**Tips:**

* Check for the namespace and context info before answering the question
* the history the command to reuse already entered commands
* Export do
* Vimrc (optional)
* <https://helm.sh/docs/>
* <https://kubernetes.io/docs/>

**General:**kubectl create -f file.yaml

Kubectl get <workload> -n <namespace>

Kubectl get <workload> --all-namespaces

kubectl describe <workload> <name>

kubectl delete <workload> <name>

kubectl logs <pod> -c <opt container> -n <ns> (-f option for live logs)

***kubectl api-resources. – Gives info about the name , shortname, namspaced/not, api version and kind of all the resources.***

**Kube api server pod: cat /etc/kubernetes/manifests/kube-apiserver.yaml- AC , authorization modes**

$HOME/.kube/config- user, context and cluster info. AUthenticatiuon

***Kubectl explain <resource-name> - Gives info about a particular resource***

[**https://dev.to/coherentlogic/answers-to-five-kubernetes-ckad-practice-questions-2020-3h0p**](https://dev.to/coherentlogic/answers-to-five-kubernetes-ckad-practice-questions-2020-3h0p) **PODS:**  
kubectl get pod redis -o yaml ==== open the yaml file for a pod

kubectl get pod redis -o yaml > redis.yaml === store the yaml into a file

kubectl get pods --selector env=dev

kubectl get pod --selector env=prod,bu=finance,tier=frontend

kubectl run newpod --image=nginx

kubectl run httpd --image=httpd:alpine --expose=true --port=80

kubectl run nginx --image=nginx --dry-run=client -o yaml > nginx-pod.yaml. === dry run will not create the resource. Instead, tell you whether the resource can be created and if your command is right. This command can be used when you want to generate a manifest and don’t apply it

kubectl delete pod NAME --grace-period=0 --force

**Editing a POD**

kubectl get pod webapp -o yaml > my-new-pod.yaml

vi my-new-pod.yaml

kubectl delete pod webapp

kubectl create -f my-new-pod.yaml

**Editing a POD inside a deployment**  
kubectl edit deployment my-deployment

**Getting into a Pod and checking logs:**  
kubectl exec webapp -- cat /log/app.log

**Replica set**

kubectl scale rs new-replica-set –replicas=5

When you update RS, even though the image is updated, it will not automatically spin up new pods. You have to manually delete the pods and then the new ones will be created.  
apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: replicaset-1

spec:

replicas: 2

**selector:**

**matchLabels:**

**tier: nginx**

template:

metadata:

**labels:**

**tier: nginx**

spec:

containers:

- name: nginx

image: nginx

**Deployments:**

kubectl create deployment nginx --image=nginx

kubectl create deployment nginx --image=nginx --replicas=4

kubectl create deployment redis-deploy --namespace=dev-ns --replicas=2 --image=redis

kubectl scale deployment nginx --replicas=4

kubectl rollout status deployment nginx. == live status of rollout

kubectl rollout history deployment nginx == This shows revision and change cause. if you do not record, the change cause will be none. To record, add –-record while creating or applying the dep.

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

**kubectl edit deployments nginx** == use this to edit the depl

**Namespace:**  
kubectl create namespace dev-ns

**ConfigMaps:**

kubectl create configmap webapp-config-map --from-literal=APP\_COLOR=darkblue --from-literal=APP\_OTHER=disregard

* Whenever you make any changes to key/value in CM, you have to restart/recreate the pod

**Three ways to use config maps in a pod:**

**Mount into volumes:**  
  
**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**configMap**:

**name**: myconfigmap

**spec**:

**containers**:

- **name**: app

**command**: ["/bin/sh", "-c", "printenv"]

**image**: busybox:latest

**envFrom**:

- **configMapRef**:

**name**: myconfigmap

**spec**:

**containers**:

- **name**: demo

**image**: alpine

**command**: ["sleep", "3600"]

**env**:

*# Define the environment variable*

- **name**: PLAYER\_INITIAL\_LIVES *# Notice that the case is different here*

*# from the key name in the ConfigMap.*

**valueFrom**:

**configMapKeyRef**:

**name**: game-demo *# The ConfigMap this value comes from.*

**key**: player\_initial\_lives *#*

**Secrets**:  
kubectl create secret **generic** db-secret --from-literal=DB\_Host=sql01 --from-literal=DB\_User=root --from-literal=DB\_Password=password123

Creating a TLS secret:  
kubectl create secret tls my-tls-secret --cert=path/to/cert/file --key=path/to/key/file -n namespace

**Service:**  
kubectl expose pod redis --port=6379 --name=redis-service

**Security:**  
kubectl exec ubuntu-sleeper – whoami => get into the pod and execute a command

To add security context, meaning who has to access the container inside a pod, what capablilites they have, add the below under the pod (apply to all containers) or at the level of container.  
  
securityContext:

runAsUser: 1010

capabilities:

add: ["SYS\_TIME"]

If you want it to be root user, remove runAsUser and also add the securityContext at the container level, not the pod level

**ServiceAccounts: (Motive is to access the api server)**

kubectl create sa dashboard-sa === this creates a plain sa without any token and stuff.

kubectl create token dashboard-sa

To add this new sa to the pod, we need to add it under spec. template.spec= > serviceAccountName: dashboard-sa

**Resource Requirements:**

At the pod level setting the resources REQUEST and LIMITS:  
**spec.containers** =>   
resources:

limits:

memory: 15Mi

requests:

memory: 5Mi  
  
  
Limit range at the NS level , to ensure that the pod has the requests and limits set by default.

* LimitRange validations occur only at Pod admission stage, not on running Pods. If you add or modify a LimitRange, the Pods that already exist in that namespace continue unchanged.

**apiVersion: v1**

**kind: LimitRange**

**metadata:**

**name: cpu-resource-constraint**

**spec:**

**limits:**

**- default: # this section defines default limits**

**cpu: 500m**

**defaultRequest: # this section defines default requests**

**cpu: 500m**

**max: # max and min define the limit range**

**cpu: "1"**

**min:**

**cpu: 100m**

**type: Container**

Resource Quota at NS level. To restrict how much mem and cpu that all the pods in NS can consume.

**apiVersion: v1**

**kind: ResourceQuota**

**metadata:**

**name: pods-medium**

**spec:**

**hard:**

**cpu: "10"**

**memory: 20Gi**

**pods: "10"**

* ResourceQuota support is enabled by default for many Kubernetes distributions. It is enabled when the [API server](https://kubernetes.io/docs/concepts/architecture/#kube-apiserver) --enable-admission-plugins= flag has ResourceQuota as one of its arguments.
* A resource quota is enforced in a particular namespace when there is a ResourceQuota in that namespace.

**Taints and Tolerations:**Taints is a way of telling the nodes to accept only specific tolerated Pods and not a way to tell the tolerated pods to go to the tainted nodes

In a cluster, why does the master node does not have any pods. Its because the master node is tainted by default.

kubectl taint nodes node01 spray=mortein:NoSchedule

kubectl taint nodes <nodeName> key=value:effect

effect can be : NoSchedule, PreferNoSchedule, NoExecute

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

env: test

spec:

containers:

- name: nginx

image: nginx

imagePullPolicy: IfNotPresent

**tolerations**:

- **key**: "spray"

**operator**: "Equal"

**effect**: "NoSchedule"

Removing the taints:  
same as creating the taint but with a “-” at the end.

kubectl taint nodes node01 spray=mortein:NoSchedule**-**

**Node Selector and Node Affinity:**This is more of pod oriented, where you are trying to tell the pod specifically to be placed in which node.

NS is more of strict selection, where you add a label to the node and then use that label in yaml file under spec.

kubectl label node <node-name> <label-key>=<label-value>

kubectl label node node01 color=blue

apiVersion: v1

kind: Pod

metadata:

name: cuda-test

spec:

containers:

- name: cuda-test

image: "registry.k8s.io/cuda-vector-add:v0.1"

resources:

limits:

nvidia.com/gpu: 1

**nodeSelector:**

**accelerator: nvidia-tesla-p100**

NA is more of lineant in terms of adding condition like, IN, NOT IN, NOT etc..

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

**affinity:**

**nodeAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**nodeSelectorTerms:**

**- matchExpressions:**

**- key: color**

**operator: In**

**values:**

**- blue**

containers:

- name: nginx

image: nginx

imagePullPolicy: IfNotPresent

**Multi\_Container Pods:**

Init containers are the containers which are started before other containers and then closes doing its job after that the actual containers start  
  
spec:

1. containers:
2. - name: myapp-container
3. image: busybox:1.28
4. command: ['sh', '-c', 'echo The app is running! && sleep 3600']
5. initContainers:
6. - name: init-myservice
7. image: busybox
8. command: ['sh', '-c', 'git clone <some-repository-that-will-be-used-by-application> ;']

**Readiness and Liveliness Probes:**

**When you do pod describe you will find below:**

**Pod status:**1. Pending - Before scheduler finds a node to put the pod

2. ContainerCreating – while the scheduler found the node and pulling the image and creating the container

3. Running – Pod is running

**Pod Conditions: More descriptive info than status**

1. Pod scheduled
2. Initialized
3. ContainersReady
4. Ready

This is a true false condition

How does k8 know whether the app inside the container is actually running ?

We need a way to tie the READY condition to the actual state of the application inside the container – **Readiness Probe**

Ex: if there are 2 pods which are serving the web ui and a new pod is added., if you do not have RP , the service might think its up and traffic might be routed to the 3rd pod and users will see error on web page. If we define proper RP, it will avoid this

Http probe:  
containers:

- env:

- name: APP\_START\_DELAY

value: "80"

image: kodekloud/webapp-delayed-start

imagePullPolicy: Always

name: simple-webapp

**readinessProbe:**

**httpGet:**

**path: /ready**

**port: 8080**

simple-webapp-2 0/1 Running 0 57s we see that the ready status is 0/1 it means even though container is running, its not ready because of the RP

Similarly once the app is running what if it stops working. May be due to a bug in the application, but the container is up and runnings. – **Liveliness Probe**

**livenessProbe:**

**httpGet:**

**path: /live**

**port: 8080**

**initialDelaySeconds: 80**

**periodSeconds: 1**

**failureThreshold: 10  
  
Observability:**kubectl top node – to see all the metrics of a node – cpu , mem etc

**Labels Selectors and Annotations:**

**Labels:** similar to tags

Selectors: uses labels to filter out

Annotations: its more for informatory purposes

**Blue Green deployment:**

Create a deployment with label version =v1 at the Pod level

Create a service with selector version =v1

Create a new deployment with label version =v2 at the Pod level

Do all the required testing on this new deployment.

Now switch back to version =v2 in service.

**Canary deployment:**

Similar to B G, but there is a canary deployment that starts up only a few number of pods and only partial traffic flows through it.

Once we are good with the testing, we replace the deployment with new version

Create a deployment with labels version =v1 **and app=FE** at the Pod level (replicas 5)

Create another deployment with labels version =v2 **and app=FE** at the Pod level (less replicas, may be 1)

Create a service with selector app=FE

**Jobs:**

Default behaviour of pods is to keep the containers running in order to keep the application running. So if the container is just performing a one time task and shutting down, the pod will again spin up the container.

This is defined by restartPolicy on pod , which is by default Always.Other options – never/failure

**Job is**  used to run set of pods to perform a given task to completion.

kubectl create job throw-dice-job --image=kodekloud/throw-dice --dry-run=client -o yaml > test.yaml

apiVersion: batch/v1

kind: Job

metadata:

creationTimestamp: null

name: throw-dice-job

spec:

**completions: 2**

**parallelism: 2**

template:

metadata:

creationTimestamp: null

spec:

containers:

- image: kodekloud/throw-dice

name: throw-dice-job

resources: {}

restartPolicy:

To check the attempts taken by the job to succeed, go to k describe and check this Pods Statuses: 0 Active (0 Ready) / 2 Succeeded / 5 Failed

**Cron Job:**  
Is a wrapper above the job which helps in scheduling the job

kubectl create cronjob throw-dice-cron-job --image=kodekloud/throw-dice **--schedule="30 21 \* \* \*"** --dry-run=client -o yaml > test1.yaml

**Service & Networking:| (Thursday EOD)**

**Services:**

kubectl create service nodeport webapp-service --tcp=8080:8080 --dry-run=client -o yaml > test.yaml

Tip: if you do not know what command to use, start with  
 kubectl create service -h

This says you can use commands like clusterip, nodeport…

kubectl create service clusterip -h  
  
You can also create and attach the service with one command:  
kubectl expose deployment nginx --port=80 --target-port=8000

**INGRESS:**

kubectl create ingress simple-pay --rule="\*/pay=pay-service:8080" --dry-run=client -o yaml > test.yaml

ssl redirect and rewrite target annotations. ??

**Network Policies:**Ingress and Egress is always – allow all on all of the pods inside a cluster.

If you allow something on a pod to ingress, the response need not be egressed. That is taken care automatically

Pod selector to match the pod.- **podSelector**

Also you can use namespace selector to match the ns- **namespaceSelector**

Also from certain ip addresses: **ipBlock**

*There is NO imperative command for Network policy. You need to copy code from the K8s doc*

**State persistence (Sat EOD)**

**Volumes:**  something which is created at pod level. It can be on host or on cloud or can be a PVC

**volumes**:

- **name**: example-volume

*# mount /data/foo, but only if that directory already exists*

**hostPath**:

**path**: /data/foo *# directory location on host*

**type**: Directory *# this field is optional*

**volumes**:

- **name**: cache-volume

**emptyDir**:

**sizeLimit**: 500Mi

**volumes**:

- **name**: config-vol

**configMap**:

**name**: log-config

**items**:

- **key**: log\_level

**path**: log\_level

**volumes**:

- **name**: test-volume

**nfs**:

**server**: my-nfs-server.example.com

**path**: /my-nfs-volume

**readOnly**: **true**

**Volume mount:** Something which is created at container level. This points to the volumes mentioned above and has its own path

**volumeMounts**:

- **mountPath**: /my-nfs-data

**name**: test-volume

Persistant volume: this is a volume that is created separately.

**apiVersion**: v1

**kind**: PersistentVolume

**metadata**:

**name**: pv0003

**spec**:

**capacity**:

**storage**: 5Gi

**volumeMode**: Filesystem

**accessModes**:

- ReadWriteOnce

**persistentVolumeReclaimPolicy**: Recycle

**storageClassName**: slow

**mountOptions**:

- hard

- nfsvers=4.1

**nfs**:

**path**: /tmp

**server**: 172.17.0.2

*Note* *There is NO imperative command for PV*  
Persistant Volume Claim:   
if PVC does not find a matching PV with access modes, it will be pending state.

**apiVersion**: v1

**kind**: PersistentVolumeClaim

**metadata**:

**name**: myclaim

**spec**:

**accessModes**:

- ReadWriteOnce

**volumeMode**: Filesystem

**resources**:

**requests**:

**storage**: 8Gi

**storageClassName**: slow

**selector**:

**matchLabels**:

**release**: "stable"

**matchExpressions**:

- {**key: environment, operator: In, values**: [dev]}

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: myfrontend

**image**: nginx

**volumeMounts**:

- **mountPath**: "/var/www/html"

**name**: mypd

**volumes**:

- **name**: mypd

**persistentVolumeClaim**:

**claimName**: myclaim

**Security (Fri EOD)**

**KubeConfig:**

The default kubeConfig is found in $HOME/.kube/config.

To find the $HOME => echo $HOME

This kubeConfig file is used by kube-api-server to look for clusters, context and users

kubectl config use-context research –kubeconfig /root/my-kube-config => when you want to set a context from different file other than default one

If you want to override the default one: mv new-config defult-config

Kubectl config view : give the current kubeconfig details

**API Groups:  
 Under**  named APIs, there are API groups like /apps , /extenstions , /storage.k8s.io etc. under them there are resources like deployment, pods etc. under each resource there are actions like list, get, create etc

**Authorization**:

Inspect the environment and identify the authorization modes configured on the cluster. - cat /etc/kubernetes/manifests/kube-apiserver.yaml

In order to check if the user has permissions: kubectl get **pods --as dev-user**

kubectl create role developer --verb=list,create,delete --resource=pods --dry-run=client -o yaml > test.yaml

kubectl create rolebinding dev-user-binding --role=developer --user=dev-user

**Cluster Roles:**  
kubectl create clusterrole new-node-role --verb=\* --resource=nodes --dry-run=client -o yaml > test.yaml

**Cluster Role Bindings**  
kubectl create clusterrolebinding new-rb1 --clusterrole=new-node-role --user=michelle --dry-run=client -o yaml > test2.yaml

**Admission Controllers**  
**Checking AC info about enabled and disabled admission plugins by default**  
kubectl get pods -n kube-system

kubectl exec -it kube-apiserver-controlplane -n kube-system -- kube-apiserver -h | grep 'enable-admission-plugins'

**if you want to override the default settings:**

Which admission controller is enabled in this cluster which is normally disabled?

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

In order to enable the creation of ns when does not exists: edit this line in above file

- --enable-admission-plugins=NodeRestriction,NamespaceAutoProvision

Note that the NamespaceExists and NamespaceAutoProvision admission controllers are deprecated and now replaced by NamespaceLifecycle admission controller.

The NamespaceLifecycle admission controller will make sure that requests  
to a non-existent namespace is rejected and that the default namespaces such as  
default, kube-system and kube-public cannot be deleted.

Do disable any admission plugins, add the below –disable line

- --enable-admission-plugins=NodeRestriction,NamespaceAutoProvision

- --disable-admission-plugins=DefaultStorageClass

**API Versions:**

Kubectl api-resources

**Custom resource Definition:**

**Helm Charts**Package managing and release managing of Kubernetes application **Helm installation:**

* **C**heck the OS : cat /etc/\*release\*
* Follow the documentation here - <https://helm.sh/docs/intro/install/#from-apt-debianubuntu>
* helm version

**Helm Commands:** do helm -h for the command and then helm <command> -h for the subcommand

To search for a chart in hub: helm search hub wordpress

Anything other than hub, we use: helm search repo <app name>

To add a repo from bitnami artifact registory: helm repo add bitnami <https://charts.bitnami.com/bitnami>

helm install <release name> repo/package

if you don’t want to install, just pull the chart: helm pull -untar bitnami/wordpress

Once you downloaded the chart and modify the values file, to install the chart  
helm install mywebapp apache

Where mywebapp is the version name

And apache is the folder name of the downloaded package.if you are inside the apache folder already use .

If you don’t want to download and change the values.

1. Get the attribute name: helm show values bitnami/node | grep -i replica
2. Install using set option : helm install mynode bitnami/node --set replicaCount=5

**NOTE:**

* kubectl create will always create a new resource, even if a resource with the same name and specifications already exists in the cluster. In such a case, it will throw an error, and not allow you to create the resource until you provide a different resource name.
* Kubectl apply will create if it doesn’t exist and update if it already exists

=====================================================================================================================================================================================================================

**Security context:**  
  
It is at the pod spec level  
spec:

securityContext: # insert this line

runAsUser: 101 # UID

can also be at container level, with capabilities:  
containers:

- image: nginx

imagePullPolicy: IfNotPresent

name: nginx

securityContext: # insert this line

capabilities: # and this

add: ["NET\_ADMIN", "SYS\_TIME"] # this as

**Resources:**  
  
Its at the level of containers:  
containers:

- image: nginx

name: nginx

resources:

requests:

memory: "256Mi"

cpu: "100m"

limits:

memory: "512Mi"

cpu: "200m"

Resource quota with request and limit  
  
apiVersion: v1

kind: ResourceQuota

metadata:

name: compute-resources

spec:

hard:

requests.cpu: "1"

requests.memory: 1Gi

limits.cpu: "2"

limits.memory: 2Gi

requests.nvidia.com/gpu: 4

**Secrets:**

### **How to get the value of a secret** Get the value of mysecret2

kubectl get secret mysecret2 -o yaml

echo -n YWRtaW4= | base64 -d # /D

**validate secrets mounted on to the pod**kubectl exec -it nginx -- /bin/bash

ls /etc/foo # shows username

cat /etc/foo/username # shows admin

**validate the secreat created thru env variable:**kubectl create -f pod.yaml

kubectl exec -it nginx -- env | grep USERNAME

### Create a Secret named 'my-secret' of type 'kubernetes.io/ssh-auth' in the namespace 'secret-ops'. Define a single key named 'ssh-privatekey', and point it to the file 'id\_rsa' in this directory.

k create secret generic my-secret $ns --type="kubernetes.io/ssh-auth" --from-file=ssh-privatekey=id\_rsa $do > sc.yaml

**ServiceAccounts**

**Observabiltiy: 40 min**

**Services: 50 min**

Deployment is created on port =8080. Which means pods can be accessed through this port  
kubectl create deploy foo --image=dgkanatsios/simpleapp --port=8080 --replicas=3

While creating a service for this, keep in mind to specify the target port if it is not same as the port in which the svc is exposed.

### Create a service that exposes the deployment on port 6262. Verify its existence, check the endpoints

kubectl expose deploy foo --port=6262 **--target-port=8080**

**Run a busybox with labels:**  
kubectl run busybox --image=busybox --rm -it --restart=Never **--labels=access=granted** -- wget -O- http://nginx:80 --timeout 2 # This should be fine

**State: 40 min**

k exec -it bb -c bb1 -- /bin/sh

cut the first column from a file and paste it to other file  
/ # cat /etc/passwd | cut -f 1 -d ':' > /etc/foo/passwd

PVC testing:  
  
exec into the pod and go to the mount path and do ls. If it works then the directory is mounted

When you create a pvc with hostPath type, it is tied up to a particular node. So if the pod you create is not in the same node. It fails to mount it correct.. So you can use other types like nfs

**Create a busybox pod with 'sleep 3600' as arguments. Copy '/etc/passwd' from the pod to your local folder**

**show**

kubectl run busybox --image=busybox --restart=Never -- sleep 3600

kubectl cp busybox:/etc/passwd ./passwd # kubectl cp command

# previous command might report an error, feel free to ignore it since copy command works

cat passwd

**Rbac: 50 min**

Kubectl config -h

In case of role binding. , if the resource names is not specified, it is applied to all resources specified. But if resource names (particular pod) is specified, then only that resource is affected

**when creating a role, how do we know what value should be added to - apiGroups attibutes under rules.**

kubectl api-resources will give you the api group name under api version. (don’t add v1)

**API version:  
Enable the v1alpha1 version for rbac.authorization.k8s.io API group on the controlplane node.**

First take a backup of existing api server yaml - cp /etc/kubernetes/manifests/kube-apiserver.yaml /root/kube-apiserverbackup.yaml   
Add this under commands section:  
- --runtime-config=rbac.authorization.k8s.io/v1alpha1

**Installing kube-convert**

Go to doc : tasks – install tools – install and setup kubectl on linux

Follow the 3 steps

Convert the version.

k convert -f ingress-old.yaml --output-version networking.k8s.io/v1

**Helm: 50 min**

**Crd : 20 min**

apiVersion: apiextensions.k8s.io/v1

kind: CustomResourceDefinition

metadata:

name: operators.stable.example.com

# name must match the spec fields below, and be in the form: <plural>.<group>

shortNames:

* op

versions:

- name: v1

served: true

# One and only one version must be marked as the storage version.

storage: true

**Podman: 30 min**

**Ingress**

**Total 5 hours.**

**To be done by: Thur EOD**

**Friday: lightning lab 1 and 2**

1. create PV
2. Create PVC
3. Attach it to the pod

Debug network policy

Mount config map. Run commands on busybox

Multiple volume mounts

**Saturday/sunday: mock exam 1 and 2**