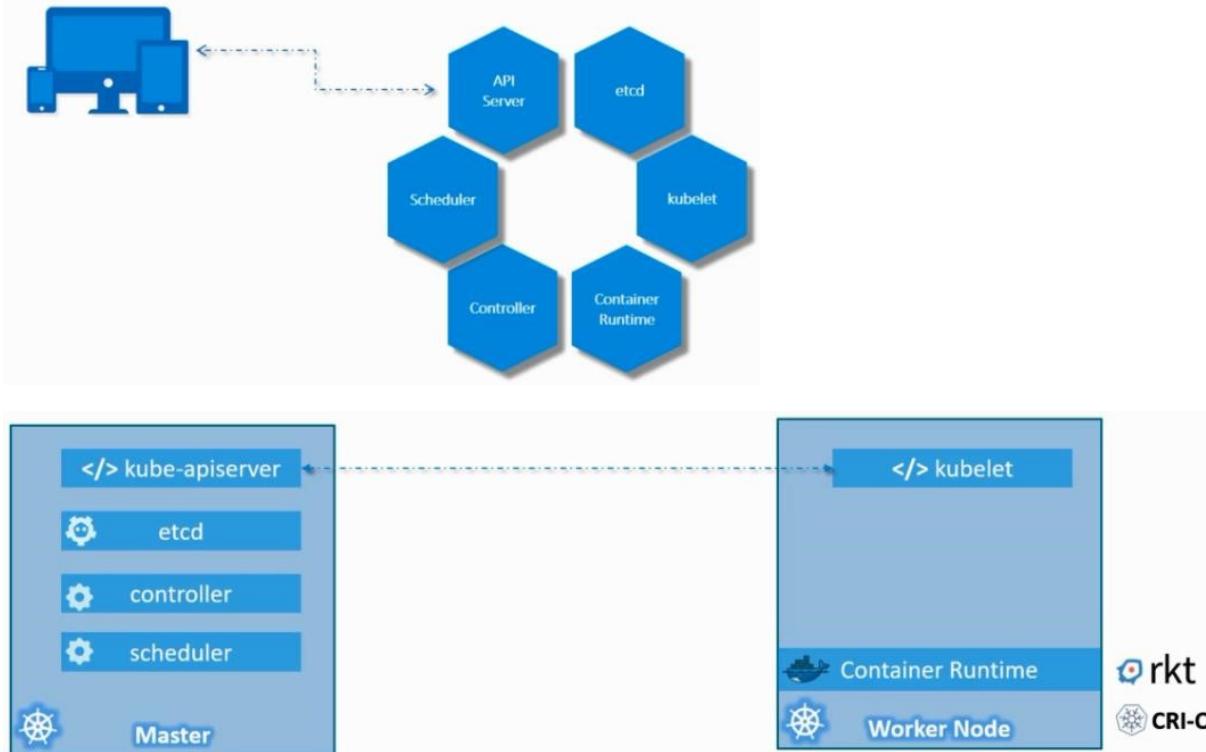


Core Concepts - 13%

1. Architecture



```
kubectl get nodes  
kubectl  
cluster-info
```

2. Pod:

```
kubectl run --generator=run-pod/v1 nginx-pod --image=nginx  
kubectl run --generator=run-pod/v1 nginx --image=nginx --port=8080 -l tier=db --env="DNS_DOMAIN=cluster" --dry-run -o yaml  
kubectl run hazelcast --image=nginx --env="DNS_DOMAIN=cluster" --env="POD_NAMESPACE=default" kubectl run hazelcast --  
image=nginx -l app=name  
kubectl run --generator=run-pod/v1 nginx --image=nginx --port=8080 --command echo hi kubectl  
create -f *.yaml  
kubectl edit pod pod-name - edit already running pod // Need to verify  
kubectl apply -f *.yaml kubectl get pod <podname> -o wide kubectl delete  
pod <podname> kubectl describe pod <podname> kubectl get pod podname -  
o yaml --export kubectl set image pod/nginx nginx=nginx:1.7.1  
kubectl exec podname -it bash or kubectl exec podname -it -- /bin/sh # get inside the pod kubectl  
exec -it nginx - env # to display env variables
```

Pod deleted automatically when its completed ** use --restart=Never

3. ReplicationController/ ReplicaSet kubectl run replicontrolr --generator=run/v1 --image=redis --replicas=2 --dry-run -o yaml //
replicationcontroller To generate file for replicaset -> generate deployment file and edit the file kubectl create -f *.yaml kubectl get
replicationcontroller kubectl get replicasesets

```
kubectl delete replicaset <rp-name> kubectl describe replicaset kubectl edit  
replicaset replicaset-name - edit already running replicaset
```

4. Scale

- 1. update replicas number in the file
kubectl replace -f file.yaml

```
2.      kubectl scale --replicas=6 -f  
file.yaml or  
kubectl scale --replicas=6 replicaset replicaset-name  
      kubectl scale --replicas=3 -f filename.yaml  
      kubectl scale --current-replicas=1 --replicas=3 deployment/mydeploy  kubectl scale --  
replicas=3 rc/rc1 rc/rc2 rc/rc3 # Multiple replication controller update  
<<this option wont update the replicas in the file>> kubectl  
edit rs rs-name
```

5. Deployment

```
kubectl create -f *.yaml --record kubectl  
run nginx --image nginx kubectl delete  
deployment dep-name  
kubectl run --generator=deployment/v1beta1 nginx --image=nginx  
Or kubectl create deployment --image=nginx nginx
```

```
kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run -o yaml  
Or kubectl create deployment --image=nginx nginx --dry-run -o yaml
```

kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml kubectl create deployment does not have a --replicas option. You could first create it and then scale it using the kubectl scale command.

```
kubectl run --generator=deployment/v1beta1 nginx --image=nginx --dry-run --replicas=4 -o yaml > nginx-deployment.yaml
```

Service: kubectl create service clusterip ngservice --tcp=80:80 --dry-run -o yaml | after creating change the selector to required pod
kubectl create service nodeport nginx --tcp=80:8000 --node-port=30080 --dry-run -o yaml
kubectl expose deployment nginx --type=NodePort --port=80 --target-port=8000 --name=nginx-serv --dry-run -o yaml
kubectl expose deployment nginx --port=80 --target-port=8000
// need to check with docs

namespace:

```
kubectl get all --all-namespaces  
kubectl exec -it app cat /log/app.log --namespace=elastic-stack  
kubectl create namespace mynamespace kubectl run nginx --  
image=nginx -n mynamespace
```

ResourceQuota: kubectl create quota myrq --hard(cpu=1,memory=1G,pods=2) --dry-run -o yaml

Multipod container: 10%

1. Many containers running in a single pod.

Create a pod with 2 container and run commands in the second container

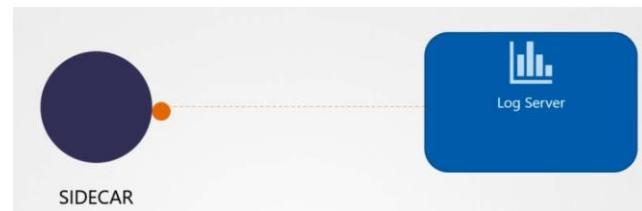
```
kubectl exec -it pod-name -c container-name-2 -- /bin/sh
```

It shares same network,volumes .

Ex: App+Log agent

Patterns : Side car , Adapter, ambassador

Side car:



Before sending log to the central server, we like to convert to the common format. For this we use **adaptor** pattern



Ambassador pattern:

We have multiple kind of DB depends on the dev,test,prod environment.

we have container that proxy the appropriate DB

Though we have 3 patterns, implementing them is always the same.

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp
  labels:
    name: simple-webapp
spec:
  containers:
    - name: simple-webapp
      image: simple-webapp
      ports:
        - containerPort: 8080
    - name: log-agent
      image: log-agent
```

POD DESIGN – 20%

Labels and Selector

When we have multiple pods, inorder to classify , separate -> Labels are used

Kubectl get pods –selector app=App1

Kubectl get all –selector app=app1,tier=db

Kubectl get pod –show-labels // to display all labels

```
Kubectl get pod -L app  
// capital L for only specifying Key .  
Kubectl get pod -l app=abc // small l for specify with key and value  
kubectl label po nginx1 nginx2 nginx3 app- // to remove app label from the  
pods kubectl label po nginx2 app=v2 -overwrite // Change the app label  
kubectl annotate po nginx1 nginx2 nginx3 description='my description' kubectl  
annotate po nginx1 nginx2 nginx3 description-  
kubectl autoscale deploy nginx --min=5 --max=10 --cpu-percent=80 --dry-run -o yaml Annotations:  
Annotations under metadata is to mention build number, contact details, build version etc
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: simple-webapp
  labels:
    app: Appl
    function: Front-end
  annotations:
    buildversion: 1.34
spec:
  replicas: 3
  selector:
    matchLabels:
      app: Appl
  template:
    metadata:
      labels:
        app: Appl
        function: Front-end
    spec:
      containers:
        - name: simple-webapp
          image: simple-webapp
```

Rolling Updates & Rollbacks in deployments

Rollout

Kubectl rollout status deployment/my-app-deployment

Kubectl rollout history deployment/my-app-deployment

When first creating deployment , following steps happen

1. New rollout

2. New revision

Update – change any config in the file and apply it

After changing deplymnt file , >>> kubectl apply -f deploy-file

Kubectl set image deployment/mydeploy nginx=nginx:1.9.1

Strategy -> Recreate and Rolling update / can be viewd from the below command

Kubectl describe deployment mydeploy

Events:				
Type	Reason	Age	From	Message
Normal	ScalingReplicaSet	1m	deployment-controller	Scaled up replica set myapp-deployment-6795844b58 to 5
Normal	ScalingReplicaSet	1m	deployment-controller	Scaled down replica set myapp-deployment-6795844b58 to 0
Normal	ScalingReplicaSet	56s	deployment-controller	Scaled up replica set myapp-deployment-54c7d6ccc to 5

Normal ScalingReplicaSet 1m deployment-controller Scaled up replica set myapp-deployment-67c749c58c to 5	Normal ScalingReplicaSet 1s deployment-controller Scaled up replica set myapp-deployment-7d57dbdb8d to 2
Normal ScalingReplicaSet 1s deployment-controller Scaled down replica set myapp-deployment-67c749c58c to 4	Normal ScalingReplicaSet 1s deployment-controller Scaled up replica set myapp-deployment-7d57dbdb8d to 3
Normal ScalingReplicaSet 0s deployment-controller Scaled down replica set myapp-deployment-67c749c58c to 3	Normal ScalingReplicaSet 0s deployment-controller Scaled up replica set myapp-deployment-7d57dbdb8d to 4
Normal ScalingReplicaSet 0s deployment-controller Scaled down replica set myapp-deployment-67c749c58c to 2	Normal ScalingReplicaSet 0s deployment-controller Scaled up replica set myapp-deployment-7d57dbdb8d to 5
Normal ScalingReplicaSet 0s deployment-controller Scaled down replica set myapp-deployment-67c749c58c to 1	

Recreate

RollingUpdate

Rollback

To undo the change run : kubectl rollout undo

deployment/mydeploy kubectl rollout undo

deploy nginx --to-revision=2

```
kubectl rollout history deploy nginx --revision=3 kubectl
rollout pause deploy nginx // to pause the rollout
kubectl rollout resume deploy nginx
```

Create

```
> kubectl create -f deployment-definition.yml
```

Get

```
> kubectl get deployments
```

Update

```
> kubectl apply -f deployment-definition.yml
```

Status

```
> kubectl rollout status deployment/myapp-deployment
```

Rollback

```
> kubectl rollout history deployment/myapp-deployment
```

```
> kubectl rollout undo deployment/myapp-deployment
```

Jobs

pods are designed to run all the time by k8s. So if it has a small process/less time helping task, even it is finished the task, it'll go to completed state. then replicaset restart the pod to make it's always running. To stop restarting we have to set restartPolicy: Never

```
kubectl create job busybox --image=busybox -- /bin/sh -c 'echo hello; sleep 30; echo world'
```

Parallelism

job-definition.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: random-error-job
spec:
  completions: 3
  parallelism: 3
  template:
    spec:
      containers:
        - name: random-error
          image: kodekloud/random-error
  restartPolicy: Never
```

▶ kubectl create -f job-definition.yaml

▶ kubectl get jobs

NAME	DESIRED	SUCCESSFUL	AGE
random-error-job	3	2	38s

▶ kubectl get pods

NAME	READY	STATUS	RESTARTS
random-exit-job-ktmitt	0/1	Completed	0
random-exit-job-sdsrf	0/1	Error	0
random-exit-job-wwqbn	0/1	Completed	0
random-exit-job-fkhfn	0/1	Error	0
random-exit-job-fvf5t	0/1	Error	0
random-exit-job-nmghp	0/1	Completed	0



Kubectl get jobs

Kubectl delete job job-name // deleting the job always delete the pod that was created by job

Like replicas -> we have completions for job to make sure it will run the mentioned number of time. By default jobs create one after another. In order to create more jobs simultaneously use parallelism.

CronJob : it is a job which can be scheduled

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

Wikipedia

```
kubectl run --generator=job/v1 --image=kodekloud/throw-dice throw-dice-job --restart=Never
```

```
kubectl run --generator=cronjob/v1beta1 --image=kodekloud/throw-dice throw-dice-cron-job --restart=Never -o yaml --dry-run -schedule="30
```

```
21 * * *"> cjb.yaml
```

```
or kubectl run --generator=cronjob/v2alpha1 once job created, if we edit the job
```

```
file -> its not working for some properties. You have to type completions and  
parallelism
```

```
backoffLimit: 25 # This is so the job does not quit before it succeeds
```

```
kubectl create cronjob busybox --image=busybox --schedule="*/1 * * * *" -- /bin/sh -c 'date; echo Hello from the  
Kubernetes cluster'
```

Create CronJob

```
kubectl create -f cron-job-definition.yaml
```

```
kubectl get cronjob
```

NAME	SCHEDULE	SUSPEND	ACTIVE
reporting-cron-job	*/1 * * * *	False	0

cron-job-definition.yaml

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: reporting-cron-job
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      completions: 3
      parallelism: 3
      template:
        spec:
          containers:
            - name: reporting-tool
              image: reporting-tool
  restartPolicy: Never
```

Configurations- 18%

Command and arguments Docker

commands:

Docker containers only lives as long as process running.

Once process inside container stops, the container go to exited state

Inside Docker file -> CMD[] it runs when the container starts

```
FROM Ubuntu
```

```
CMD sleep 5
```

```
▶ docker run ubuntu-sleeper sleep 10
```

Command at Startup: sleep 10

```
FROM Ubuntu
```

```
ENTRYPOINT ["sleep"]
```

```
▶ docker run ubuntu-sleeper 10
```

Command at Startup: sleep 10

Command line arg over rides the command in the file. But if we use Entry point it appends the command and argument from the terminal

If user not sending 10 as arguments, then it'll fail at the startup. So in order to give default value use CMD .

```
FROM Ubuntu  
  
ENTRYPOINT ["sleep"]  
  
CMD ["5"]
```

```
▶ docker run ubuntu-sleeper  
sleep: missing operand  
Try 'sleep --help' for more information.
```

Command at Startup: sleep 5

```
▶ docker run ubuntu-sleeper 10
```

Command at Startup: sleep 10

Use `--entry-point` when running docker command to over ride the entry point command.

K8s Command and arguments

```
FROM Ubuntu
```

```
ENTRYPOINT ["sleep"]
```

```
CMD ["5"]
```

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-sleeper-pod
spec:
  containers:
    - name: ubuntu-sleeper
      image: ubuntu-sleeper
      command: ["sleep2.0"]
      args: ["10"]
```

Over ride the Entry point command in the docker file → use command in k8s

Override the default CMD in docker file → use args in k8s

ENV Variables in Kuber

```
▶ docker run -e APP_COLOR=pink simple-webapp-color
```

pod-definition.yaml

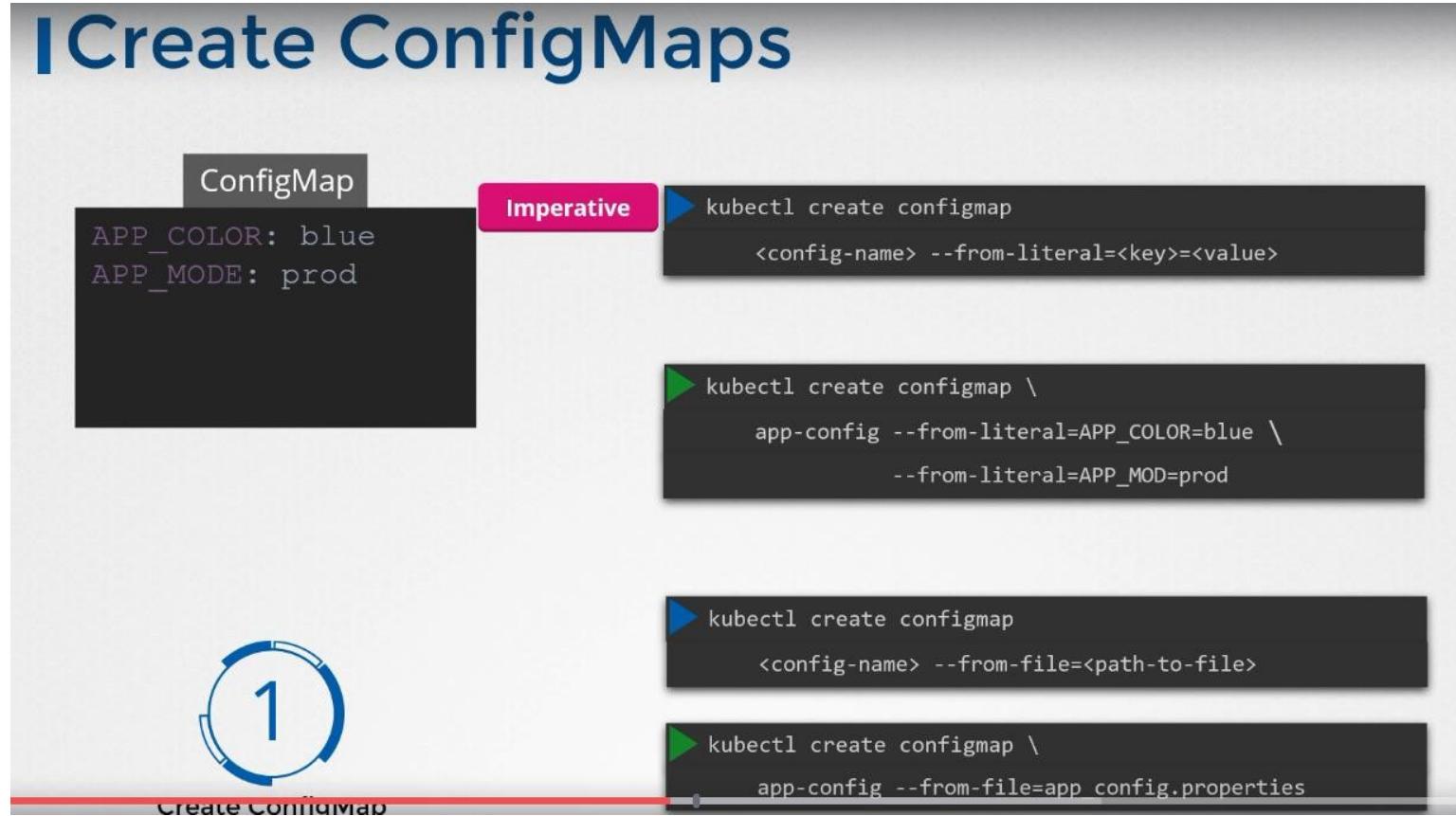
```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
    - containerPort: 8080
  env:
  - name: APP_COLOR
    value: pink
```

```
kubectl run --generator=run-pod/v1 nginx --image=nginx --port=8080 -l tier=db --env="DNS_DOMAIN=cluster" --dry-run -o yaml
```

ConfigMaps

When we have lot of pod files and difficult to manage environmental variable details. So Env variables can be taken out of the pod definition file and put it in a separate file called configmaps. Config maps are key value pair in k8s.

Kubectl create configmap app-config --from-literal=Name=abc --from-literal=Hname=karthi



Create ConfigMaps

ConfigMap Declarative `kubectl create -f`

APP_COLOR: blue
APP_MODE: prod

config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```

 Create ConfigMap

`kubectl create -f config-map.yaml`

app-config mysql-config redis-config

APP_COLOR: blue
APP_MODE: prod

port: 3306
max_allowed_packet: 128M

port: 6379
rdb-compression: yes

Kubectl get configmaps -> cmd

IConfigMap in Pods

pod-definition.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:
    - configMapRef:
        name: app-config
```

```
kubectl create -f pod-definition.yaml
```

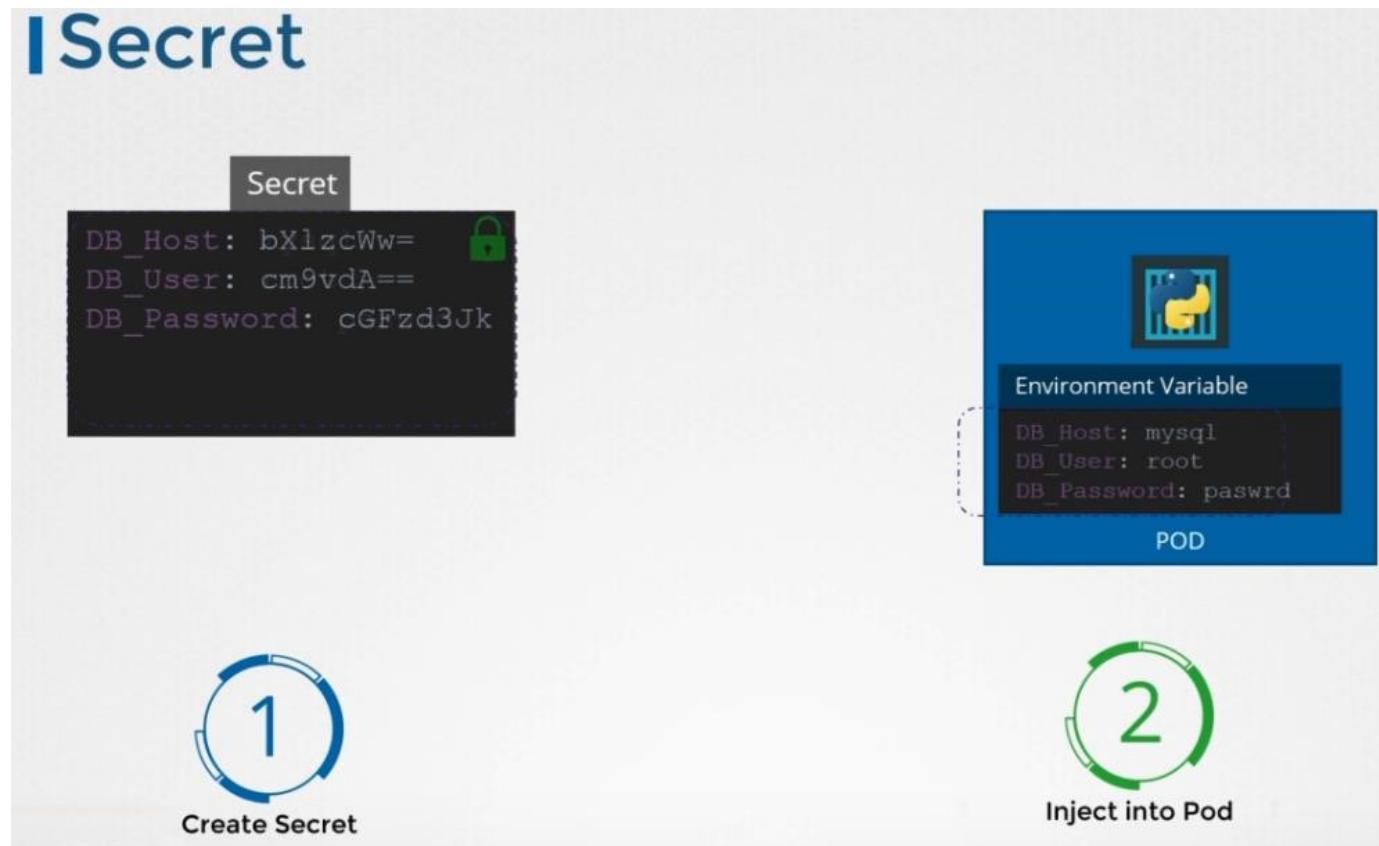
config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```

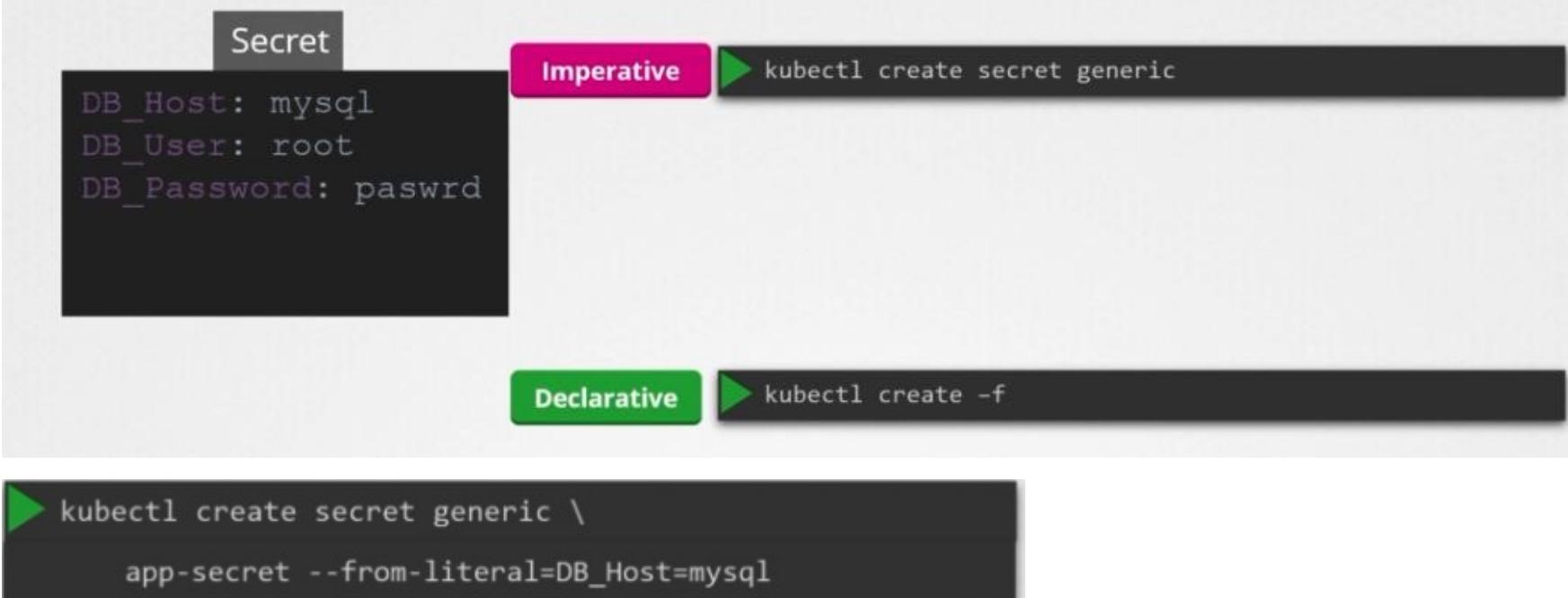


Secrets:

Secrets are used to store the sensitive informations, like password



Create Secrets



Kubectl create secret generic app-secret --from-literal=DB_HOST=mysql

Imperative

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

```
▶ kubectl create secret generic \  
    app-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
```

Secret Declarative `kubectl create -f`

`DB_Host: mysql
DB_User: root
DB_Password: paswrd`


Create Secret

`secret-data.yaml`

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bXlzcWw=
  DB_User: cm9vdA==
  DB_Password: cGFzd3Jk
```

`kubectl create -f secret-data.yaml`

`echo -n 'mysql' | base64
bXlzcWw=`

`echo -n 'root' | base64
cm9vdA==`

`echo -n 'paswrd' | base64
cGFzd3Jk`

```
kubectl get secrets
```

NAME	TYPE	DATA	AGE
app-secret	Opaque	3	10m
default-token-mvtkv	kubernetes.io/service-account-token	3	2h

```
kubectl describe secrets
```

```
Name:           app-secret
Namespace:      default
Labels:          <none>
Annotations:    <none>

Type:  Opaque

Data
=====
DB_Host:       10 bytes
DB_Password:   6 bytes
DB_User:        4 bytes
```

```
kubectl get secret app-secret -o yaml
```

```
apiVersion: v1
data:
  DB_Host: bXlzcWw=
  DB_Password: cGFzd3Jk
  DB_User: cm9vdA==
kind: Secret
metadata:
  creationTimestamp: 2018-10-18T10:01:12Z
  labels:
    name: app-secret
    name: app-secret
    namespace: default
  uid: be96e989-d2bc-11e8-a545-080027931072
type: Opaque
```

Decode the hash value

```
echo -n 'bXlzcWw=' | base64 --decode
```

```
mysql
```

1 Secrets in Pods

pod-definition.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
```

secret-data.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bXlzcWw=
  DB_User: cm9vdA==
  DB_Password: cGFzd3Jk
```



Inject into Pod

```
envFrom:  
  - secretRef:  
      name: app-config
```

ENV

SINGLE ENV

```
env:  
  - name: DB_Password  
    valueFrom:  
      secretKeyRef:  
        name: app-secret  
        key: DB_Password
```

```
volumes:  
- name: app-secret-volume  
  secret:  
    secretName: app-secret
```

VOLUME

Docker Security:

For a host -> root user || For a Docker container → root user

Security contexts

Pod level:

Security Context

```
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  securityContext:
    runAsUser: 1000

  containers:
    - name: ubuntu
      image: ubuntu
      command: ["sleep", "3600"]
```

Container level:

```
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
    - name: ubuntu
      image: ubuntu
      command: ["sleep", "3600"]
      securityContext:
        runAsUser: 1000
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
    - name: ubuntu
      image: ubuntu
      command: ["sleep", "3600"]
      securityContext:
        runAsUser: 1000
      capabilities:
        add: ["MAC_ADMIN"]
```

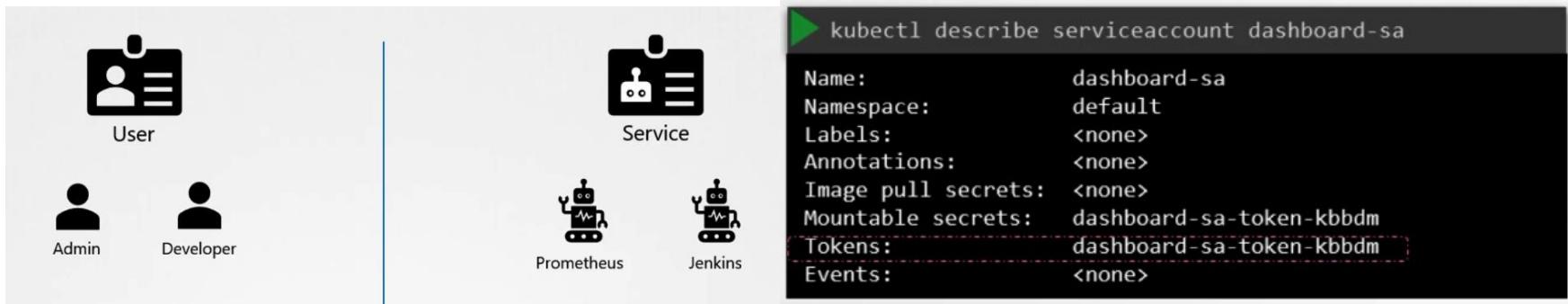


Note: Capabilities are only supported at the container level and not at the POD level

```
kubectl exec ubuntu-sleeper whoami
```

```
kubectl exec -it ubuntu-sleeper -- date -s '19 APR 2012 11:14:00'
```

Service Accounts:



```
▶ kubectl describe secret dashboard-sa-token-kbbdm
```

```
Name: dashboard-sa-token-kbbdm
```

```
Namespace: default
```

```
Labels: <none>
```

```
Type: kubernetes.io/service-account-token
```

```
Data
```

```
====
```

```
ca.crt: 1025 bytes
```

```
namespace: 7 bytes
```

```
token:
```

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcmtldGVzL  
3NlcnZpY2VhY2NvdW50Tiwiia3ViZXJuZXRlcw5nbv9zZXJ2aWN1YWNNib3V
```

```
▶ kubectl exec -it my-kubernetes-dashboard ls /var/run/secrets/kubernetes.io/serviceaccount
```

```
ca.crt  namespace  token
```

```
kubectl get serviceaccount  
NAME      SECRETS   AGE  
default    1          218d  
dashboard-sa 1          4d
```

pod-definition.yml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: my-kubernetes-dashboard  
spec:  
  containers:  
    - name: my-kubernetes-dashboard  
      image: my-kubernetes-dashboard  
  serviceAccount: dashboard-sa
```

```
pod-definition.yml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: my-kubernetes-dashboard  
spec:  
  containers:  
    - name: my-kubernetes-dashboard  
      image: my-kubernetes-dashboard  
  automountServiceAccountToken: false
```

You can replace the POD type with Deployment, then add your **serviceAccountName: default** under

```
1 | template:  
2 |   spec:  
3 |     serviceAccountName: default
```

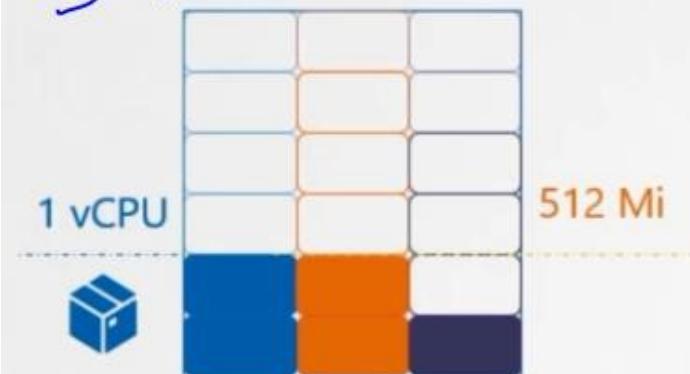
Resources Requirement:



We can also set limit for resource usages in the pod

I Resource Limits

Default



pod-definition.yaml

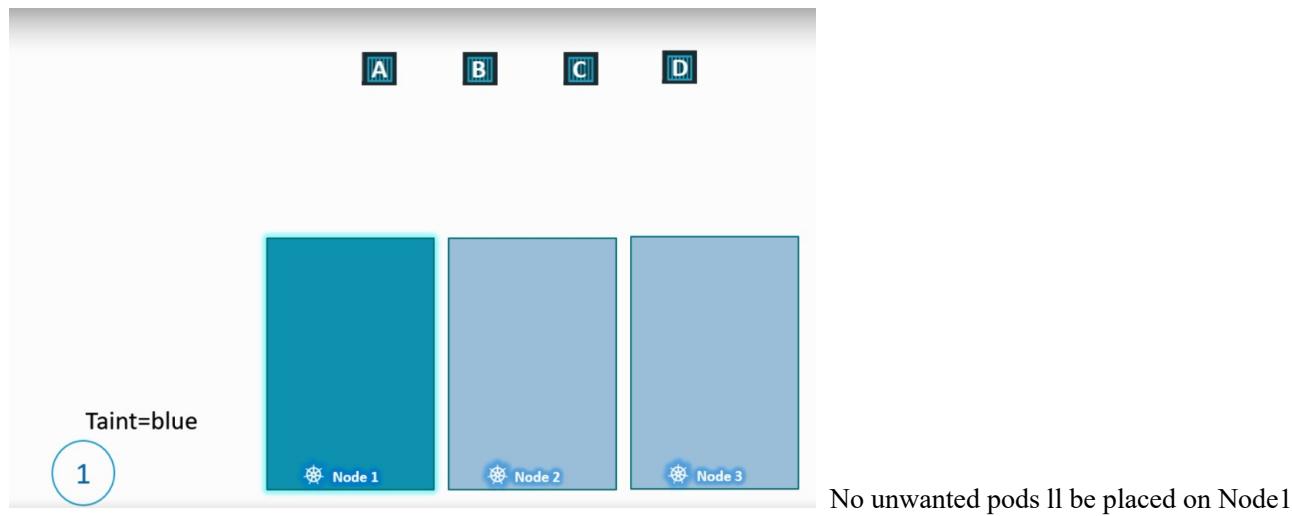
```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
    - containerPort: 8080
    resources:
      requests:
        memory: "1Gi"
        cpu: 1
      limits:
        memory: "2Gi"
        cpu: 2
```

Container can use the more memory resources than its limit, But not CpU
Taints and Tolerants

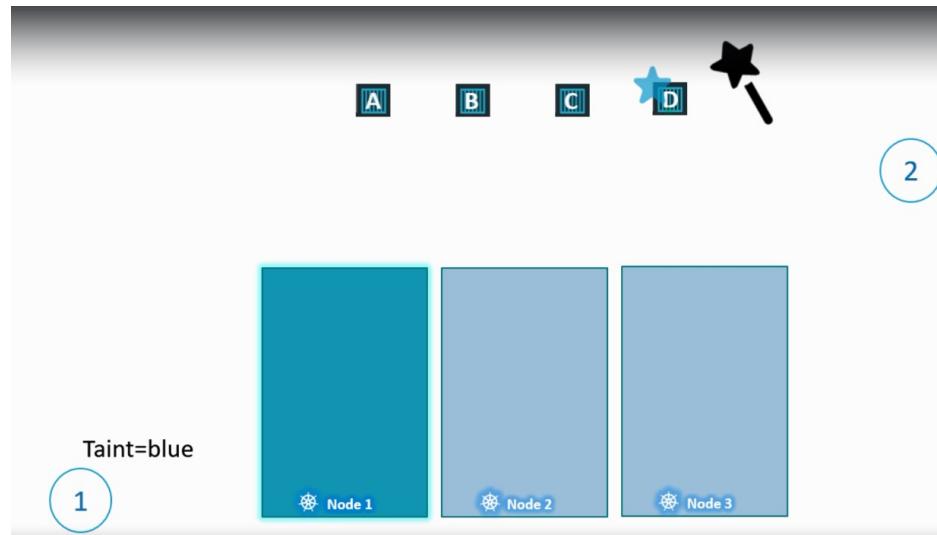


Imp 1. Taints on Node 2. Tolerant level

For some specific use cases , we nee dto deploy the pod on specific Nodes.



Taints on set on Nodes , Tolerance set on Pods



Add Tolerant to pod-D , so that it can deploy in Node1

⌚ Taints - Node

```
kubectl taint nodes node-name key=value:taint-effect
```

NoSchedule | PreferNoSchedule | NoExecute

What happens to PODs
that do not tolerate this taint?

```
kubectl taint nodes node1 app=blue:NoSchedule
```

★ Tolerations - PODs

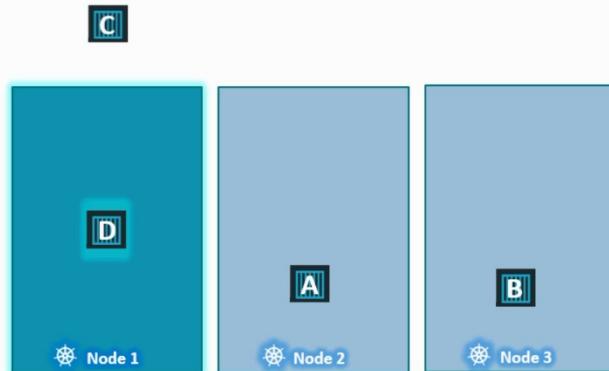
```
pod-definition.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
  tolerations:
    - key: "app"
      operator: "Equal"
      value: "blue"
      effect: "NoSchedule"
```

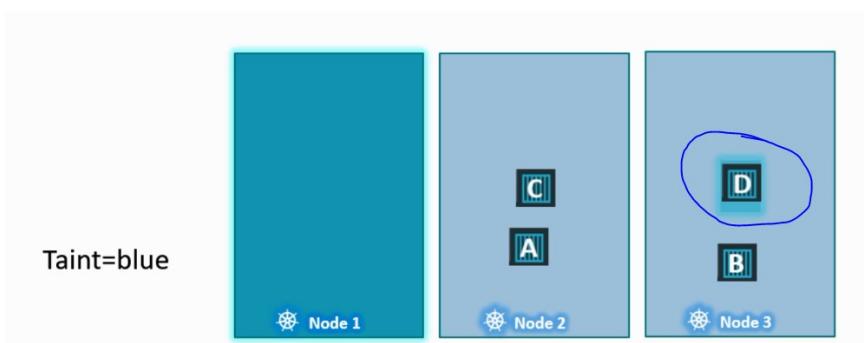
Initially pod C deployed in the Node1, before adding taint to node1.

Pod C from Node1 was evicted because it cannot tolerate the newly added taint

Taint - NoExecute



No Guarantee that pod D always deployed in Node1. It could be deployed in Node3.



BY default Master node is tainted to avoid deploying pod.

```
kubectl describe node kubemaster | grep Taint
```

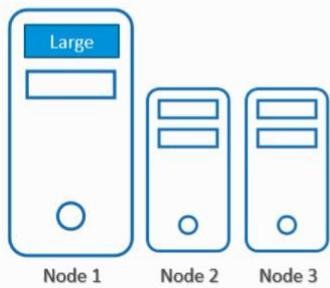
```
Taints:           node-role.kubernetes.io/master:NoSchedule
```

Node Selector

Label Nodes

```
▶ kubectl label nodes <node-name> <label-key>=<label-value>
```

```
▶ kubectl label nodes node-1 size=Large
```

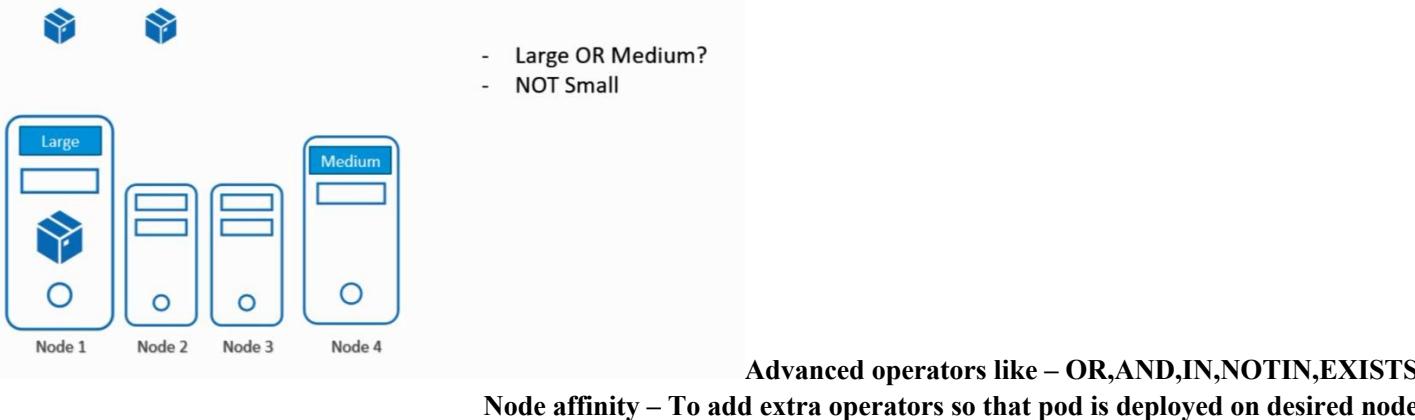


Node Selector



Some cases cannot be achieved by Node selector

Node Selector - Limitations



Node Affinity

```
pod-definition.yml
apiVersion:
kind:
metadata:
  name: myapp-pod
spec:
  containers:
    - name: data-processor
      image: data-processor
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: size
              operator: NotIn
              values:
                - Small
```

Many operators are available , ex – In, NotIn, Exists

Node Affinity Types

Available:

`requiredDuringSchedulingIgnoredDuringExecution`

`preferredDuringSchedulingIgnoredDuringExecution`

Planned:

`requiredDuringSchedulingRequiredDuringExecution`

Node Affinity Types

Available:

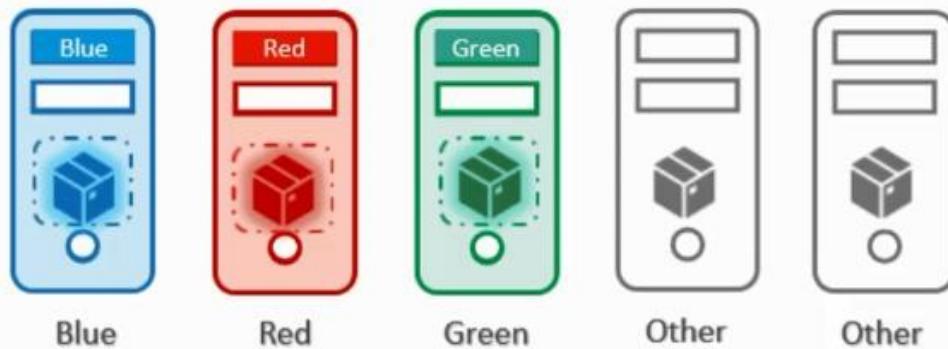
`requiredDuringSchedulingIgnoredDuringExecution`

`preferredDuringSchedulingIgnoredDuringExecution`

	DuringScheduling	DuringExecution
Type 1	Required	Ignored
Type 2	Preferred	Ignored

A pod is running in a Node, in future the label of the Node is changed, in that case pod is continued to be run on the same Node.

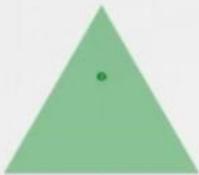
Taints/Tolerations and Node Affinity



Observability- 18%

Have to search liveness probe in search get the template

I Readiness Probe



✓ Ready

HTTP Test - /api/ready

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp
  labels:
    name: simple-webapp
spec:
  containers:
  - name: simple-webapp
    image: simple-webapp
    ports:
    - containerPort: 8080
  readinessProbe:
    httpGet:
      path: /api/ready
      port: 8080
```

| Readiness Probe

```
readinessProbe:  
  httpGet:  
    path: /api/ready  
    port: 8080  
  
  initialDelaySeconds: 10  
  periodSeconds: 5  
  
  failureThreshold: 8
```

```
readinessProbe:  
  tcpSocket:  
    port: 3306
```

```
readinessProbe:  
  exec:  
    command:  
      - cat  
      - /app/is_ready
```

HTTP Test - /api/ready

TCP Test - 3306

Exec Command

Default threshold is 3 attempts, after that it declare it as fail. But we can increase threshold by failureThreshold option

POD Conditions



Liveness Probes



HTTP Test - /api/healthy



TCP Test - 3306



Exec Command

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp
  labels:
    name: simple-webapp
spec:
  containers:
  - name: simple-webapp
    image: simple-webapp
    ports:
    - containerPort: 8080
    livenessProbe:
      httpGet:
        path: /api/healthy
        port: 8080
```

Similar to readinessProbe

I Logs - Docker

```
▶ docker run -d kodekloud/event-simulator  
  
▶ docker logs -f ecf  
  
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in  
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out  
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2  
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3  
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1  
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out  
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in  
2018-10-06 15:57:22,957 - root - INFO - USER3 is viewing page2  
2018-10-06 15:57:23,959 - root - INFO - USER1 logged out  
2018-10-06 15:57:24,959 - root - INFO - USER2 is viewing page2  
2018-10-06 15:57:25,961 - root - INFO - USER1 logged in  
2018-10-06 15:57:26,965 - root - INFO - USER4 is viewing page3  
2018-10-06 15:57:27,965 - root - INFO - USER4 is viewing page3  
2018-10-06 15:57:28,967 - root - INFO - USER2 is viewing page1  
2018-10-06 15:57:29,967 - root - INFO - USER3 logged out
```

I Logs - Kubernetes

```
▶ kubectl create -f event-simulator.yaml  
  
▶ kubectl logs -f event-simulator-pod  
  
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in  
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out  
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2  
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3  
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1  
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out  
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in
```

```
event-simulator.yaml  
  
apiVersion: v1  
kind: Pod  
metadata:  
  name: event-simulator-pod  
spec:  
  containers:  
    - name: event-simulator  
      image: kodekloud/event-simulator
```

Multi container Pod

```
kubectl logs -f event-simulator-pod event-simulator  
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in  
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out  
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2  
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3  
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1  
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out
```

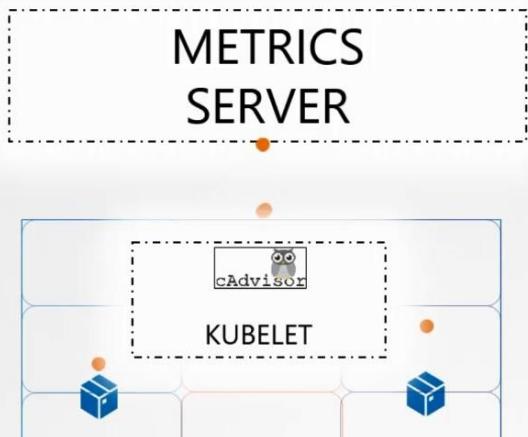
event-simulator.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: event-simulator-pod  
spec:  
  containers:  
    - name: event-simulator  
      image: kodekloud/event-simulator  
    - name: image-processor  
      image: some-image-processor
```

Monitoring: Node level, Pod level – collect metrics of Cpu,Ram, memory usages and give proper analytics

Many 3rd party open source solutions are available. Now we look only Metrics Server

Metrics-Server: In memory only



After installing Metrics server in k8s cluster run the below command to see the metrics

```
kubectl top node
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
kubemaster	166m	8%	1337Mi	70%
kubenode1	36m	1%	1046Mi	55%
kubenode2	39m	1%	1048Mi	55%

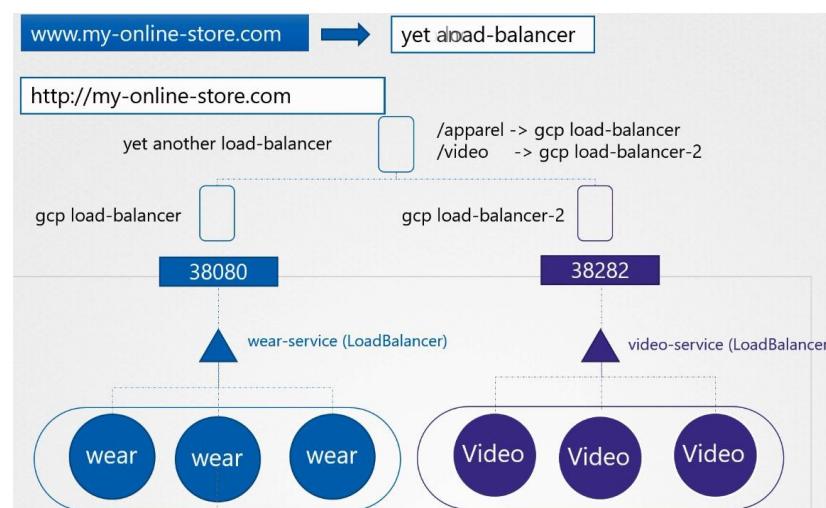
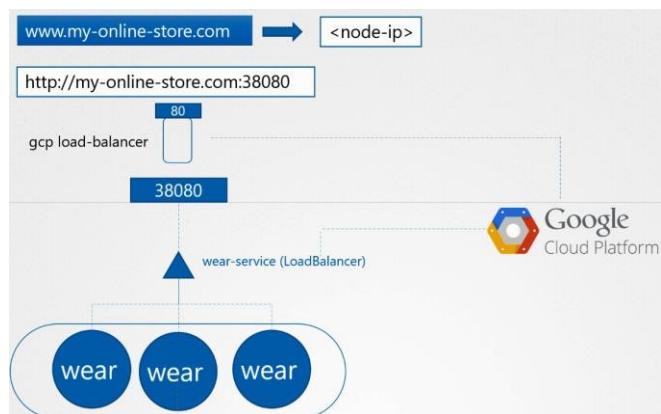
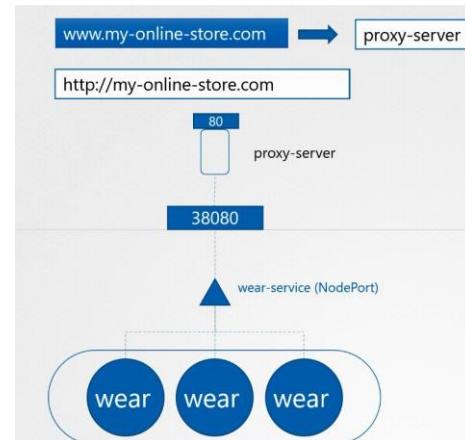
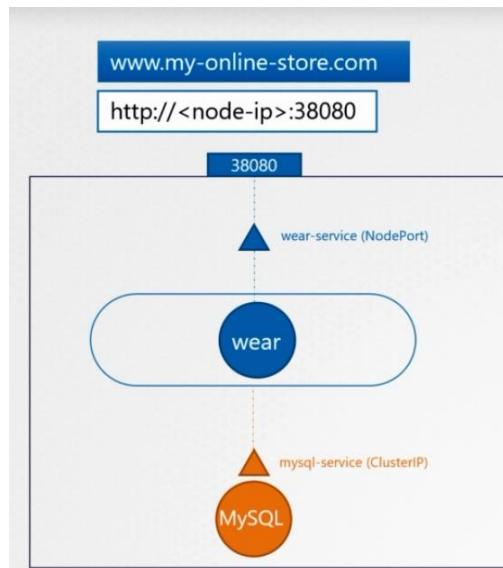
Network and Services – 13%

```
service-definition.yml
```

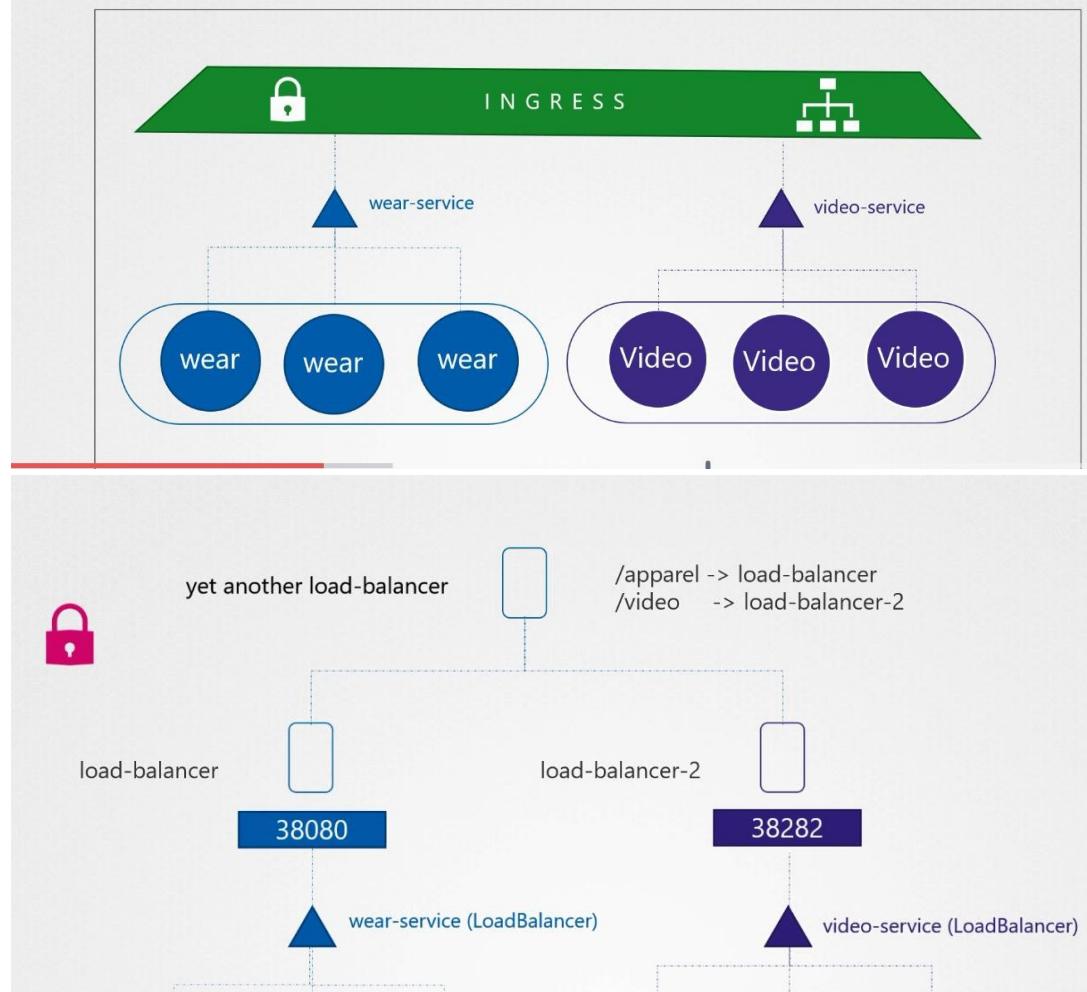
```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
  selector:
    app: myapp
    type: back-end
```



Ingress



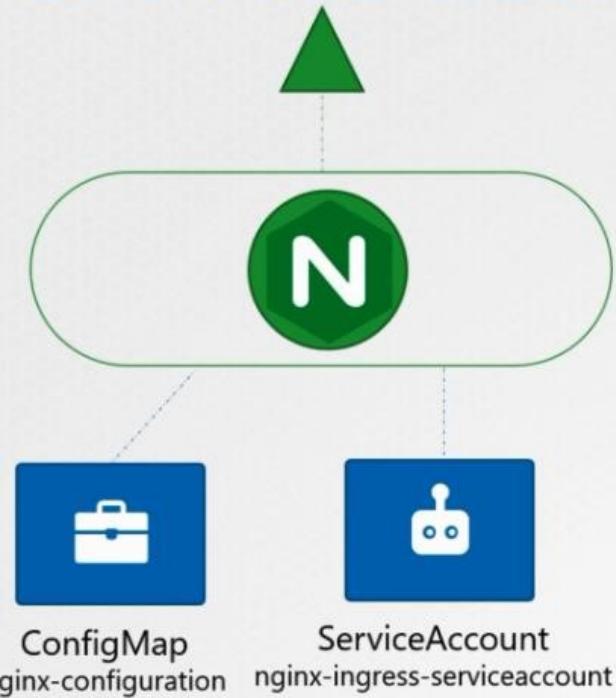
Ingress



INGRESS CONTROLLER



INGRESS CONTROLLER



Deployment

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      name: nginx-ingress
  template:
    metadata:
      labels:
        name: nginx-ingress
    spec:
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
      ports:
        - name: POD_NAME
          containerPort: 80
          fieldRef:
            fieldPath: metadata.name
        - name: POD_NAMESPACE
          containerPort: 443
          fieldRef:
            fieldPath: metadata.namespace
      ports:
        - name: http
          containerPort: 80
        - name: https
          containerPort: 443
```

Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: http
    - port: 443
      targetPort: 443
      protocol: TCP
      name: https
  selector:
    name: nginx-ingress
```

ConfigMap

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
```

Auth

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
```

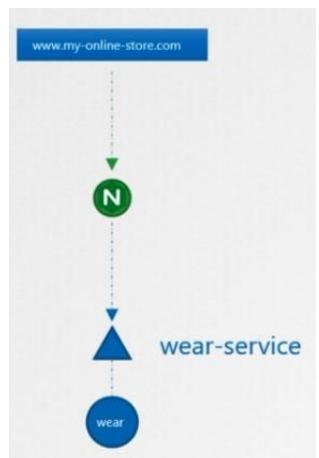
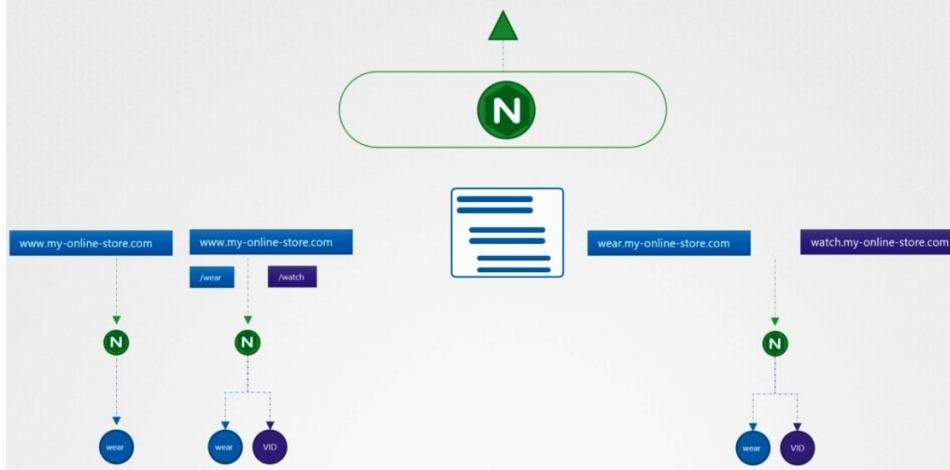
Roles

ClusterRoles

RoleBindings

11 people bookmarked this moment.

INGRESS RESOURCE



Ingress-wear.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-wear
spec:
  backend:
    serviceName: wear-service
    servicePort: 80
```

INGRESS RESOURCE - RULES

www.my-online-store.com	www.wear.my-online-store.com	www.watch.my-online-store.com	Everything Else
http://www.my-online-store.com/wear	http://www.wear.my-online-store.com/	http://www.watch.my-online-store.com/	http://www.listen.my-online-store.com/
http://www.my-online-store.com/watch	http://www.wear.my-online-store.com/returns	http://www.watch.my-online-store.com/movies	http://www.eat.my-online-store.com/
http://www.my-online-store.com/listen	http://www.wear.my-online-store.com/support	http://www.watch.my-online-store.com/tv	http://www.drink.my-online-store.com/tv

Rule 1

- Path /wear
- Path /watch
- Path /

Rule 2

- Path /
- Path /returns
- Path /support

Rule 3

- Path /
- Path /movies
- Path /tv

Rule 4

- Path /

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-wear-watch
spec:
  rules:
  - http:
    paths:
      - path: /wear
        backend:
          serviceName: wear-service
          servicePort: 80
      - path: /watch
        backend:
          serviceName: watch-service
          servicePort: 80
```

INGRESS RESOURCE

2 Rules



Ingress-wear-watch.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-wear-watch
spec:
  rules:
    - host: wear.my-online-store.com
      http:
        paths:
          - backend:
              serviceName: wear-service
              servicePort: 80
    - host: watch.my-online-store.com
      http:
        paths:
          - backend:
              serviceName: watch-service
              servicePort: 80
```

INGRESS RESOURCE

Ingress-wear-watch.yaml

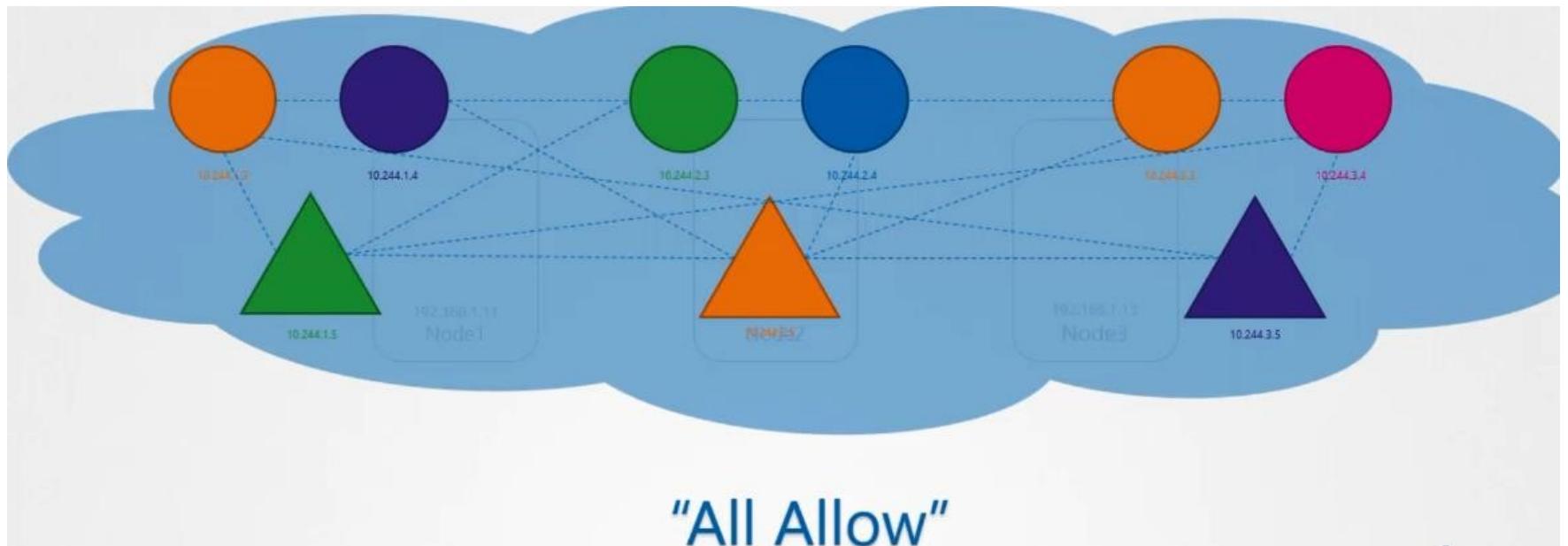
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-wear-watch
spec:
  rules:
  - http:
    paths:
    - path: /wear
      backend:
        serviceName: wear-service
        servicePort: 80
    - path: /watch
      backend:
        serviceName: watch-service
        servicePort: 80
```

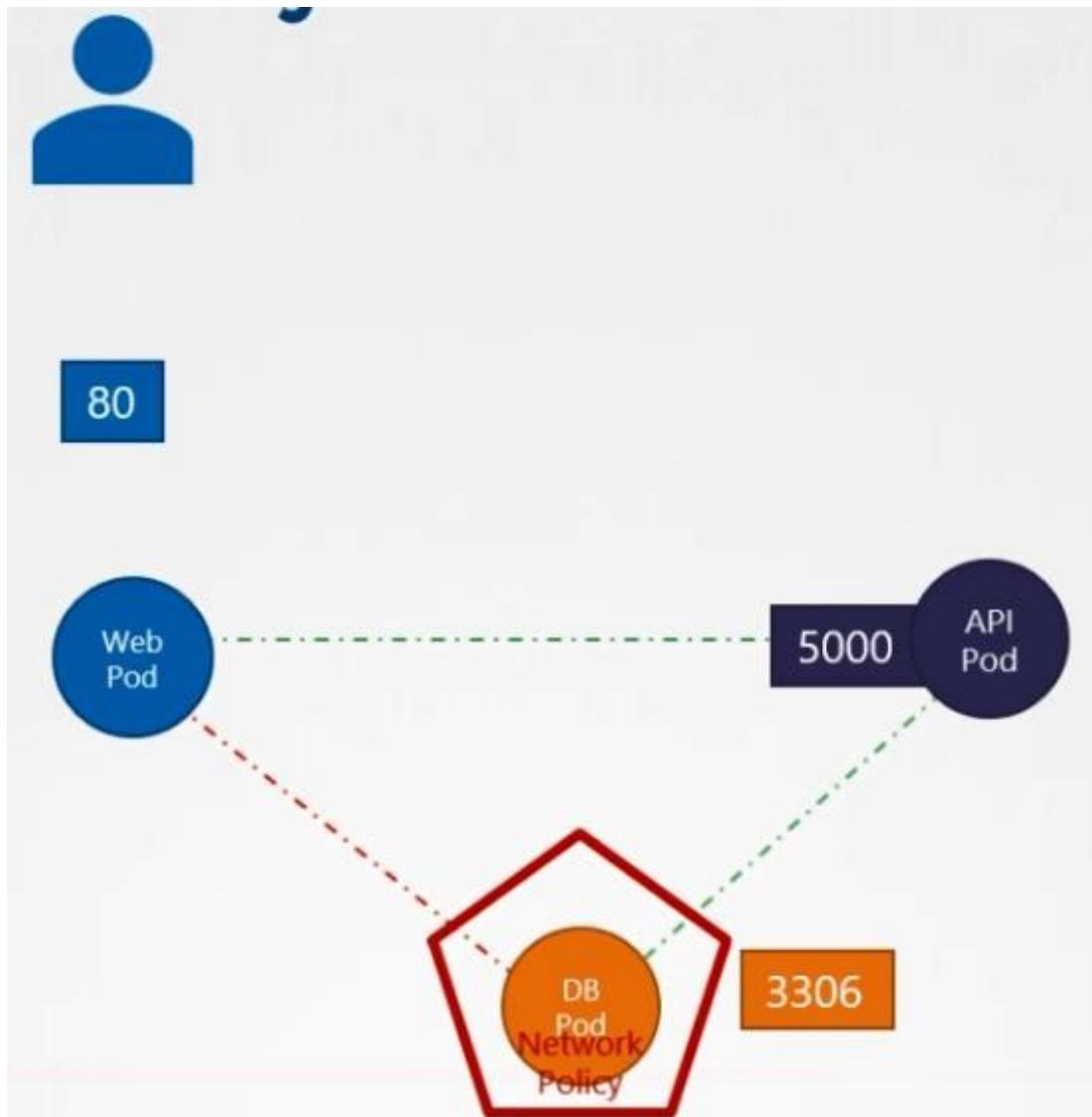
Ingress-wear-watch.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-wear-watch
spec:
  rules:
  - host: wear.my-online-store.com
    http:
      paths:
      - backend:
          serviceName: wear-service
          servicePort: 80
  - host: watch.my-online-store.com
    http:
      paths:
      - backend:
          serviceName: watch-service
          servicePort: 80
```

By Default each pod can communicate with all pod. But some cases it should not be , that's where we specify network policy where we set

rules which pod can communicate with which pod





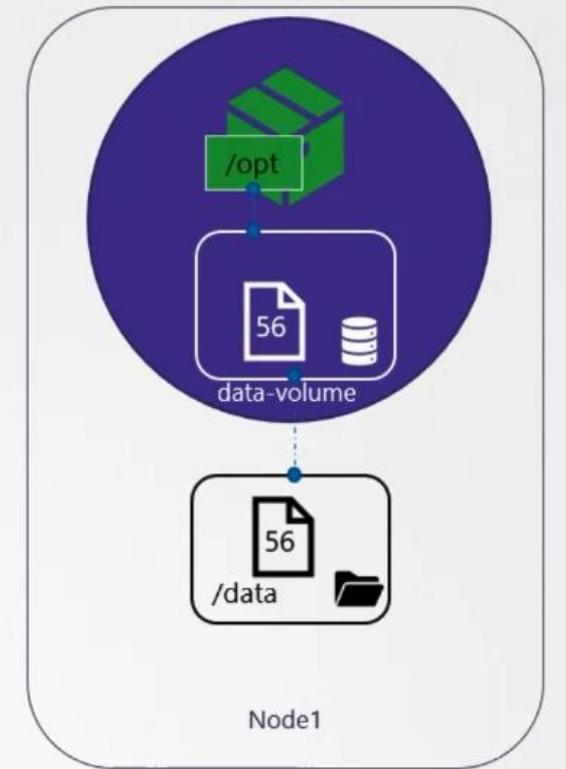
Network Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: api-pod
  ports:
  - protocol: TCP
    port: 3306
```

State Persistence – 8%

Volumes & Mounts

```
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
    - image: alpine
      name: alpine
      command: ["/bin/sh", "-c"]
      args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
  volumeMounts:
    - mountPath: /opt
      name: data-volume
  volumes:
    - name: data-volume
      hostPath:
        path: /data
        type: Directory
```



```
volumes:
- name: data-volume
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```



Persistent Volume

pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-voll
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  hostPath:
    path: /tmp/data
```

pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-voll
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

Persistent Volume Claim

pvc-definition.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-vol1
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

kubectl get persistentvolumeclaim

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
myclaim	Bound	pv-vol1	1Gi	RWO		43m

```
▶ kubectl delete persistentvolumeclaim myclaim  
persistentvolumeclaim "myclaim" deleted
```

persistentVolumeReclaimPolicy: Retain

persistentVolumeReclaimPolicy: Recycle

persistentVolumeReclaimPolicy: Delete

Use Shortcuts/Aliases

po for PODs

rs for ReplicaSets

deploy for Deployments

svc for Services

ns for Namespaces

netpol for Network policies

pv for Persistent Volumes

pvc for PersistentVolumeClaims

sa for service accounts