# **CKAD**

## **Certified Kubernetes Application Developer**

#### [Kubernetes Core Components](#_q1e6n0nk9sn5) [Basics of Kubernetes](#_gs5hhaosmoaj) [Yaml File Configuration](#_vg4u5t984qba) [Dockerfile and kubernetes yaml](#_5305oa5xsgay) [Environment Variables](#_zhpw4zt0h3ib) [Configuration](#_tkwc9y7gomup) [ConfigMap](#_u15ck6h7npx) [Secrete](#_9hn61zowdyl8) [Service Account](#_kkr7h2hts021) [Resources](#_sdk54utlo3z6) [Taint & Tolerance](#_9r66o9e068uz) [Node Selector](#_msdmw62bqqlf) [Node Affinity](#_d13p1nvj62ta) [Multi Container](#_c9sg0vqgvk3f) [Observability](#_rn1eqxf2n7pu) [Readiness Probes](#_ri3d1gbw7046) [Liveness Probes](#_ivrskfz1kh2z) [Logging](#_fdnwe2u9neq9) [Monitoring](#_trv6ob9qp01f) [POD Design](#_2mrju3o304h4) [Labels, Selectors & Annotation](#_tzioyifipk7y) [Rolling Updates & Rollback Deployments](#_wac02vhi5emf) [Jobs and CronJobs](#_suix4jwlxflm) [Services and Networking](#_cjj9b0k4qj7t) [Service](#_e9b20r23d3i1) [Network Policy](#_rmi5bq683un7) [Ingress Networking](#_1j3y5n6dab1u) [State Persistent](#_o2myaaq1otmq)

#### **Kubernetes Core Components Explanation**

**Master**

* ***API Server*** - Responsible for orchestration operations inside the cluster and it works as front end for kubernetes. Every component talks to the api server to interact with the kubernetes cluster.
* ***ETCD Cluster -*** Its a key value store used by kubernetes to store all data to manage the kubernetes cluster.
* ***Controller* -** Responsible for responding when nodes/pods go down, then controllers bring new containers in such cases. Types:-  
  *Node Controller.  
  Replication Controller.*
* ***Kube-Scheduler -*** Responsible for distributing load (containers/pods) across the nodes.  
  It assigns pods to the nodes.

**Worker**

* ***KUBLET -*** It runs as an agent on each node in the cluster and listens to instructions from kube api server and manages the server to make sure containers are running on nodes as expected.
* ***KUBE-Proxy -*** Helps in enabling communication between services in clusters.
* ***Container Runtime*** - Underline Software used to run Containers like Docker.

#### 

#### 

#### **Yaml File Configuration**

* Top Level / Root Level Properties

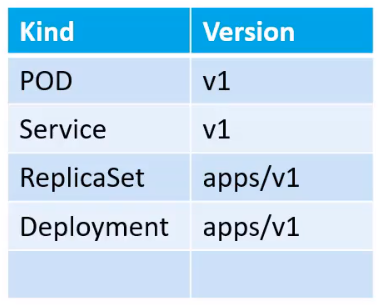
apiserver:

kind:

metadata:

specs:

* Types of api version according to kind



* Deployment File:

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

#*namespace: dev --optional*

labels:

app: myapp

type: front-app

spec:

template:

metadata:

name: myapp-pod

labels:

app: myapp

type: front-app

spec:

containers:

- name: nginx-container

image: nginx

replicas: 2

selector:

matchLabels:

type: front-app

#### **Basics of Kubernetes**

* *To get information about* ***cluster*** *-* kubectl get cluster-info
* *To Deploy Application on Cluster -* kubectl run nginx --image=nginx
* *To get* ***pods*** *information in detail like on which node and IP*

kubectl get pods -o wide

* *To Describe any pods details of used image*

kubectl describe pod <*pod.name*> -n <*namespace*> | grep -i image

* *To Save pod to be created configuration without creating pod*

kubectl run redis --image=redis --namespace=dev --dry-run -o yaml > pod.yml

* *To edit and create running pods yaml file*  
  kubectl get pod <*pod.name*> -o yaml > pod1.yml
* *To edit Pods Property -* kubectl edit pod <*pod.name*> -n <*namespace*>
* *To get available* ***replicasets*** *details -* Kubectl get replicaset
* *To scale replicas  
  - by edit .yml for replicaset -* kubectl replace -f replicaset.yml

*- can scale directly replicaset yml, this will not change replicas in yml file.*

kubectl scale --replicas=4 -f replicaset.yml  
*- can scale directly using replicaset name, this will not change replicas in yml file.*

kubectl scale --replicas=4 -f replicaset myapp-rs

* *To describe Replicaset -* Kubectl describe replicaset myaap-rs
* *To create and get available* ***deployment*** *and replicaset as well*

kubectl create -f deployment.yml && kubectl get deployment && kubectl get replicaset

* *To create deployment directly with command*

kubectl create deployment http-frontend --image=httpd:2.4-apline

Kubectl scale deployment http-frontend --replicas=3

* *To Create* ***namespace***- Kubectl create namespace dev
* *To set ns permanently* -   
  kubectl config set-context $(kubectl config current-context) --namespace=dev
* *To get resources in all namespace -* kubectl get pods --all-namespaces

| Quota in Namespaces | Connecting db service in same ns and in different ns. |
| --- | --- |

**--dry-run**: By default as soon as the command is run, the resource will be created. If you simply want to test your command , use the **--dry-run=client** option. This will not create the resource, instead, tell you whether the resource can be created and if your command is right.

**-o yaml**: This will output the resource definition in YAML format on screen.

To get Syntex of PODs yaml file if syntax is not clear:-  
***Kubectl explain pods --recursive | less***

**POD**

***Create an NGINX Pod***

kubectl run nginx --image=nginx

***Generate POD Manifest YAML file (-o yaml). Don't create it(--dry-run), adding labels***

kubectl run redis --image=redis --labels=tier=db --dry-run -o yaml

**Deployment**

***Create a deployment***

kubectl create deployment --image=nginx nginx

***Generate Deployment YAML file (-o yaml). Don't create it(--dry-run)***

kubectl create deployment --image=nginx nginx --dry-run -o yaml

***Generate Deployment with 4 Replicas***

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

***You can also scale a deployment using the kubectl scale command.***

kubectl scale deployment nginx --replicas=4

***Another way to do this is to save the YAML definition to a file and modify***

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

You can then update the YAML file with the replicas or any other field before creating the deployment.

**Service**

***Create a Service named redis-service of type ClusterIP to expose pod redis on port 6379***

* kubectl expose pod redis --port=6379 --name redis-service --target-port 6379 --dry-run=client -o yaml

*(This will automatically use the pod's labels as selectors)*

**OR**

* kubectl create service clusterip redis --tcp=6379:6379 --dry-run=client -o yaml

*(This will not use the pods labels as selectors, instead it will assume selectors as app=redis. You cannot pass in selectors as an option. So it does not work very well if your pod has a different label set. So generate the file and modify the selectors before creating the service)*

**Create a Service named nginx of type NodePort to expose pod nginx's port 80 on port 30080 on the nodes**:

* kubectl expose pod nginx --port=80 --name nginx-service --type=NodePort --dry-run=client -o yaml

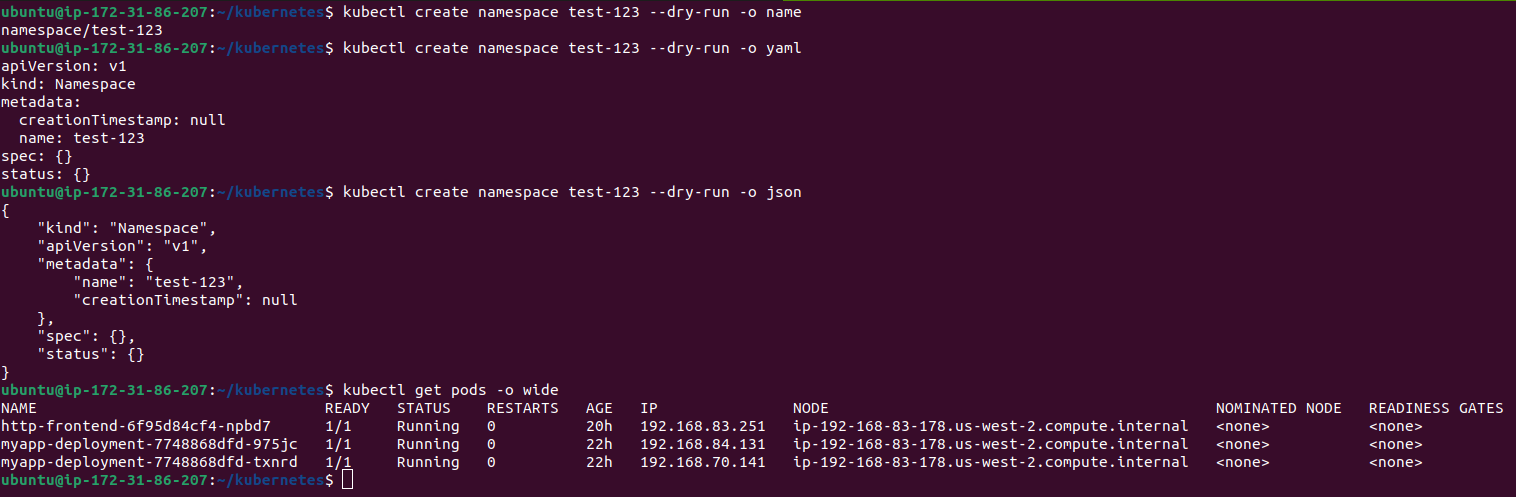
*(This will automatically use the pod's labels as selectors, but you cannot specify the node port. You have to generate a definition file and then add the node port manually before creating the service with the pod.)*

**OR**

* kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml *(This will not use the pods labels as selectors)*

***To Create Pod of name httpd & expose to port 80 with service name httpd type ClusterIP***

* *Kubectl run httpd –image=httpd:alpine --port 80 --expose*
* -o jsonOutput a JSON formatted API object.
* -o namePrint only the resource name and nothing else.
* -o wideOutput in the plain-text format with any additional information.
* -o yamlOutput a YAML formatted API object.



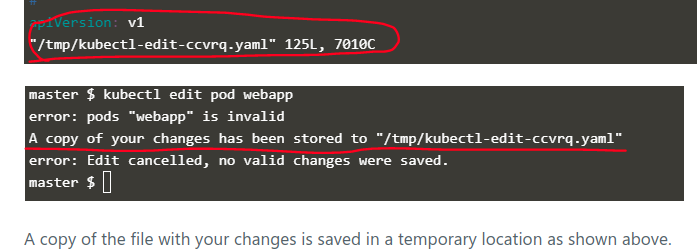
**Edit Pod**:-

**Option 1**

kubectl edit pod nginx

*We can not edit pod’s all fields, and it will popup error after saving the file, but file will also be saved under* ***temp directory /tmp/nginx-pod.yml***

So we have to delete the pod and have to create a new pod using a new yml file with required changes available in the tmp directory.



**Option 2**

kubectl get pod nginx -o yaml > nginx-pod.yaml

Then we can make changes to the extracted yaml file and then again we have to delete the not required pod and have to create a new pod using the saved yaml file.

vi nginx-pod.yaml

Kubectl delete pod nginx-pod

kubectl create -f nginx-pod.yaml

**Option 3 - If we are asked to edit pod which is the part of deployment then we can edit deployment and can edit all the aspects of pod**

Kubectl edit deployment nginx-deployment

#### 

#### **Dockerfile & Kubernetes yaml**

* ***DockerFile Simple***:- it will run command sleep & container will sleep for 5 sec on start

FROM Ubuntu

CMD sleep 5 OR CMD [“sleep”, “5”]

*Command to build new image and then to run using the newly created image*

docker build -t ubuntu-server .

Docker run ubuntu-server

**If we want to change sleep seconds - docker file configuration will be as below:-**

| **Using default dockerfile** | **Entrypoint** without specify cmd | **Modify Entrypoint** |
| --- | --- | --- |
| FROM Ubuntu  CMD sleep 5  **docker run ubuntu-sleeper sleep 10** | FROM Ubuntu  ENTRYPOINT [“sleep”]  # 5 Seconds as default  CMD [“5”]  **docker run ubuntu-sleeper 10** | FROM Ubuntu  ENTRYPOINT [“sleep”]  # 5 Seconds as default  CMD [“5”]  **docker run --entrypoint sleep2.0 ubuntu-sleeper 10** |

To achieve above in kubernetes, then pod definition file will be as:-

| **Using default dockerfile** | **Entrypoint** without specify cmd | **Modify Entrypoint** |
| --- | --- | --- |
| apiVersion: v1  kind: Pod  metadata:  name: ubuntu-sleeper-pod  spec:  containers:  -name: ubuntu-sleeper  image: ubuntu-sleeper | apiVersion: v1  kind: Pod  metadata:  name: ubuntu-sleeper-pod  spec:  containers:  -name: ubuntu-sleeper  image: ubuntu-sleeper  command: [“sleep”,”5”] OR  command:   * “sleep” * “5” | apiVersion: v1  kind: Pod  metadata:  name: ubuntu-sleeper-pod  spec:  containers:  -name: ubuntu-sleeper  image: ubuntu-sleeper  command: [“sleep2.0”]  args: [“10”]  *Argument Example:-*  args: [“--color=green”] |

**We can describe pods to check which commands ran during pod/container creation.**

#### **Environment Variables**

**In Docker**

docker run -e APP\_COLOR=pink simple-web-color

**In Kubernetes**

apiVersion: v1  
kind: Pod

metadata:

name: simple-web-color-pod

spec:

containers:

- name: simple-web-color

image: simple-web-color

ports:

- containerPort: 8080

env:

- name: APP\_COLOR

value: pink

#### **Configuration**

#### **ConfigMap** - used to store values of multiple environment variables in the form of key value pairs.

kubectl create configmap <config-name> --from-literal=<key>=<value>

* kubectl create configmap \ app-config --from-literal=APP\_COLOR=pink
* kubectl create configmap \ app-config --from-literal=APP\_COLOR=blue \ --from-literal=APP\_MOD=prod
* kubectl create configmap \ mysql-config --from-literal=port=3306 \ max\_allowed\_packet=128M

Kubectl create configmap <config-name> --from-file=<path-to-file>

* kubectl create configmap app-config --from-file=app\_config.properties

***config-map.yaml***

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

APP\_COLOR: pink

APP\_MODE: prod

kubectl create -f config-map.yaml

kubectl get configmaps or kubectl get cm

kubectl describe configmaps *app-config*

Defining configmaps in pod yaml file

***pod-defination.yaml***

apiVersion: v1

Kind: Pod

metadata:

name: simple-web-color

labels:

name: simple-web-color

spec:

containers:

- name: simple-web-color

image: simple-web-color

ports:

- containerPort: 8080

envFrom:

- configMapRef:

name: app-config

OR

env:

- name = APP\_COLOR

valueFrom:

configMapKeyRef:

name: app-config

key: APP\_COLOR

#### **Secret** - used to store sensitive information.

kubectl create secret generic <secret-name> --from=literal=<key>=<value>

* Kubectl create secret generic aap-secret --from=literal=DB\_Host=mysql \

--from=literal=DB\_User=root

--from=literal=DB\_Password=paswrd

kubectl create secret generic <secret-name> --from-file=<path-to-file>

* Kubectl create secret generic app-secret --from-file=app\_secret.properties

DB\_Host: mysql

DB\_User:root

DB\_Password:paswrd

* Creating secret using yaml file

***secret-data.yaml***

| Hard Coded Secret Values | Encode Secret Values | Encoded Secret Values |
| --- | --- | --- |
| apiVersion: v1  kind: Secret  metadata:  name: app-secrete  data:  DB\_Host: mysql  DB\_User: root  DB\_Password: password | echo -n ‘mysql’ | base64  bXlzcWw=  echo -n ‘root’ | base64  cm9vdA==  echo -n ‘paswrd’ | base64  4oCYcGFzd3Jk  ***Decode***  echo -n ‘cm9vdA==’ | base64 --decode => root | apiVersion: v1  kind: Secret  metadata:  name: app-secrete  data:  DB\_Host: bXlzcWw=  DB\_User: cm9vdA==  DB\_Password: 4oCYcGFzd3Jk |

kubectl create -f secret-data.yaml

kubectl get secrets

kubectl describe secrets app-secrete #*it will not show encoded secret value*

kubectl get secrets app-secrete -o yaml #*it will show encoded secret value*

* Defining Secret in pod yaml file

***pod-defination.yaml***

apiVersion: v1

Kind: Pod

metadata:

name: simple-webapp-color

labels:

name: simple-webapp-color

spec:

containers:

- name: simple-webapp-color

image: simple-webapp-color

ports:

- containerPort: 8080

envFrom:

- secretRef:

name: app-secret

OR

env:

- name = DB\_Host

ValueFrom:

secretKeyRef:

name: app-secret

key: APP\_COLOR

* **Security Context** - to Limit Permission of containers like to restrict or provide access particular to Kill processes. TO edit Linux Capability to add and remove from the container.

***Docker***:-

docker run --user=1001 ubuntu sleep 36000, running image   
To run particular process with user 1001 instead of Root (By Default)

docker run --cap-add MAC\_ADMIN ubuntu

Provided container with linux capability of MAC\_ADMIN

***Kubernetes***:-

***pod-defination.yaml***

apiVersion: v1

Kind: Pod

metadata:

name: web-prod

spec:

containers:

- name: ubuntu

image: ubuntu

command: [“sleep”.”3600”]

securityContext :

runAsUser: 1001

capabilities:

add: [“MAC\_ADMIN”]

#### **Resource** - It has to be set for each container with in the pod For memory it kills/terminates pod For CPU it throttles and went to no schedule

apiVersion: v1

kind:Pod  
metadata:

name: web-prod

labels:

name: web-prod

spec:

containers:

- name: web-prod

Image: web-prod

ports:

* containersPort: 8080

resources:

requests:

memory: “1Gi”

cpu: 1

limits:

memory: “2Gi”

cpu: 2

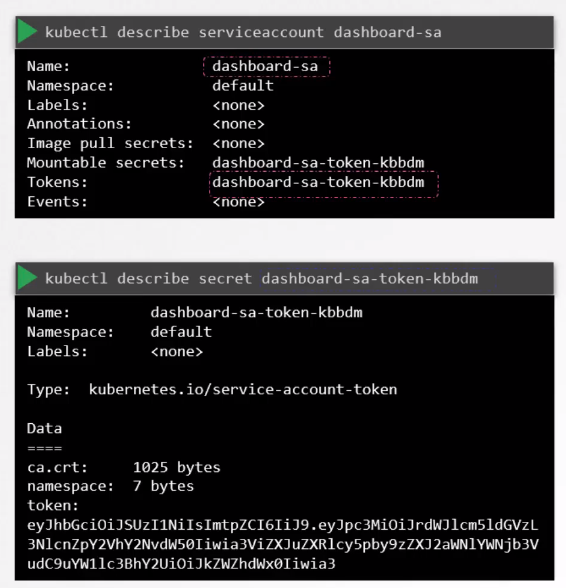
#### **ServiceAccount**

Used for providing access to the third party applications through api-servers.

Kubectl create serviceaccount dashboard-sa

Kubectl get serviceaccount

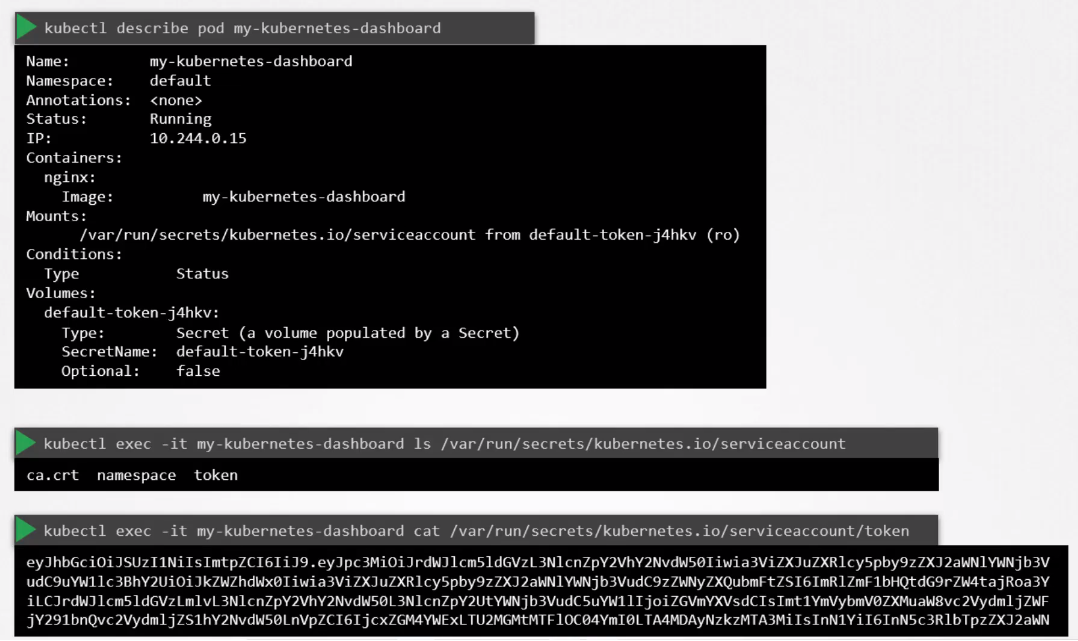
Kubectl describe serviceaccount dashboard-sa

****

* Default service account mounted to all the pods by default.   
  Service account creates secret token and mount to pod.

**kubectl exec -ti my-kubernetes-dashboard ls /var/run/secrets/kubernetes.io/serviceaccount**

**kubectl exec -ti my-kubernetes-dashboard cat /var/run/secrets/kubernetes.io/serviceaccount/token**



Mounting serviceaccount to the pod definition file:-

pod-defination.yml

apiServer: v1

kind: Pod

metadata:

name: my-kubernetes-dashboard

spec:

containers:

* name: my-kubernetes-dashboard

image: my-kubernetes-dashboard

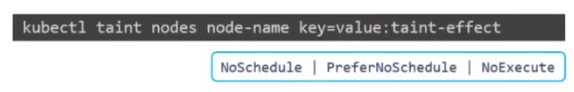
serviceAccountName: dashboard-sa

kubectl describe pod my-kubernetes-dashboard

If we add and change a security account after the pod is already in running state, we need to terminate that pod to implement a new ServiceAccount, but for the Deployment it updates using rollout deployment.

#### **Taints & Tolerance** - It can only restrict pods to not schedule having no tolerance but it can not ensure pods to be scheduled on tainted nodes only pods having tolerance. They can be scheduled on nodes having no taints.

**Taint - Nodes -** *Kubectl taint nodes node1 app=blue:NoSchedule*

****

*NoSchedule*:- It will not schedule pod’s who don't support node’s taint having tolerance.

*PreferNoSchedule*:- Will try not to schedule pods on nodes, but not guarantee.

*NoExecute*:- Pods - Terminate which are already running on nodes having no tolerance.

**Tolerance - Pods**

***<bee.yaml>***

apiServer: v1

kind: Pod

metadata:

Name: bee

spec:

containers:

- name: nginx-controller

Image: nginx

tolerance:

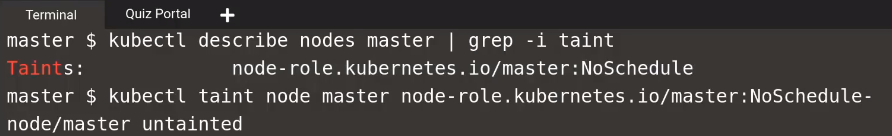
- key: “app”

operator: “=”

value: “blue”

effect: “NoSchedule”

Kubectl create -f bee.yaml

Kubectl describe node master | grep -i Taint  
*To* ***untaint*** *any node*  
Kubectl taint node master node-role.kubernetes.io/master:NoSchedule-

#### **Node Selector**

apiVersion: v1

kind:Pod  
metadata:

name: web-prod

spec:

containers:

- name: data-processor

Image: data-processor

nodeSelector:

Size: Large   
  
*We have to mention labels on nodes while creating node group or we can label with command:-*

Kubectl label nodes <node-name> <label-key>=<label-value>  
eg:- *kubectl label nodes node1 size=Large*

#### **Node Affinity**

apiVersion: v1

kind:Pod  
metadata:

name: web-prod

spec:

containers:

- name: data-processor

Image: data-processor

affinity:

nodeAffinity:

RequiredDuringSchedulingIgnoredDuringExecution:

nodeSelctorTerms:

- matchExpressions

- key: Size

operator: In

values:

- Large

- Medium

Some of the examples of node affinity and nodeSelector Terms, Use Cases:-

**nodeAffinity:**

requiredDuringSchedulingIgnoredDuringExecution: -   
*Must have labels, otherwise no pod schedule. If Scheduled then no issue after changes.*

preferedDuringSchedulingIgnoredDuringExecution: -

*Labeled Nodes Preferred, no issue once scheduled*

requiredDuringSchedulingRequiredDuringExecution: -

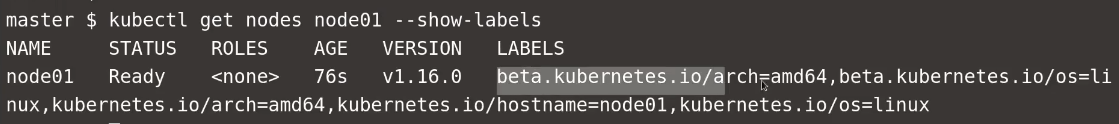
*Must have labels, otherwise no pod schedule. If Scheduled pod will terminate after changes.*

**nodeSelctorTerms:**

| nodeSelctorTerms:  - matchExpressions  - key: Size  operator: In  values:  - Large  - Medium | nodeSelctorTerms:  - matchExpressions  - key: Size  operator: Not In  values:  - Small | nodeSelctorTerms:  - matchExpressions  - key: Size  operator: Exists |
| --- | --- | --- |

To check Available Labels on Nodes:-

Kubectl describe node node1 --show-labels



To Label the node:-

Kubectl label node node01 color=blue

#### **Multi Container Pods**

apiVersion: v1

kind:Pod  
metadata:

labels:

name: app

name: app

namespace: elastic

spec:

containers:

- name: app

image: elastic

**volumeMounts:**

- mountPath: /log

name: log-volume

- name: sidecar

image: sidecar\_image

**volumeMounts**:

- mountPath: /var/log/event

name: log-volume

**volumes**:

- hostPath:

path: /var/log/webapp

type: DirectoryOrCreate

name: log-volume

#### **Observability**

#### **Readiness Probes**

apiVersion: v1

kind:Pod  
metadata:

name: simple-webapp

spec:

containers:

- name: simple-webapp

Image: simple-webapp

ports:

- containerPort: 8080

**readinessProbe:**

httpGet:

path: /api/ready

port: 8080

initialDelaySeconds: 10 # FOR WARMUP

periodSeconds: 5 # Duration after each probe to run

failureThreshold: 8 # By default 3 attempts

***OR***

**readinessProbe:**

tcpSocket:

port: 3306

***OR***

**readinessProbe:**

exec:

command:

- cat

- /app/is\_ready

#### **Liveness Probes**

apiVersion: v1

kind:Pod  
metadata:

name: simple-webapp

spec:

containers:

- name: simple-webapp

Image: simple-webapp

ports:

- containerPort: 8080

**livenessProbe:**

httpGet:

path: /api/ready

port: 8080\

initialDelaySeconds: 10 # FOR WARMUP

periodSeconds: 5 # Duration after each probe to run

failureThreshold: 8 # By default 3 attempts

***OR***

**livenessProbe:**

tcpSocket:

port: 3306

***OR***

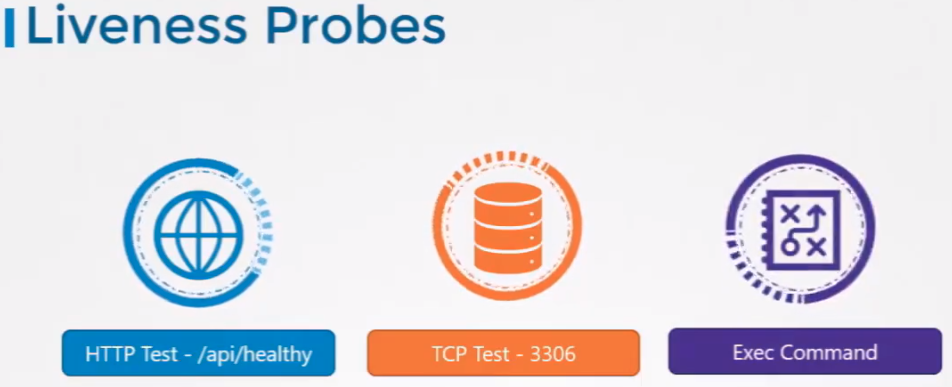
**livenessProbe:**

exec:

command:

- cat

- /app/is\_ready



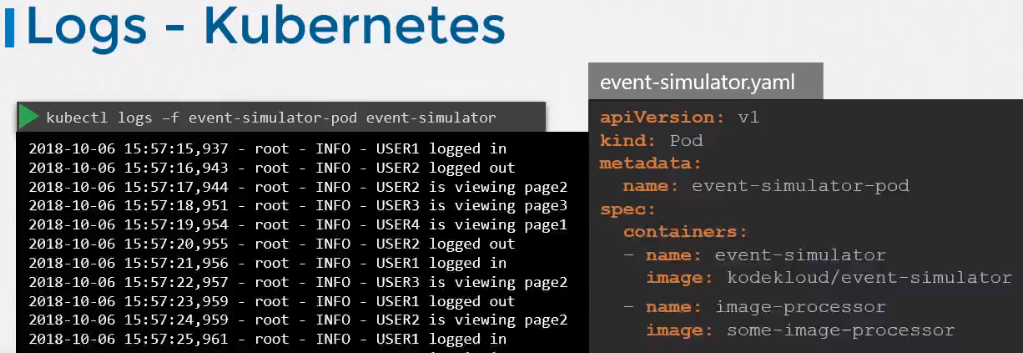
#### **Logging**

*If* ***single*** *container is running inside a POD*

**kubectl logs -f event-simulator-pod**

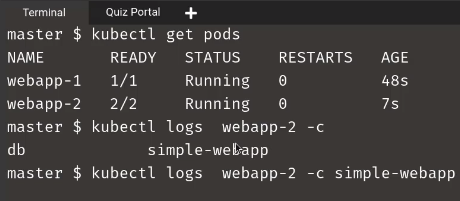
*If* ***Multiple*** *Containers are running then specify name of container whose logs are needed*

**kubectl logs [pod-name] --all-containers=true**

****

***OR***

**kubectl logs webapp-2 -c simple-webapp**

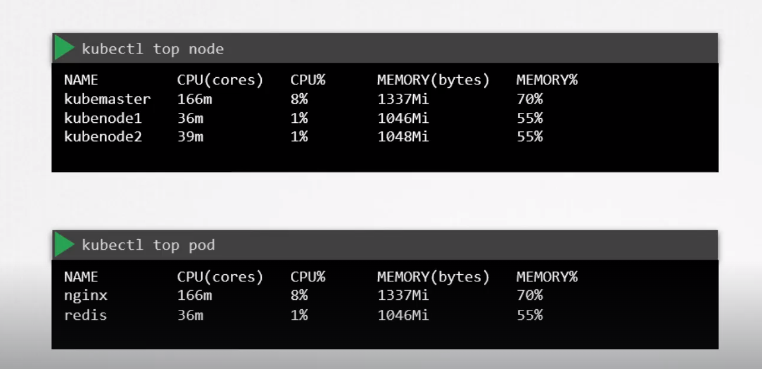
****

#### **Monitoring** - watch “kubectl get node” #keep on showing latest value for ***top***

Metric Server is being used for monitoring Pods, Nodes and clusters.  
Before heapster was used but metric server is the slim down version of heapster.   
Metric Server stores data in Memory so historical data can not be retrieved.

After deploying metric server we can use below commands:

*Kubectl top node* **OR***Kubectl top pod*



### **POD Design**

#### **Labels, Selectors & Annotation**

| ***POD\_DEFINATION YAML***  apiVersion: v1  kind: Pod metadata:  name: simple-webapp  labels: ***# we are defining labels on POD***  app = App1  function = Front-end  spec:  containers:  name: simple-webapp  image: simple-webapp  ports:  - containerPort: 8080 | ***REPLICASET\_DEFINATION YAML***  apiVersion: apps/v1  kind: ReplicaSet metadata:  name: simple-webapp  labels: ***# Labels for Replicasets, only used if we want other objects to discover this replica set.***  app = App1  function = Front-end  spec:  replicas: 3 **# Selector for POD to be discovered by replicaset**  selector:  matchLabels: ***# matching labels with pod’s***  app: App1  ***OR / Both***  function: Front-end  template: ***# Labels for the PODs***  metadata:  labels:  app: App1  function: Front-end  spec:  containers:  name: simple-webapp  image: simple-webapp  ports:  - containerPort: 8080 | ***SERVICE\_DEFINATION YAML***  apiVersion: v1  kind: Service metadata:  name: my-service  spec:  **# Selector for POD to be discovered by Service**  selector:  app: App1  ports:  - protocol: TCP  port: 80  targetPort: 9376  **ANNOTATION**  ***Use- Contact / Information purpose***  ***POD\_DEFINATION YAML***  apiVersion: v1  kind: Pod metadata:  name: simple-webapp  **annotation:**  **buildversion: 1.34**  labels: ***# we are defining labels on POD***  app = App1  function = Front-end  spec:  containers:  name: simple-webapp  image: simple-webapp |
| --- | --- | --- |

Kubectl get pods --show-labels

Kubectl get pods -l env=dev **OR** Kubectl get pods -l env=dev --no-headers | wc -l *# wordcount*  
Kubectl get pods --selector app=App1

Kubectl get all -l env=prod --no-headers | wc -l

Kubectl get pods -l env=dev,bu=finance,tier=frontend --no-headers | wc -l

#### **Rolling Updates & Rollback Deployments**

<https://legacy.kodekloud.com/courses/438996/lectures/14261742>

***It actually creates new replicaset and make old replicaset counts to 0, 1 by 1 and vice-versa for rollback.***

***Deployment Types****:- (By Default Kubernetes uses Rollback deployments)*kubectl apply -f deployment-defination.yml  
*#after doing changes in the deployments file.*

kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1

*#but it will not change anything in the deployment file.*  
***Rollback***  
kubectl rollout **undo** deployment/myapp-deployment  
*#to shift back to the previous deployment.*  
kubectl rollout **status** deployment/myapp-deployment  
*#to check the status of rollout deployment.*  
kubectl rollout **history** deployment/myapp-deployment

*#to check the history of rollouts.*

kubectl create -f deployment-defination.yml

kubectl get deployment

#### **Jobs and CronJobs -**

[*https://kubernetes.io/docs/concepts/workloads/controllers/job/*](https://kubernetes.io/docs/concepts/workloads/controllers/job/)

*Workloads meant to live for a short period of time - means to carry out specific tasks and then finish, like processing images, analytics, batch processing.*

| ***JOB\_DEFINATION YAML***  apiVersion: batch/v1  kind: Job metadata:  name: random-error-job  spec:  **completions**: 3 **#3 pods for jobs 3 successes.**  **parallelism**: 3 **#run 3 pods parallelly**  backoffLimit: 25  template:  spec:  containers:  name:  image:  **command**: [‘expr’, '3’, ‘+’, ‘2’]  **restartPolicy:** Never **# it will not restart the container after job completion** | * kubectl create -f job-defination.yml * kubectl get jobs * watch “kubectl get jobs” * kubectl create job random-job --image nginx-job --dry-run =client -o yaml > job.yaml * kubectl explain job --recursive | less |
| --- | --- |

| ***CRON\_JOB\_DEFINATION YAML***  apiVersion: batch/v1  kind: CronJob  metadata:  name: reporting-cron-job  spec:  **schedule**: "\*/1 \* \* \* \*"  **jobTemplate**:  spec:  **completions**: 3 **#3 pods for jobs 3 successes.**  **parallelism**: 3 **#run 3 pods parallelly**  backoffLimit: 25  template:  spec:  containers:  name:  image:  **command**: [‘expr’, '3’, ‘+’, ‘2’]  **restartPolicy:** Never **# it will not restart the container after job completion** | kubectl create cronjob throw-dice-cron-job --image random --schedule “30 21 \* \* \*” --dry-run=client -o yaml > cron.yaml |
| --- | --- |

#### **Services and Networking**

#### **Services** - **Kubectl get services**

**NodePort**:- Service which makes internal pod accessible on a port on the node.

**ClusterIP**:- service creates virtual ip inside the cluster to enable communication between different services, e.g. set of front-end servers and back-end servers.

**LoadBalancer**:- service which exposes applications to external load balancers like aws.

***SERVICE\_NodePort***  
apiServer: v1

kind: Service

metadata:

name: myapp-service

spec:

type: NodePort

ports:

* targetPort:80

port: 80

nodePort: 30008

selector:

app : app1

type: Front-end

* *If we don’t Mention nodeport it will take any free port between range 30000-32767*
* *If We don't provide a target port, it will assume the target port is the same as port.*

|  |  |
| --- | --- |

#### 

#### **Network Policies** - Traffic Flow.

* Ingress Traffic - Request from user to web-server on port 80, request from web-server to api server on port 5000, request from api server to db on port 3306.
* Egress Traffic - outgoing request to the app server is an egress traffic, outgoing request from api server to db.

|  |  |
| --- | --- |

| Ingress Traffic Network Policy | Ingress and Egress Traffic Network Policy |
| --- | --- |
|  |  |

**Network Policy useful Command:-**

| Kubectl create -f policy-defination.yml kubectl get netpol #to list available network policies  kubectl get pods -l name=payroll  kubectl describe netpol payroll-policy |  |
| --- | --- |

Below policy is used to allow ingress/egress traffic to db pods form api pods only and from db ports to backupserver.

**policy-definition.yml**  
apiVersion:

kind: NetworkPolicy

metadata:

name: db-policy

spec:

podSelector:

matchLabels:  
 role: db #assigning policy to pods having label as role: db

policyTypes: #Policy for ingress & egress from pods with label role: db label.

- Ingress

- Egress

ingress: #allowing traffic **to** db pods from **api pods & namespace:- prod.**

- from:

- podSelector: #from api-pods only

matchLabels:  
 name: api-pod

namespaceSelector: #api-pods from namespace prod only

matchLabels:

name: prod

- ipBlock: #**OR** allowing traffic from IP Block

cdir: 192.31.4.10/32 #select IP Block Range selector to limit ingress traffic to db pods

ports: #allowing ingress through specific port

- protocol: TCP

port: 3306

egress: #sending traffic **from** db pods to Backup Server from port 80.

- to:

- ipBlock: #sending traffic to IP Block

cdir: 192.31.4.10/32 #select IP Block Range selector to limit egress traffic from db

ports: #request to be sent to backup server from port specific port

- protocol: TCP

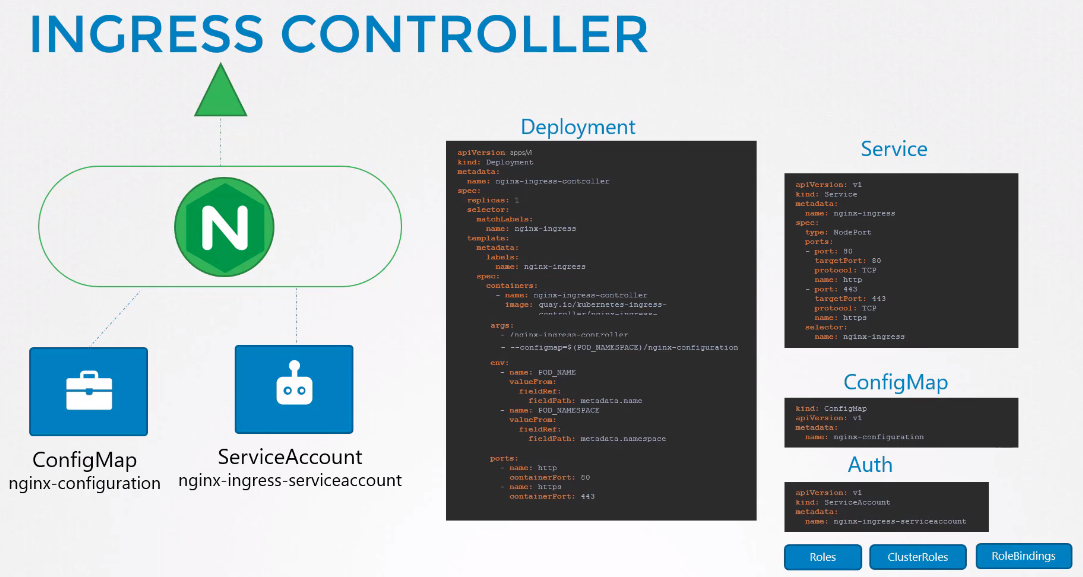
port: 80

#### **Ingress Networking**

Helps users to access applications using a single externally accessible url that can be configured to route different services within a cluster based on url path and at the same time implementing ssl certificate security as well.

***kubectl get ingress*** *#to get list of network resources*

***Ingress Controller***:- Layer7 Load Balancer (implemented as solution like Haproxy/Nginx)



***Ingress controller*** to be deployed -  
1. We need nginx-ingress-controller as a deployment, with match labels, with pod name and pod’s namespace to be attached with, along with the pods ports as shown in above image.

2. Then we need a service of type nodeport to expose application to an external world with a selector label name of nginx-ingress-controller deployment to attach service to it.

3. Then we need configmap to store data for the nginx ingress controller.

4. News a service account to provide access using Roles, ClusterRoles, RoleBindings.

***Ingress Resources***:- It is the set of Rules and configuration that binds with ingress controller.  
We can configure rules to forward all traffic to a single application or forward traffic to multiple applications according to the url path.

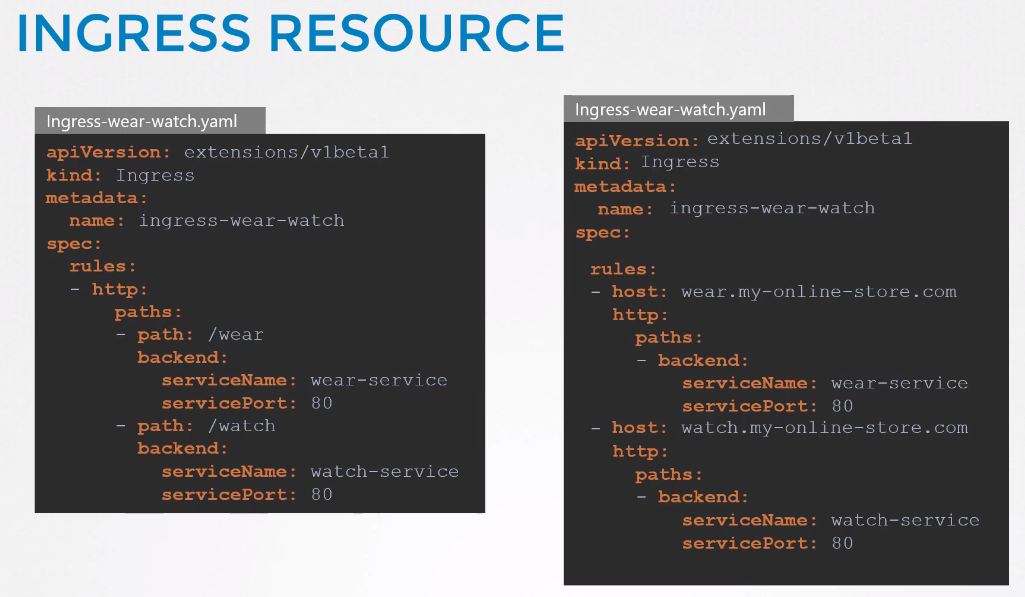
***Note:-***

1. We should deploy a default backend service so that traffic can be routed to it if traffic is on paths which are not mentioned ingress controller's resources.



1. If we don’t specify host filed it will consider it as **\*** means all the traffic will be routed through that particular rule without matching the host name.

*We can Route traffic by*:-

**Url path** called (1 Rule 2 path) and by **Host**/**Domain** Name called (2 Rule 1 path each)  


kubectl get deployments.apps --all-namespaces *#to list deployments in all namespaces*

kubectl get ingress --all-namespaces *#to list ingress in all namespaces*

*#to save existing ingress resource file to do changes*  
kubectl -n <*namespaces>* get ingress ingress-wear-watch -o yaml > ingress.yaml

*#to delete ingress resource*

kubectl -n <*namespaces>* delete ingress ingress-wear-watch

*#to save existing ingress resource file to do changes*

kubectl -n <*namespaces>* apply -f ingress.yaml

Documents for Ingress Resources

<https://legacy.kodekloud.com/courses/438996/lectures/13333428>

#### **State Persistence**

| **Docker** | **Kubernetes** |
| --- | --- |
| Docker uses ***Storage Driver*** to enable layered architecture  **Volume Mounting**  **-** By default saves volume under path ***/var/lib/docker/volumes***  **-** We can create volume using cmd:- ***docker volume create data\_volume***  **-** Then we can run the container by attaching the newly created volume.  - **Note:-** just in case we don’t have volume create, *docker run -v data\_volume2:/var/lib/mysql mysql* will create new volume with the name of data\_volume2 automatically and save under default path ***/var/lib/docker/volume***    **Bind Mounting**   * To mount any particular volume directory with a container like to mount /data/ directory to a container from the docker host. * We can the bind mount using below cmd:-   *Docker run -v /data/mysql:/var/lib/mysql mysql*  Docker ***Volume Driver*** to attach AWS EBS to container, If Container terminates it will not lose its data.  *Docker run -it --name mysql --volume-driver rexray/ebs --mount src=ebs-vol, target=/var/lib/mysql mysql* | Local host (in case of single node cluster) directory can be mounted on pod as below:-    To use aws ebs as persistence volume, we have to specify configuration under pod’s yaml file as below:-    This is not an efficient way to specific storage in each pod's configuration file, so we use persistent volume claims by users who are deploying pods. |

Persistent volumes Claims used to bind persistent volumes to claims based on requests in claim yaml request. <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#claims-as-volumes>

| kubectl create -f pvc-defination.yaml    kubectl get presistentvolumeclaim    kubectl delete presistentvolumeclaim my-claim | * Every PVC binds to a single persistent volume * PVC tries to find PV of the same match found to satisfy all the requests like - requested storage, access mode, volume modes, storage class. * If there are multiple PV matches PVC then we can use labels(PV) and selecters(PVC).      * If there are no PV available PVC will remain in pending state until new volumes are available.   **Policy** - Retain, Delete, Recycle. |
| --- | --- |
|  |  |