manifest looks like

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: my-nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    ports:
      - containerPort: 80
```

3. Apply the manifest with command:

   kubectl apply -f nginx.yaml
4. Validate the pod with command:

   kubectl describe pod nginx

```
WKMIN7290299:kubernets jaskumar$ kubectl describe pod nginx
Name:         nginx
Namespace:    dev
Priority:     0
Node:         gke-cluster-1-default-pool-d74ac0e8-tprj/10.128.0.3
Start Time:   Sat, 18 Jul 2020 22:10:39 +0530
Labels:       app=my-nginx
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"app":"my-nginx"},"name":"nginx","names
Status:       Running
IP:           10.4.1.28
Containers:
  nginx:
    Container ID:   docker://42efd063d973836c2b67b15d3dc4f1fe478ec9e86f446f996e844a92b1e16faa
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:8ff4598873f588ca9d2bf1be51bdb117ec8f56cdfd5a81b5bb0224a61565aa49
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 18 Jul 2020 22:10:41 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v6pqq (ro)
Conditions:
```

5. Next create a service manifest with command:

   kubectl create service clusterip nginx-service --tcp=80:80 --dry-run -o yaml > service.yaml

6. Change label **app: nginx-service** with **app: my-nginx** in manifest file. It should look like the one shown below:

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-nginx
  name: nginx-service
spec:
  ports:
  - name: tcp-port
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: my-nginx
    type: ClusterIP
```

7. Apply the manifest with command:

   kubectl apply -f nginx-service.yaml

8. You can validate this service endpoints with command:

```
WKMIN7290299:kubernets jaskumar$ kubectl describe svc nginx-service
Name:              nginx-service
Namespace:         dev
Labels:            app=my-nginx
Annotations:       cloud.google.com/neg: {"ingress":true}
                   kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app"
Selector:          app=my-nginx
Type:              ClusterIP
IP:                10.8.8.169
Port:              tcp-port  80/TCP
TargetPort:        80/TCP
Endpoints:         10.4.1.28:80
Session Affinity:  None
Events:            <none>
WKMIN7290299:kubernets jaskumar$
```