

# Kubernetes Important Commands

---

- `kubectl get nodes / pods / services / deployments / replicaset / namespaces`
- `kubectl get all -o wide / yaml / json`
- `kubectl get events`
- `kubectl get endpoints`
- `kubectl get pods --namespace= dev`
- `Kubectl get pods -n namespace`
- `kubectl get pods --namespace= dev --show-labels`
- `kubectl config set-context --current --namespace= namespace-name`
- `kubectl config set-context context-name --cluster= new-cluster-name --user=new- user-name -- namespace= new-namespace`
- `kubectl describe pod / service / deployment / replicaset`
- `kubectl run name --image= image --command -- cmd args1 args2`
- `kubectl run name --image= image --dry-run=client -o yaml > name .yaml`
- `kubectl run name --image= image -- args1 ... argsN # default commands and custom arguments`
- `kubectl replace --force -f name .yaml # replace`
- `kubectl edit pod / service / deployment / replicaset resource-name # edit`
- `kubectl set image deployment deployment-name name = image-name : version`
- `kubectl delete pod name`
- `kubectl delete pod name --grace-period= 0 --force # force delete`
- `kubectl delete pod name --grace-period= 0 --force --namespace= namespace`
- `kubectl delete pod name --grace-period= 0 --force --namespace= namespace`
- `kubectl logs -f pod / service / deployment / replicaset # get logs`
- `kubectl logs -f pod -C container_name`
- `kubectl logs --tail=20 pod-name`
- `kubectl logs --since=1h pod-name`
- `Kubectl exec pod_name -it -C container_name -- /bin/bash # execute command into the pod or particular container`
- `kubectl create pod / service / deployment / replicaset / namespace -f name .yaml`
- `kubectl create -f name .yaml`

- `kubectl create -f name.yaml --record` # record the history
- `kubectl apply -f config1.yaml -f config2.yaml` # Applies multiple configuration files.
- `kubectl apply -f https://url.com/config.yaml` # Applies a configuration from a URL.
- `kubectl scale deployment deployment-name --replicas= 5 -f deploy.yml --record` # Scales a deployment to 5 replicas and records the change in the revision history.
- `kubectl scale replicaset rs-name --replicas= 3 -f replica.yml`
- `kubectl rollout history deployment/ deployment-name` # Shows the history of deployments
- `kubectl rollout pause deployment/ deployment-name` # Pauses the rollout of a deployment
- `kubectl rollout resume deployment/ deployment-name` # Resumes a paused deployment rollout.
- `Kubectl rollout status deployment deployment-name`
- `kubectl expose deployment deployment-name --port=80 --target-port=8080` # Exposes a deployment as a new service on port 80, targeting the container port 8080.
- `kubectl label pods pod-name app=myapp`
- `kubectl label nodes node-name disk=ssd`
- `kubectl patch deployment deployment-name -p '{"spec":{"replicas":5}}'` # Updates the number of replicas for a deployment to 5.
- `kubectl patch service service-name -p '{"spec":{"type":"LoadBalancer"}}'` # Updates the service type to LoadBalancer.
- `kubectl autoscale deployment deployment-name --min=2 --max=10 --cpu-percent=80` # Configures auto-scaling for a deployment based on CPU utilization.
- `kubectl autoscale replicaset rs-name --min=3 --max=6 --cpu-percent=70` # Configures auto-scaling for a ReplicaSet based on CPU utilization.
- `kubectl top nodes` # Shows CPU and memory usage for nodes in the cluster.
- `kubectl top pods --containers` # Shows CPU and memory usage for containers in pods.
- `kubectl drain node-name` # Safely drains a node in preparation for maintenance.
- `kubectl drain node-name --ignore-daemonsets` # Drains a node while ignoring DaemonSet pods.
- `kubectl taint nodes node-name key=value:NoSchedule`
- `kubectl taint nodes node-name key=value:NoSchedule-` # remove taint
- `kubectl taint nodes node-name key=value:NoExecute`
- `kubectl taint nodes node-name key=value:NoExecute-` # remove taint
- `kubectl cordon node-name` # Marks a node as unschedulable, preventing new pods from being scheduled on it.
- `kubectl uncordon node-name` # Marks a node as schedulable, allowing new pods to be scheduled on it.

- `kubectl cp file.txt pod-name : /path/to/dir # Copies a file from the local machine to a pod.`
- `kubectl cp pod-name : /path/to/file.log file.log # Copies a file from a pod to the local machine.`
- `kubectl api-resources | tail +2 | awk ' { print $1 }'; do kubectl explain $kind; done | grep -e "KIND:" -e "VERSION:"`

## Labels and Selectors:

---

- Labels are the way by which you can filter and sort the number pods in a cluster
- selector tells the replicaset/controller/deployment which pods to watch/belongs to
- The label selector is the core grouping primitive in Kubernetes.
- Labels are useful for filtering the environment specific pods
- You can assign multiple labels to a pod, node, cluster objects.
- By giving multiple labels you can filter out exact pods that you required.
- Labels are key=value pair without any predefined meaning
- They are similar to tags in AWS and Git
- You are free to choose labels of any name.
- You can provide label through both imperative and declarative methods.
- Define it under the **metadata**
- **Annotations** are extra details for the pod

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myDeployment

  # This are the Deployment labels
  labels:
    app: myApp
    env: dev
spec:
  replicas: 3

  # the Selector selects the pod with specific labels
  selector:
```

```
    matchLabels:
      tier: front-end

# template defines the PodSpec for the new pods being created and what Labels it contains
template:
  metadata:
    name: testpod
  labels:
    tier: front-end
  spec:
    containers:
    - name: c00
      image: nginx:latest
```

## Imperative Commands to find pods with selected Labels

---

```
kubectl get pods --selector key=value --no-headers | wc -l
```

```
kubectl get pods --selector env=prod,env=prod,tier=frontend
```

## Taints and Tolerance:

---

- Taints are placed on Nodes
- Tolerations is placed on Pods
- Tolerations allow the scheduler to schedule pods with matching taints.
- if Taint are placed on the specific node it will not allow a pod without tolerance to be scheduled on that Node
- if we place the tolerance on the specific pod then only it will be able to be scheduled on the node else not
- Taints are the opposite -- they allow a node to repel a set of pods.
- Taints and tolerations work together to ensure that pods are not scheduled onto **inappropriate** nodes.
- One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints
- Taints and Tolerations does not tell pod to go on specific node
- *Taints and Tolerations only tells accept pods on the node with certain tolerations.*
- if you want to restrict a certain pod on a single node then you have to use the node **Affinity**.

- Taint is already placed on masterNode which tells the scheduler not to schedule pods on the masterNode to view which taint effect is applied on the master node use below command.
  - `kubectl describe node kubemaster | grep Taint`

## Taint-Effects:

---

- **NoExecute**
  - Pods that do not tolerate the taint are evicted immediately
  - Pods that tolerate the taint without specifying `tolerationSeconds` in their toleration specification remain bound forever
  - Pods that tolerate the taint with a specified `tolerationSeconds` remain bound for the specified amount of time. After that time elapses, the node lifecycle controller evicts the Pods from the node
  - Pods currently running on the node are **evicted** if it does not match the tolerance.
  - `NoExecute` effect can specify an optional `tolerationSeconds` field that dictates how long the pod will stay bound to the node after the taint is added
- **NoSchedule**
  - No new Pods will be scheduled on the tainted node unless they have a matching toleration. Pods currently running on the node are not evicted.
- **PreferNoSchedule**
  - `PreferNoSchedule` is a "preference" or "soft" version of `NoSchedule`.
  - The control plane will try to avoid placing a Pod that does not tolerate the taint on the node, *but it is not guaranteed*.

## Commands: (Applying Tolerance to Node:)

To Apply the Taint on the Node

```
kubectl taint nodes node1 key1=value1:NoSchedule
```

To remove the taint from the node:

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```

```
kubectl taint nodes node1 key1=value1:NoSchedule
kubectl taint nodes node1 key1=value1:NoExecute
kubectl taint nodes node1 key2=value2:NoSchedule
```

## Applying Tolerance to Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent

# add tolerance to Pod
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
```

## Node Selectors:

---

- Node selectors helps in scheduling the pod on specific node based on labels assigned to node and the pods
- suppose we have Three nodes namely Large, Small with labels as size = large/small.
- We can use this label information to schedule our pods accordingly.

Example :

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
```

```
# Apply node selector to run pod on node with size=large label on it
nodeSelector:
  size: large
```

## Commands:

---

Add a label to a particular node

```
kubect1 label nodes <node_name> key=value
```

## Limitations:

NodeSelector will not be able to help in complex senerios such as

- size=large or size=small
- size !=medium (not equal)

## Node Affinity:

---

- with the help of Node Affinity , you can control the scheduling of pods onto nodes more granularly using various criteria like
- It is an extension of node selectors which allows you to specify more flexible rules about where to place your pods by specifying certain requirements.

Example: This manifest describes a Pod that has a `requiredDuringSchedulingIgnoredDuringExecution` node affinity, `size=large` or `size=small` . This means that the pod will get scheduled only on a node that has a `size=large` or `size=small`

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: size
                operator: In
```

```
      values: # The Pod will scheduled on the nodes that have at least one of these values
      - large
      - small
containers:
- name: nginx
  image: nginx
  imagePullPolicy: IfNotPresent
```

## Node Affinity types:

### 1. **required DuringScheduling Ignored DuringExecution**

- Pods with this affinity rule will only be scheduled on nodes that meet the specified requirements. However, if a node loses the label that made it eligible after the pod is scheduled, the pod will still continue to run on that node.

### 2. **Preferred DuringScheduling Ignored DuringExecution**

- Pods with this affinity rule will prefer to be scheduled on nodes that meet the specified requirements, but they are not strictly required to. If no nodes match the preferred requirements, the pod can still be scheduled on other nodes.

Syntax	DuringScheduling	DuringExecution
Type1	Required	Ignored
Type2	preferred	Ignored
Type3	Required	Required

## Taints and Tolerance & Node Affinity:

- You can use both taints and tolerations & Node affinity to granular control over the Pod scheduling on the specific nodes.
- **what if we applied taint to specific node and Node Affinity to a pod to be schedule on the specific node only without tolerance set what will happen?**
  - When the scheduler attempts to place the pod onto a node, it evaluates the Node Affinity rules against the labels of each node in the cluster.
  - If there is a node that meets the Node Affinity requirements specified in the pod's configuration and doesn't have a taint that repels the pod (i.e., the node matches the Node Affinity and doesn't have any taints), the pod will be scheduled on that node.
  - However, if the node with the matching Node Affinity also has the taint that the pod lacks



tolerations for, the pod won't be scheduled on that node due to the taint-repelling behavior. In this case, the pod won't be scheduled anywhere unless there is another node in the cluster that meets the Node Affinity requirements and doesn't have the taint applied.

## Resource Requests and Limits:

---

- **Requests:** Requests specify the minimum amount of resources (CPU and memory) that a pod needs to run.
  - When a pod is scheduled onto a node, Kubernetes ensures that the node has enough available resources to satisfy the pod's resource requests.
  - Requests are used by the **scheduler** to determine the best node for placing a pod.
- **Limits:** Limits, on the other hand, specify the maximum amount of resources that a pod can consume.
  - Kubernetes enforces these limits to prevent pods from using more resources than allowed, which helps in maintaining stability and preventing resource contention.
  - If a pod exceeds its specified limits, Kubernetes may take actions such as throttling or terminating the pod.
  - Pod cannot exceed the CPU limit but can exceed the Memory Limit in that case K8s will try to Terminate or kill the Pod when it finds that the pod is using more resources than allowed  
OOM (out of memory (OOM) error) `OOMKilled` means the pod has been killed by Out of Memory.
  - if you donot specify the Request the Limit will be `Request = Limit`

## Use Cases:

---

### 1. No Request and No Limit

- This scenario is suitable for non-critical applications or experiments where resource requirements are flexible, and the pod can utilize whatever resources are available on the node without any constraints. However, it can lead to potential resource contention issues if other pods on the node require resources.

### 2. No Request and Limit

- This scenario is useful when you want to constrain the resource usage of a pod to prevent it from

using excessive resources and impacting other pods on the node. However, without a request, Kubernetes scheduler may place the pod on a node without considering its resource needs, potentially leading to inefficient resource allocation.

### 3. Request and Limit

- This is the most common and recommended configuration. By specifying both requests and limits, Kubernetes can ensure that the pod gets scheduled on a node with sufficient resources and enforce resource constraints to maintain stability and fairness within the cluster.

### 4. Request and No Limits

- This scenario is suitable when you want to ensure that the pod gets scheduled on a node with sufficient resources based on its requirements, but you're not concerned about limiting its resource usage. It allows the pod to scale up its resource consumption based on demand without strict constraints, which can be useful for certain workloads with variable resource requirements.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "1"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

#### Note:

Kubernetes doesn't allow you to specify CPU resources with a precision finer than `1m` or `0.001` CPU. To avoid accidentally using an invalid CPU quantity, it's useful to specify CPU units using the milliCPU form instead of the decimal form when using less than `1` CPU unit. For example, you have a Pod that uses `5m` or `0.005` CPU and would like to decrease its CPU resources. By using the decimal form, it's harder to spot that `0.0005` CPU is an invalid value, while by using the milliCPU form, it's easier to spot that `0.5m` is an invalid value. Limits and requests for memory

are measured in bytes. You can express memory as a plain integer or as a fixed-point number using one of these quantity suffixes: E, P, T, G, M, k. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. For example, the following represent roughly the same value: 128974848, 129e6, 129M, 128974848000m, 123Mi. Pay attention to the case of the suffixes. If you request 400m of memory, this is a request for 0.4 bytes. Someone who types that probably meant to ask for 400 mebibytes ( 400Mi ) or 400 megabytes ( 400M ).

## DaemonSet:

- DaemonSet are similar to ReplicaSets but inly difference is that DaemonSet ensure that every node has at least one instance of a pod running.
- when the node gets created DaemonSet automatically creates the pod on the Node.
- Pods created by the DaemonSet are ignored by the Kube-scheduler.

### Use Cases:

- running a cluster storage daemon on every node
- running a *logs collection* daemon on every node
- running a *node monitoring* daemon on every node
- one DaemonSet, covering all nodes, would be used for each type of daemon. alt text

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
        # these tolerations are to have the daemonset runnable on control plane nodes
```

```
# remove them if your control plane nodes should not run pods
- key: node-role.kubernetes.io/control-plane
  operator: Exists
  effect: NoSchedule
- key: node-role.kubernetes.io/master
  operator: Exists
  effect: NoSchedule
containers:
- name: fluentd-elasticsearch
  image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
  resources:
    limits:
      memory: 200Mi
    requests:
      cpu: 100m
      memory: 200Mi
  volumeMounts:
  - name: varlog
    mountPath: /var/log
# it may be desirable to set a high priority class to ensure that a DaemonSet Pod
# preempts running Pods
# priorityClassName: important
terminationGracePeriodSeconds: 30
volumes:
- name: varlog
  hostPath:
    path: /var/log
```

```
kubectl apply -f DaemonSet.yml
```

## Running Pods on select Nodes

If you specify a `.spec.template.spec.nodeSelector`, then the DaemonSet controller will create Pods on nodes which match that **node selector**. Likewise if you specify a `.spec.template.spec.affinity`, then DaemonSet controller will create Pods on nodes which match that **node affinity**. If you do not specify either, then the DaemonSet controller will create Pods on all nodes.

## Static Pods:

- Static Pods are managed directly by the **kubelet daemon** on a specific node, without the API server observing them.

- kube-Scheduler ignore the Static pods
- Static pod are used to depoly the Control plane pods
- static pods are only created by kubelet.
- Static Pods are always bound to one Kubelet on a specific node.
- to Identify the Static pods, they always have a nodeName at the end. ex- **etcd- controlplane** , **kube-apiserver- controlplane** , **kube-controller-manager- controlplane** , **kube-scheduler- controlplane** or you can refer to the OwnerReference in the Pod defination file.
- Kubelet does not manage the Deployment and Replicasset it only manages the Pod
- Static pods are visible to the Kube-Apiserver, but does not control from there
- Kubelet also runs on the master node in Kubernetes to provision the master components as static pods.

There are two way to craete a Static pod

1. Filesystem-hosted static Pod manifest
2. Web-hosted static pod manifest

- Manifests are standard Pod definitions in JSON or YAML format in a specific directory. Use the `staticPodPath: <the directory>` field in the [kubelet configuration file](#), which periodically scans the directory and creates/deletes static Pods as YAML/JSON files **appear/disappear there**.
- to modify the kubelet use the [KubeletConfiguration](#) YAML file which is the recommended way.

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
address: "192.168.0.8"
port: 20250
serializeImagePulls: false
evictionHard:
  memory.available: "100Mi"
  nodefs.available: "10%"
  nodefs.inodesFree: "5%"
  imagefs.available: "15%"
  staticPodPath: "/etc/kubernetes/manifests"
```

to apply this file

```
kubectl apply -f kubelet-config.yaml
```

- `staticPodPath: "/etc/kubernetes/manifests"` is were the static-pod manifest are stored this path is not same always *refer the kubelet config file for the mention path*

- `cat /var/lib/kubelet/config.yaml` look for `staticPodPath`

or

- `--pod-manifest-path=/etc/kubernetes/manifests/` (**command line method for creating the static pod**)
  - `systemctl restart kubelet`
- `kubectrl run --restart=Never --image=busybox static-busybox --dry-run=client -o yaml --command -- sleep 1000 > /etc/kubernetes/manifests/static-busybox.yaml`

```
controlplane ~ → ls -lrt /etc/kubernetes/manifests/
total 16
-rw----- 1 root root 2406 Apr  7 16:35 etcd.yaml
-rw----- 1 root root 1463 Apr  7 16:35 kube-scheduler.yaml
-rw----- 1 root root 3393 Apr  7 16:35 kube-controller-manager.yaml
-rw----- 1 root root 3882 Apr  7 16:35 kube-apiserver.yaml
```

- to ssh into any node within k8s cluster you can use the command `ssh <Node-name>/<Node-IP>`

## Multiple Schedulers:

- K8s comes with a single default scheduler but as the k8s is highly extensible, If the default scheduler does not suit your needs you can **implement your own scheduler**.
- you can even run multiple schedulers simultaneously alongside the default scheduler.

### STEP 1:

- Package your scheduler binary into a container image.
- Clone the [Kubernetes](#) source code from GitHub and build the source.
- `Docker Build -t` and `Docker Push` to push the Image to Docker Hub (or another registry). refer k8s documentation

### Docker File

```
FROM busybox
ADD ./_output/local/bin/linux/amd64/kube-scheduler /usr/local/bin/kube-scheduler
```

## STEP 2: Define a Kubernetes Deployment for the scheduler

- you have your scheduler in a container image
- create a pod/Deployment configuration for it and run it in your Kubernetes cluster.
- Save it as [my-scheduler.yaml](#)
- in the deployment use this

```
---
# **my-scheduler-config.yaml**
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: my-scheduler
leaderElection:
  leaderElect: false

---
# **my-scheduler-configmap.yaml**
apiVersion: v1
data:
  my-scheduler-config.yaml: |
    apiVersion: kubescheduler.config.k8s.io/v1
    kind: KubeSchedulerConfiguration
    profiles:
      - schedulerName: my-scheduler
    leaderElection:
      leaderElect: false
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: my-scheduler-config
  namespace: kube-system

---
# **my-scheduler.yml**
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-scheduler
```

```
name: my-scheduler
namespace: kube-system
spec:
  serviceAccountName: my-scheduler
  containers:
  - command:
    - /usr/local/bin/kube-scheduler
    - --config=/etc/kubernetes/my-scheduler/my-scheduler-config.yaml
    image: registry.k8s.io/kube-scheduler:v1.29.0
    livenessProbe:
      httpGet:
        path: /healthz
        port: 10259
        scheme: HTTPS
      initialDelaySeconds: 15
    name: kube-second-scheduler
    readinessProbe:
      httpGet:
        path: /healthz
        port: 10259
        scheme: HTTPS
    resources:
      requests:
        cpu: '0.1'
    securityContext:
      privileged: false
    volumeMounts:
    - name: config-volume
      mountPath: /etc/kubernetes/my-scheduler
  hostNetwork: false
  hostPID: false
  volumes:
  - name: config-volume
    configMap:
      name: my-scheduler-config
```

**STEP 3:** How to use the custom scheduler into pod definitions.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
```



```
- name: nginx
  image: nginx
```

```
# This is how you can add the custom scheduler to pod.
schedulerName: my-scheduler
```

### Example:

```
controlplane ~ → kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-flannel	kube-flannel-ds-nssdp	1/1	Running	0	10m
kube-system	coredns-69f9c977-25wxh	1/1	Running	0	10m
kube-system	coredns-69f9c977-9v84m	1/1	Running	0	10m
kube-system	etcd-controlplane	1/1	Running	0	11m
kube-system	kube-apiserver-controlplane	1/1	Running	0	11m
kube-system	kube-controller-manager-controlplane	1/1	Running	0	10m
kube-system	kube-proxy-zhj22	1/1	Running	0	10m
kube-system	kube-scheduler-controlplane	1/1	Running	0	10m
kube-system	**my-scheduler**	1/1	Running	0	2m22s

```
kubectl get events -o wide
```

```
kubectl logs <scheduler_name> -n <name_space>
```

## Scheduler Profiles:

- Suppose if you have multiple schedulers you can combine all those schedulers to use same binary rather than creating multiple binaries, this can be achieved with the help of Scheduler Profile in latest k8s versions.
- this is an efficient way to use and configure multiple schedulers in k8s cluster.
- How a Pod gets scheduled on the nodes
  - Scheduling Queue
  - Filtering
  - Scoring
  - Binding

alt text

## Scheduling Queue:

---

### PriorityClass

- you can set the priority of pod to be scheduled

```
---
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."

---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
  resources:
    requests:
      memory: 1Gi
      cpu: 10
```

## Filtering:

---

- Filter which nodes are not suitable for running the pod, e.g., lack of resource or taints and tolerations

## Scoring:

---

- after the filtering phase Nodes which are eligible will go through the scoring phase . The node with the highest score is selected.
- Score means Suitable to run the Pod and have more available free resource after scheduling.
- The scheduler assigns a score to each Node that survived filtering, basing this score on the active

scoring rules.

## Binding:

---

- The nodes which have a good score are selected for binding.

## Sceduler Plugins:

- **Scheduling Queue**
  - PrioritySort
- **Filtering**
  - NodeResourcesFit
  - NodeName
  - NodeUnschedule
- **Scoring**
  - NodeResourcesFit
  - ImageLocality
- **Binding**
  - DefaultBinder

These Plugins are plugged to **Extention points**

1. queueSort
2. filter
3. score
4. bind

## Logging and Monitoring the Kubernetes Components:

---

### Monitoring The k8s Cluster

---


- `Metric Server` is available on each node to gather logs.
- Metric server can be enabled using `minikube addons enable metrics-server` on minikube only
- for Others to deploy **metric server**

- `git clone https://github.com/kubernetes-incubator/metrics-server.git`
- `kubectl create -f deploy/1.8+/`
- `kubectl top node` to view resource consumption of cluster.

controlplane kubernetes-metrics-server on  master → `kubectl top node`

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
controlplane	258m	0%	1084Mi	0%
node01	118m	0%	268Mi	0%

- `kubectl top pod` to view resources consumption of pods.

controlplane kubernetes-metrics-server on  master → `kubectl top pod`

NAME	CPU(cores)	MEMORY(bytes)
elephant	15m	32Mi
lion	1m	18Mi
rabbit	102m	14Mi

## Logging in K8s

- to view logs of a pod
  - `kubectl logs <pod-name>`
- to view logs of a pod in a specific container
  - `kubectl logs <pod-name> -c <container-name>`
- to view logs of a pod in a specific container and follow the logs
- `kubectl describe pod` to find the container name

```
Name:          my-pod
Namespace:     default
Priority:       0
Node:          node1/10.0.0.1
Start Time:    Tue, 01 Mar 2023 10:00:00 +0000
Labels:        app=my-app
Annotations:    <none>
Status:        Running
IP:            10.0.0.2
IPs:
  IP: 10.0.0.2
Containers:
  **my-container:**
    Container ID:  docker://abcdefg1234567890
```

```
Image:          my-image:latest
Image ID:       docker-pullable://my-image@sha256:abcdefg1234567890
Port:          8080/TCP
Host Port:      0/TCP
State:          Running
Started:        Tue, 01 Mar 2023 10:00:01 +0000
```

# Application Lifecycle Management in K8s

---

## Rollout and Versioning

---

- Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates
- A **rolling update** allows a Deployment update to take place with **zero downtime**
- It does this by incrementally replacing the current Pods with new ones.
- Kubernetes waits for those new Pods to start before removing the old Pods.
- In Kubernetes, updates are versioned and any Deployment update can be reverted to a previous (stable) version.
- the Service will load-balance the traffic only to available Pods during the update.
- when updates happens the new set of replicas are created one by one and the old set of replicas are down to 0 one by one in **rolling update**. when doing the undo old ones are up and new ones are down to zero one by one.

```
> kubectl get replicaset
NAME DESIRED CURRENT READY AGE
myapp-deployment-67c749c58c 0 0 0 22m
myapp-deployment-7d57dbdb8d 5 5 5 20m
```

### Rolling updates allow the following actions:

1. Promote an application from one environment to another (via container image updates)
2. Rollback to previous versions
3. Continuous Integration and Continuous Delivery of applications with zero downtime

alt text

to update the image

- `kubectl set image deployment/<deployment-name> <container-name>=<new-image>`

To view the status of application upgrade

- `kubectl rollout status deployment/<deployment-name>`

```
> kubectl rollout status deployment/myapp-deployment
Waiting for rollout to finish: 0 of 10 updated replicas are available...
Waiting for rollout to finish: 1 of 10 updated replicas are available...
Waiting for rollout to finish: 2 of 10 updated replicas are available...
Waiting for rollout to finish: 3 of 10 updated replicas are available...
Waiting for rollout to finish: 4 of 10 updated replicas are available...
Waiting for rollout to finish: 5 of 10 updated replicas are available...
Waiting for rollout to finish: 6 of 10 updated replicas are available...
Waiting for rollout to finish: 7 of 10 updated replicas are available...
Waiting for rollout to finish: 8 of 10 updated replicas are available...
Waiting for rollout to finish: 9 of 10 updated replicas are available...
deployment "myapp-deployment" successfully rolled out
```

To view the history of application upgrade

- `kubectl rollout history deployment/<deployment-name>`

```
> kubectl rollout history deployment/myapp-deployment
deployments "myapp-deployment"
REVISION CHANGE-CAUSE
1          <none>
2          kubectl apply --filename=deployment-definition.yml --record=true
```

To rollback to a previous version

- `kubectl rollout undo deployment/<deployment-name>`

## Deployment Strategy

---

- The Deployment controller supports two different strategies for rolling updates:

### 1. Recreate

- In this strategy, all existing Pods are killed one at a time, and new ones are created to replace them.
- In the strategy user will face the downtime of application during the update.

## 2. Rolling Update

- In this strategy, new Pods are created and the old ones are killed one at a time.
- In this strategy, user will not face the downtime of application during the update.

alt text

alt text alt text

There are two ways to update the deployment

### 1. Manually

- The user can manually update the deployment by changing the image version in the deployment manifest. `kubectl edit deployment <deployment-name>`
- or using commandline `kubectl set image deployment/<deployment-name> <container-name>=<new-image-name>`

### 2. Automatically

- The user can also update the deployment automatically by using the image update policy.

## Application Commands, EntryPoint and Arguments in Pod definition

---

### Docker

Difference Between `ENTRYPOINT` and `CMD` in Docker and use case.

- **`CMD`** is used to run the command when the container is started.
  - The `CMD` instruction specifies the default command or arguments that will be executed by the `ENTRYPOINT` if no other command is provided when running the container.
  - If you use both `ENTRYPOINT` and `CMD` in a Dockerfile, the `CMD` instruction will provide the default arguments for the `ENTRYPOINT`.
- **`ENTRYPOINT`** is used to run the command when the container is started.
  - It is the preferred way to define the main command for a Docker container.
  - If you provide additional arguments when running the container, they will be appended to the `ENTRYPOINT` instruction.
  - The `ENTRYPOINT` cannot be overridden by arguments provided during container runtime unless you use the `--entrypoint` flag.

Command: `docker run ubuntu [COMMAND]`

alt text

### Example:

1.

```
# Example 1: Using ENTRYPOINT
FROM ubuntu
ENTRYPOINT ["sleep"]
```

```
# If you run: docker run my-image 10
# It will execute: sleep 10
```

```
# If you run: docker run my-image sleep2.0 10
# It will not execute: sleep 10 (cannot replace sleep the default ENTRYPOINT) need to use '--entrypoint'
```

```
docker run --entrypoint sleep2.0 my-image 10
```

2.

```
# Example 2: Using CMD
FROM ubuntu
CMD ["sleep", "5"]
```

```
# If you run: docker run my-image
# It will execute: sleep 5
```

```
# If you run: docker run my-image 10
# It will execute: sleep 10 (overriding the default CMD)
```

```
# If you run: docker run my-image sleep2.0 10
# It will execute: sleep2.0 10 (replace the default CMD)
```

3.

```
# Example 3: Using ENTRYPOINT and CMD together
FROM ubuntu
ENTRYPOINT ["sleep"]
CMD ["5"]
```



```
# If you run: docker run my-image
# It will execute: sleep 5 (CMD provides the default argument)

# If you run: docker run my-image 10
# It will execute: sleep 10 (additional arguments are appended)
```

## POD

- we can use `command` to override the `ENTRYPOINT` from pod definition, works like `--entrypoint`
- we can use `args` to override the `CMD` from pod definition

alt text

Example 1 of `command` and `args`

```
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
    command: ["sleep", "5000"]
# OR
  command:
  - "sleep"
  - "5000"
```

Example 2 of `command` and `args`

```
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
    command: ["sleep"]
```

```
args: ["5000"]
```

**Note:** Both the `command` and `args` need to string not number

## Config Map:

### What is a ConfigMap

- ConfigMap can pass the key value pair to the pod
- ConfigMap can be used to pass the configuration to the pod
- it helps in managing the environment variables in the pod definition centrally
- A ConfigMap is an API object used to store non-confidential data in key-value pairs
- Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
- The Pod and the ConfigMap must be in the same namespace.
- ConfigMap does not provide secrecy or encryption
- A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.'
- a ConfigMap has `data` and `binaryData` fields rather than `spec`.
- The `data` field is designed to contain `UTF-8` strings while the `binaryData` field is designed to contain binary data as `base64-encoded` strings.
- The Kubernetes feature `Immutable Secrets and ConfigMaps` provides an option to set individual Secrets and ConfigMaps as immutable.
  - protects you from **accidental** (or unwanted) updates that could cause applications outages
  - **improves performance** of your cluster by significantly reducing load on kube-apiserver, by closing watches for ConfigMaps marked as immutable.

```
apiVersion: v1
kind: ConfigMap
metadata:
  ...
data:
  ...
immutable: true
```

**Note:** Once a ConfigMap is marked as `immutable`, it is not possible to revert this change nor to

mutate the contents of the data or the binaryData field. You can only delete and recreate the ConfigMap. Because existing Pods maintain a mount point to the deleted ConfigMap, it is recommended to recreate these pods.

**Note:** A ConfigMap is not designed to hold large chunks of data. The data stored in a ConfigMap cannot exceed 1 MiB. If you need to store settings that are larger than this limit, you may want to consider **mounting a volume** or use a separate database or file service.

## Creating a ConfigMap

- Imperative
  - Adding config map through command line
- Declarative
  - Adding through the pod definition

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

There are four different ways that you can use a ConfigMap to configure a container inside a Pod:

1. Inside a container command and args
2. Environment variables for a container
3. Add a file in read-only volume, for the application to read
4. Write code to run inside the Pod that uses the Kubernetes API to read ConfigMap

1. Using `env` in pod definition

```

---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
    env:
      - name: USERNAME
        value: admin
      - name: APP_COLOUR
        value: Pink

```

- adding env variables as USERNAME = admin and APP\_COLOUR = pink in pod definition directly.

#### 1. Using envFrom in pod definition

```

---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
    env:
      # Define the environment variable
      - name: APP_COLOUR
        # Notice that the case is different here
        # from the key name in the ConfigMap.
        valueFrom:
          configMapKeyRef:
            name: APP
            # The ConfigMap this value comes from.
            key: frontend
            # The key to fetch.
      - name: UI_PROPERTIES_FILE_NAME
        valueFrom:
          configMapKeyRef:
            name: game-demo
            # different configmap
            key: ui_properties_file_name
            # key to fetch the value of

```

- The value of frontend will be assigned to APP\_COLOUR env variable from APP configmap
- The value of ui\_properties\_file\_name will be assigned to UI\_PROPERTIES\_FILE\_NAME env variable

```
from game-demo configmap
```

1. Mounting the configMap through the `volumes` . To consume a ConfigMap in a volume in a Pod:

- Create a ConfigMap or use an existing one. Multiple Pods can reference the same ConfigMap.
- Modify your Pod definition to add a volume under `.spec.volumes[]` . Name the volume anything, and have a `.spec.volumes[].configMap.name` field set to reference your ConfigMap object.
- Add a `.spec.containers[].volumeMounts[]` to each container that needs the ConfigMap. Specify `.spec.containers[].volumeMounts[].readOnly = true` and `.spec.containers[].volumeMounts[].mountPath` to an unused directory name where you would like the ConfigMap to appear inside the container.
  - If there are multiple containers in the Pod, then each container needs its own `volumeMounts` block, but only **one** `.spec.volumes` is needed per ConfigMap
- Modify your image or command line so that the program looks for files in **that directory**. Each key in the ConfigMap data map becomes the filename under `mountPath`.
- Mounted ConfigMaps are updated automatically

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    configMap:
      name: myconfigmap
```

## Secrets

```
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
```

```
spec:
  priorityClass: high-priority
  containers:
  - name: nginx
    image: nginx
    env:
      - name: APP_COLOUR
        valueFrom:
          secretKeyRef:
```