Certified Kubernetes Administrator

Scheduling

Learning Objectives

By the end of this lesson, you will be able to:

- Present an overview of scheduler and scheduling frameworks
- Describe kube-scheduler, rolling updates, and rollbacks
- List the predicates and priorities in scheduling policies
- State the ten steps to be followed for best performance tuning
- Discuss managing resources for Workloads

TECHNOLOGY

Scheduling Overview

Scheduler Introduction

A Scheduler looks out for Pods that have been freshly created and which have no Nodes assigned to them.

For each Pod discovered, it becomes the responsibility of the Scheduler to find the best Node for that Pod to run on.

Scheduler Introduction

kube-Scheduler is the default Scheduler for Kubernetes that runs as part of the Control Plane.

For each newly created Pod or other unscheduled Pod, kube-Scheduler selects an optimal Node for it to run on.

In a cluster, Nodes that meet the scheduling requirements for a Pod are called viable Nodes.

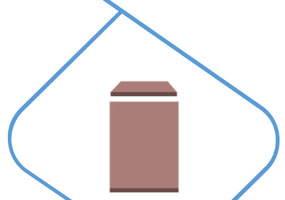
The Scheduler finds viable Nodes for a Pod and runs a set of functions to score viable Nodes. From the viable Nodes, it picks the Node with the highest score to run the Pod.

Scheduling Frameworks

Scheduling Framework

For the Kubernetes Scheduler, the Scheduling Framework is a pluggable architecture.

This Scheduler adds a set of plug-in APIs to the existing Scheduler.



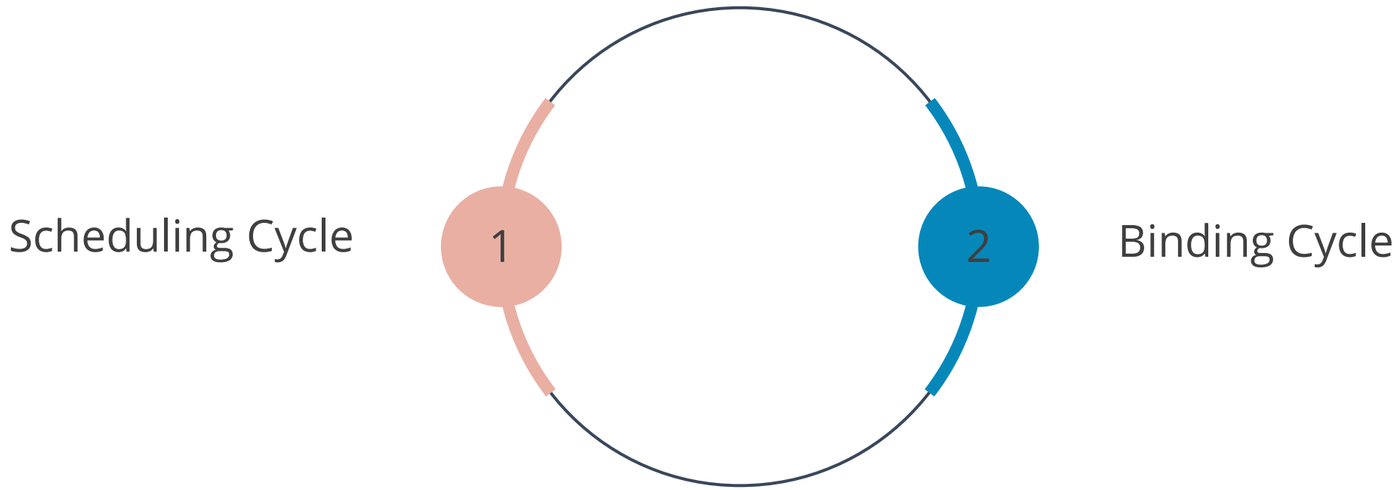
Most Scheduling features can be implemented by APIs, courtesy Plug-ins.

Framework Workflow

Extension Points are defined by the Scheduling Framework.

One or more of them invoke registered Scheduler plug-ins.

Scheduling a Pod happens in two phases (at each attempt):



Binding Cycle and Scheduling Cycle

The Binding Cycle applies the decision to the cluster and the Scheduling Cycle selects a Node for the Pod.

The Binding Cycle and the Scheduling Cycle are referred together as a **Scheduling Context**.

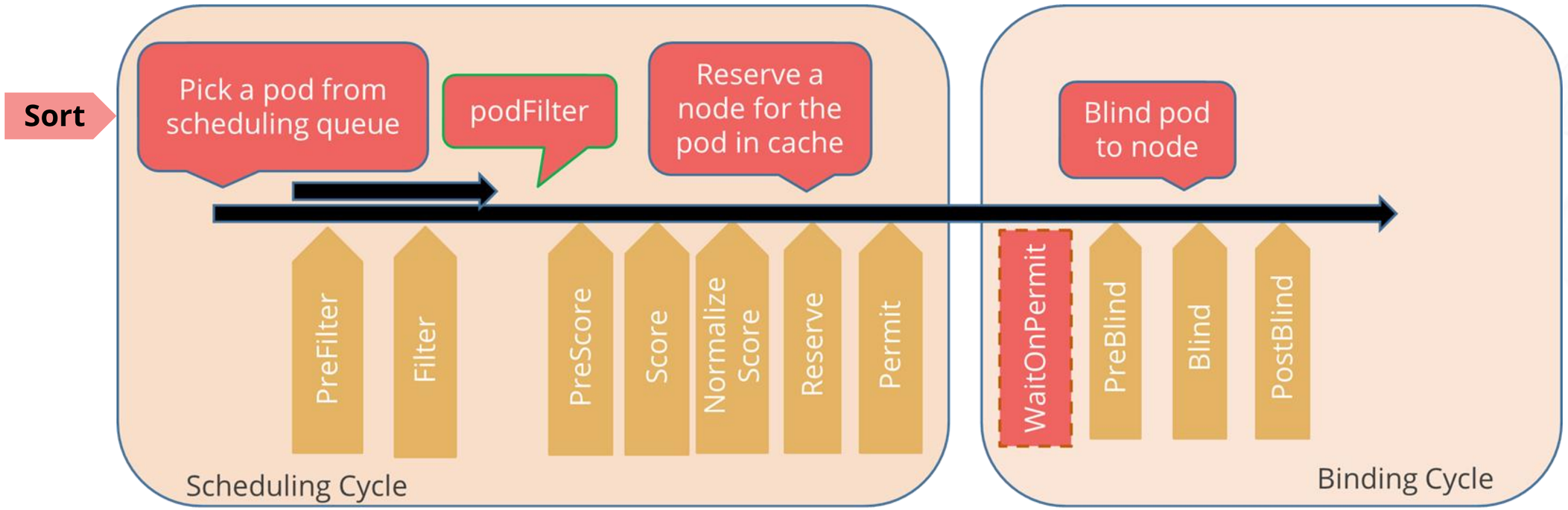
Binding Cycles can be run concurrently. Scheduling Cycles are run serially.

If there is an internal error or a Pod is non-schedulable, either Cycle may be aborted.

Extension Points

This is the Scheduling Context of a Pod.

The Extension Points exposed by the Scheduling Framework are shown:



Pod Scheduling Context

Plug-ins for Scheduling Cycle and Binding Cycle

QueueSort

Sorts the Pod in the scheduling queue

PreFilter

Pre-processes information about the Pod or checks particular conditions that the Pod or the cluster must meet

Filter

Filters out those Nodes that cannot run the Pod

PostFilter

Is called after the Filter phase when no viable Nodes are found for the Pod

Plug-ins for Scheduling Cycle and Binding Cycle

PreScore

Generates a shareable state for Score plug-ins to use

Score

Ranks Nodes that have passed the filtering phase

NormalizeScore

Changes scores before the Scheduler computes a final ranking for Nodes

NormalizeScore

Here is a plug-in that ranks Nodes based on the number of blinking lights they contained:

Demo

```
 func Scorenode(_ *v1.pod, n *v1.node) (int, error) {
    return getBlinkingLightCount(n)
}

#However, the maximum count of blinking lights may be small compared to
#nodescoreMax. To fix this, BlinkingLightScorer should also register for this
#Extension Point.

func NormalizeScores(scores map[string]int) {
    highest : = 0
    for _, score := range scores {
        highest = max(highest, score)
    }
    for Node, score := range scores {
        scores[node] = score*nodescoreMax/highest
    }
}
```

Plug-ins for Scheduling Cycle and Binding Cycle

Reserve

Implements reserve extension; uses reserve and unreserve methods

Permit

Is invoked at the end of the Scheduling Cycle for each Pod in order to prevent or delay the binding to the candidate Node

PreBind

Performs any work necessary before a Pod is bound

Plug-ins for Scheduling Cycle and Binding Cycle

PostBind

Is an Extension Point (informational) and is called after a Pod is successfully bound

Bind

Is called only after PreBind plug-ins have completed their part; binds a Pod to a Node

Plug-in API

To use Extension Points, plug-ins must register first and get configured. The Extension Point interfaces must have the following syntax:

Demo

```
 type Plugin interface {
    Name() string
}

type QueueSortPlugin interface {
    Plugin
    Less(*v1.pod, *v1.pod) bool
}

type PreFilterPlugin interface {
    Plugin
    PreFilter(context.Context, *framework.CycleState, *v1.pod) error
}

// ...
```

Plug-in Configuration

Plug-ins can be disabled or enabled in the Scheduler configuration.

In Kubernetes v1.18 or later

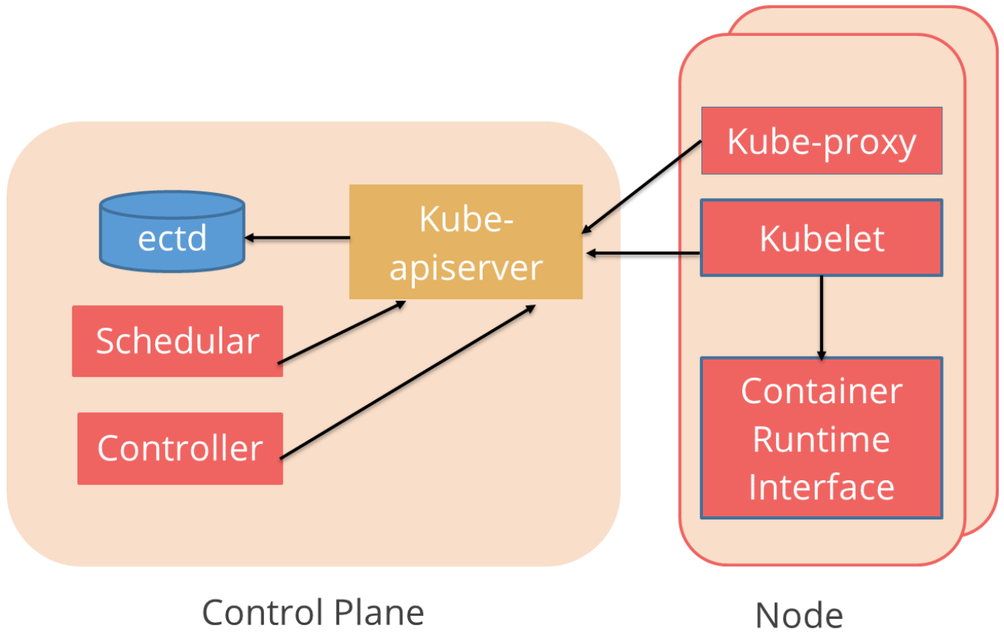
- Most scheduling plug-ins are utilized and enabled by default.
- New scheduling plug-ins can be configured and implemented along with default plug-ins.
- A set of plug-ins can be configured as a Scheduler profile. Post this, multiple profiles may be defined to fit different types of workloads.

Kube-Scheduler

Kubernetes Scheduler

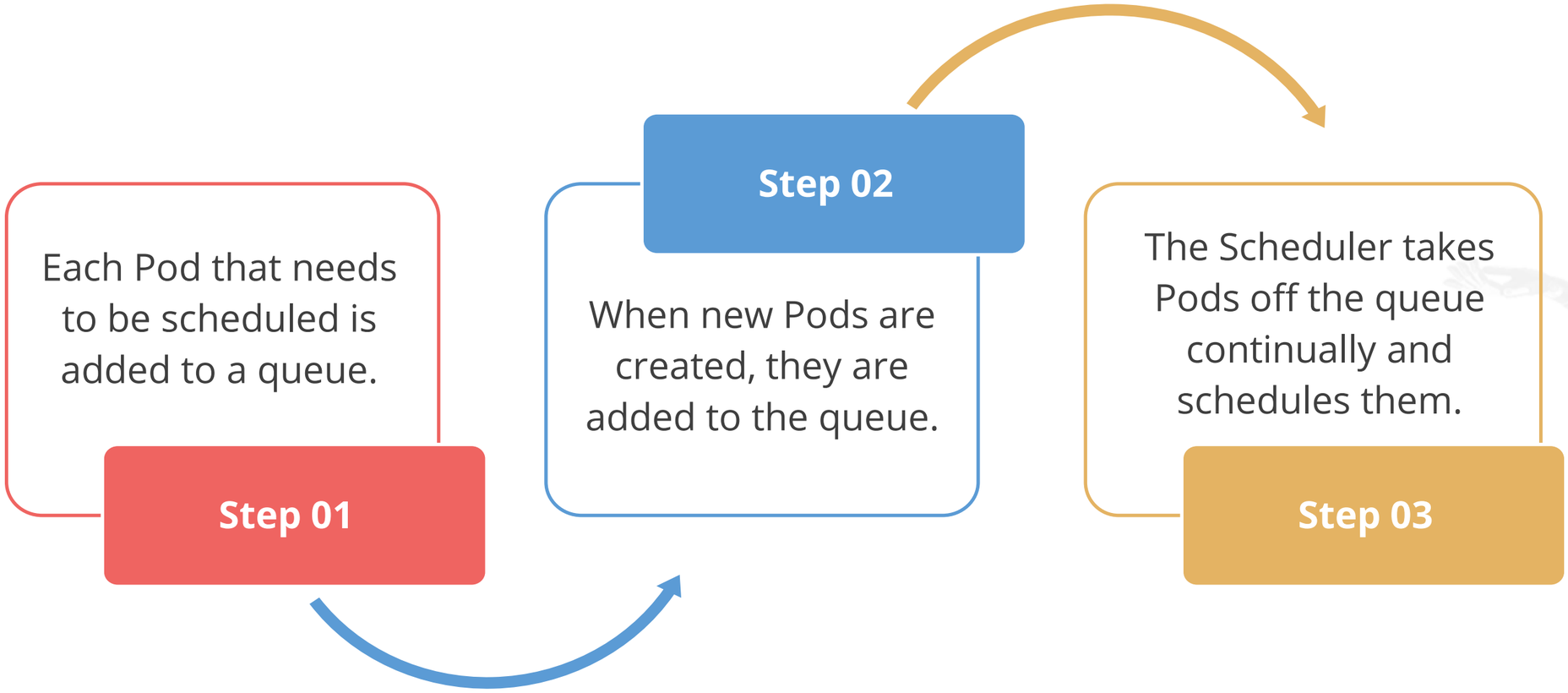
The Kubernetes Scheduler is a Control Plane process, which assigns Pods to Nodes.

High-level view of Kubernetes Scheduler



How Kubernetes Scheduler Works

The Scheduler's role is to ensure that each Pod is assigned to a Node on which it has to run.



Working of Kubernetes Scheduler

The Scheduler's code is at **scheduler.go.**

There are three important facets that need to be considered while scheduling:

1. Code that watches or waits for Pod creation:

Demo

```
 // Run begins watching and scheduling. It waits for cache to be synced, then
starts a goroutine and returns immediately.

func (sched *Scheduler) Run() {
        if !sched.config.WaitForCacheSync() {
                return
        }

        go wait.Until(sched.scheduleOne, 0, sched.config.StopEverything)
```

Working of Kubernetes Scheduler

2. Code responsible for queuing the Pod:

Demo

```
    // queue for Pod that need scheduling
    podQueue *cache.FIFO
```

Demo

```
// Event handler triggering

func (f *ConfigFactory) getNextpod() *v1.pod {
for {
         Pod := cache.Pop(f.podQueue).(*v1.pod)
                  if f.ResponsibleForpod(pod) {
                           glog.V(4).Infof("About to try and schedule Pod
%v", pod.Name)
                           return Pod
                  }
         }
}
```

Working of Kubernetes Scheduler

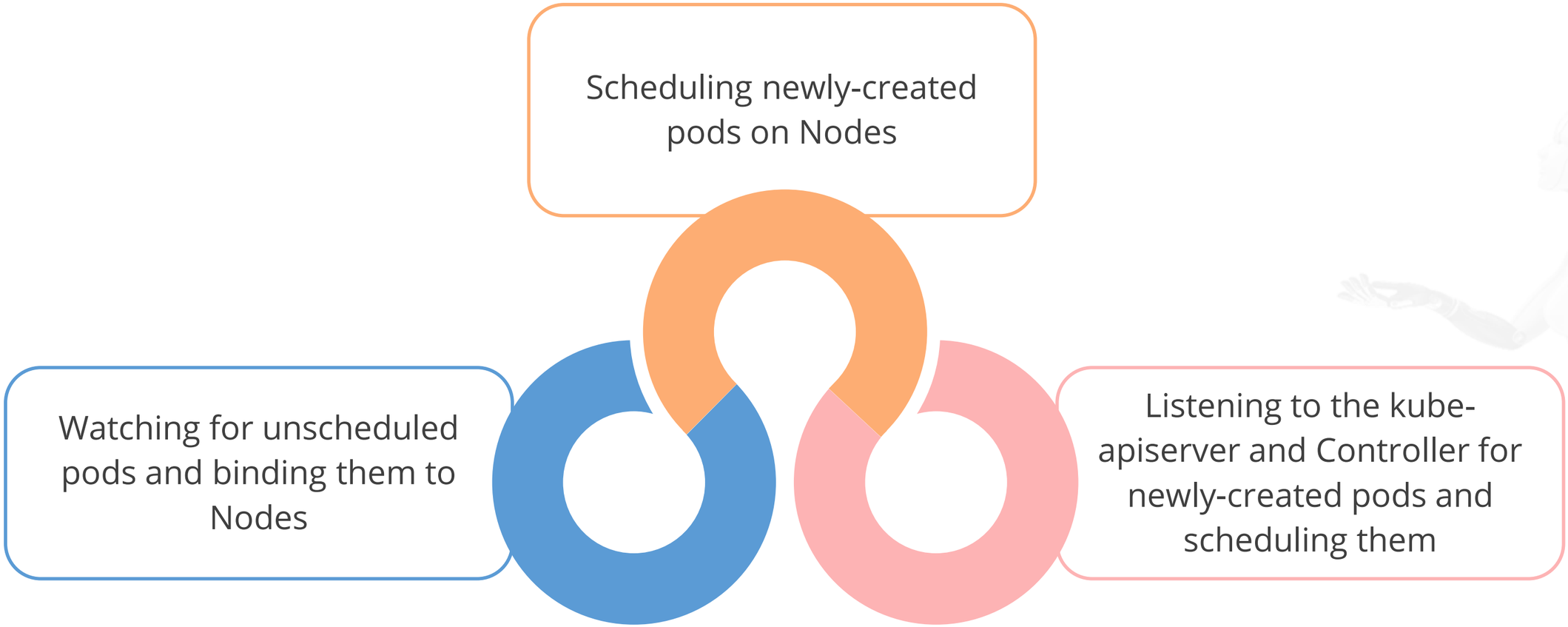
3. Code handling errors:

Demo

```
//error handling
//scheduled Pod cache
        podInformer.Informer().AddEventHandler(
                cache.FilteringResourceEventHandler{
                        FilterFunc: func(obj interface{}) bool {
                                switch t := obj.(type) {
                                case *v1.pod:
                                        return assignedNonTerminatedpod(t)
                                default:

runtime.HandleError(fmt.Errorf("unable to handle object in %T: %T", c, obj))
                                        return false
                                }
                        },
```

Working of Kubernetes Scheduler

The Kubernetes Scheduler is responsible for three tasks.



Kube-Scheduler Commands and Options

Here are some non-deprecated options that can be used with **kube-scheduler [flags]**:

--add_dir_header

Adds the file directory to the header of the log messages, if true

--allow-metric-labels stringToString

The map from metric-label to value allow-list of this label.

--alsologtostderr

Log to standard error as well as files

Understanding Kube-scheduler

Problem Statement:

Creating a custom scheduler and using it to schedule a pod.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate kube-Scheduler in Kubernetes:

1. Creating a new custom scheduler
2. Creating a pod to use the lab-scheduler

Node Selection in Kube-Scheduler

Node selection in Kube-Scheduler

kube-Scheduler selects a Node for the Pod in a two-step operation.

Filtering

kube-Scheduler finds the set of Nodes where it is viable to schedule a Pod

Scoring

It ranks the remaining Nodes to choose the most apt Pod placement

kube-Scheduler assigns the Pod to the Node with the highest ranking

Node selection in Kube-Scheduler

There are two methods to configure the Scheduler's filtering and scoring behavior:

- **Scheduling Policies**
- **Scheduling Profiles**

Understanding Node Selection in Kube-scheduler

Problem Statement:

Understand the working of Node selection in kube-Scheduler.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Node Selection in kube-Scheduler in Kubernetes:

1. Creating a pod with Node Affinity configuration

Perform Rolling Updates on a DaemonSet

DaemonSet Update Strategy

A DaemonSet has two update strategy types:

OnDelete

Users update a DaemonSet template. A new DaemonSet Pod is created only after the old DaemonSet Pod is deleted

RollingUpdate

Users update a DaemonSet template. The old DaemonSet Pod is killed and a new DaemonSet Pod is created automatically

Performing a Rolling Update

To enable the rolling update feature of a DaemonSet, **.spec.updateStrategy.type** must be set to RollingUpdate. The YAML file below specifies a DaemonSet with the update strategy set to RollingUpdate:

Demo

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
 spec:
      tolerations:
```

Demo

```
# this toleration is to have the daemonset runnable on
master nodes;remove it if your masters can't run Pod
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image:
quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Performing a Rolling Update

After verifying the update strategy of the DaemonSet manifest, create the DaemonSet.

Demo

```
kubectl create -f https://k8s.io/examples/controllers/fluentd-daemonset.yaml

//Alternatively, use kubectl apply to create the same DaemonSet if you plan to update the DaemonSet with
kubectl apply.

kubectl apply -f https://k8s.io/examples/controllers/fluentd-daemonset.yaml
```

To check the update strategy, use the following command:

Demo

```
//Check the update strategy of your DaemonSet, and make sure it's set to RollingUpdate:

kubectl get ds/fluentd-elasticsearch -o go-template='{{.spec.updateStrategy.type}}{{"\n"}}' -n kube-system
```

Performing a Rolling Update

Use the following command to check the DaemonSet manifest if the DaemonSet has not been created in the system.

Demo

```
kubectl apply -f https://k8s.io/examples/controllers/fluentd-daemonset.yaml --dry-run=client -o go-template='{{.spec.updateStrategy.type}}{{"\n"}}'
```

The output will be as seen below:

Demo

```
RollingUpdate

//If the output isn't RollingUpdate, go back and modify the DaemonSet object or manifest accordingly.
```

Updating a DaemonSet Template

DaemonSet can be updated by applying a new YAML file, as shown below.

Demo

```
    apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      # this toleration is to have the daemonset runnable
on master Nodes; remove it if your masters can't run Pod
- key: node-role.kubernetes.io/master
```

Demo

```
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
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Commands

To update DaemonSets using configuration files, use **kubectl apply**.

Demo

```
//Declarative command
kubectl apply -f https://k8s.io/examples/controllers/fluentd-daemonset- update.yaml
```

To update DaemonSets using imperative commands, use **kubectl edit**.

Demo

```
//Imperative command
kubectl edit ds/fluentd-elasticsearch -n kube-system
```

To update a container image in the DaemonSet template, use **kubectl set image**.

Demo

```
kubectl set image ds/fluentd-elasticsearch fluentd-
elasticsearch=quay.io/fluentd_elasticsearch/fluentd:v2.6.0 -n kube-system
```

Commands

To know the rollout status of the latest DaemonSet Rolling Update, use **kubectl rollout status**.

Demo

```
kubectl rollout status ds/fluentd-elasticsearch -n kube-system
```

Upon completion of rollout, the output will be as shown below:

Demo

```
daemonset "fluentd-elasticsearch" successfully rolled out
```

Troubleshooting

When the Rolling Update is not completed due to non-availability of resources, find the Nodes that do not have DaemonSet Pods scheduled. This is done by comparing the output of kubectl get Nodes and the output of the following:

Demo

```
kubectl get Pod -l name=fluentd-elasticsearch -o wide -n kube-system
```

Troubleshooting

Broken rollout

If a recent DaemonSet template update is broken; For example, if the Container is crash looping or the Container image does not exist (often due to a typo), DaemonSet rollout will not progress.

Clock skew

If .spec.minReadySeconds is specified in the DaemonSet, Clock Skew between master and Nodes will make the DaemonSet unable to detect the right rollout progress.

Clean up

To delete DaemonSet from a namespace, use the command shown below.

```
kubectl delete ds fluentd-elasticsearch -n kube-system
```

Understanding Rolling Updates

Problem Statement:

Learn how to perform rolling update in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Performing Rolling Update in Kubernetes:

1. Creating a Deployment to rollout a ReplicaSet
2. Updating the Deployment and checking the rollout status

Rollbacks

Performing a Rollback on a DaemonSet

Rolling back a DaemonSet is a three-step process

Step 1: Find the DaemonSet revision and list all the revisions of a DaemonSet

Demo

```
kubectl rollout history daemonset <daemonset-name>

This returns a list of DaemonSet revisions:
daemonsets "<daemonset-name>"
REVISION        CHANGE-CAUSE
1               ...
2               ...
...
```

Performing a Rollback on a DaemonSet

Here are the details of a specific revision:

Demo

```
Kubectl rollout history daemonset <daemonset-name> --revision=1

//This returns the details of that revision:

daemonsets "<daemonset-name>" with revision #1
Pod Template:
Labels:       foo=bar
Containers:
app:
 Image:         ...
 Port:          ...
 Environment:   ...
 Mounts:        ...
 Volumes:       ...
```

Performing a Rollback on a DaemonSet

Step 2: Roll back to a specific revision.

Demo

```
//set the revision number you get from Step 1 in --to-revision

kubectl rollout undo daemonset <daemonset-name> --to-revision=<revision>

//If it succeeds, the command returns:

daemonset "<daemonset-name>" rolled back
```

Performing a Rollback on a DaemonSet

Step 3: Watch the progress of the DaemonSet rollback.

Demo

```
//kubectl rollout undo daemonset tells the server to start rolling back the DaemonSet. The real
rollback is done asynchronously inside the cluster control plane.
//To watch the progress of the rollback:

kubectl rollout status ds/<daemonset-name>
```

After completion of rollback, the output will be as follows:

Demo

```
daemonset "<daemonset-name>" successfully rolled out
```

DaemonSet Revisions

To see what is stored in each revision, find the DaemonSet revision raw resources.

Demo

```
kubectl get controllerrevision -l <daemonset-selector-key>=<daemonset-selector-value>

//This returns a list of ControllerRevisions:

NAME                                CONTROLLER                      REVISION   AGE
<daemonset-name>-<revision-hash>    DaemonSet/<daemonset-name>      1          1h
<daemonset-name>-<revision-hash>    DaemonSet/<daemonset-name>      2          1h
```

Every ControllerRevision stores annotations and template of a DaemonSet revision.

Understanding Rollbacks

Problem Statement:

Understand the working of Rollbacks in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Rollbacks in Kubernetes:

1. Check the rollout history
2. Rollback to a specific version

Scheduler Performance Tuning

Kube-Scheduler

Kube-Scheduler is the default Scheduler for Kubernetes.

- It is tasked with the placement of pods on Nodes in a cluster.
- Viable Nodes for the pods are Nodes in a cluster that meet the scheduling requirements of a Pod.
- In large clusters, users can tune the Scheduler's behavior balancing scheduling outcomes between accuracy and latency.
- Users may configure this setting via the kube-Scheduler, setting percentageOfnodesToScore.

Setting the Threshold

The percentageOfnodesToScore option accepts whole numeric values between 0 and 100.

To change the value, users may first edit the kube-Scheduler configuration file and then restart the Scheduler.

Configuration file path

/etc/kubernetes/config/kube-scheduler.yaml.

Run the command shown below to check the health of kube-Scheduler:

Demo

```
kubectl get Pod -n kube-system | grep kube-scheduler
```

Node Scoring Threshold

After it has found enough of them, kube-Scheduler can stop looking for viable Nodes; this will help improve performance.

Threshold scoring is done as given below:

Users set a threshold as a percentage (whole number) of all the Nodes present in the cluster.

While scheduling, if kube-Scheduler identifies enough viable Nodes to exceed the configured percentage, it moves on to the scoring phase.

If users do not set a threshold, Kubernetes calculates a figure using a linear formula that yields 50% for a 100-node cluster and 10% for a 5000-node cluster.

Example

Here is an example configuration that sets percentageOfnodesToScore to 50%:

Demo

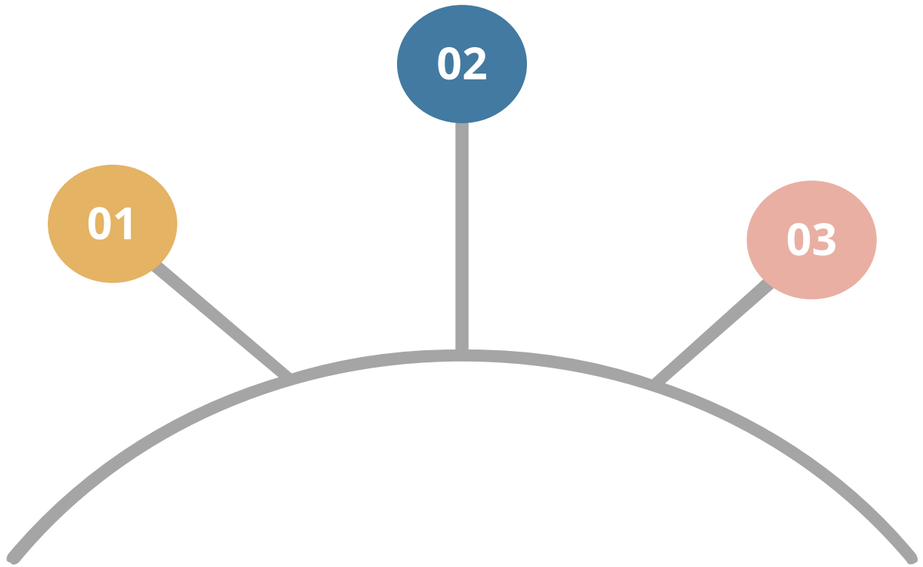
```
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
algorithmSource:
  provider: DefaultProvider

...

percentageOfnodesToScore: 50
```

Tuning percentageOfnodesToScore

The percentageOfnodesToScore must be a value between 1 to 100. The default value is calculated based on the cluster size.

The change is ineffective if a small value is set for percentageOfnodesToScore.

Scheduler checks all Nodes in clusters with less than 50 viable Nodes.



When the cluster has 100 Nodes or less, it is optimal to set the configuration option at default value.

How the Scheduler Iterates over Nodes

If Nodes are in multiple zones, the Scheduler iterates over them to ensure that Nodes from different zones are considered in feasibility checks.

The example shown below has six Nodes in two zones:

Demo

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6

//The Scheduler evaluates feasibility of the Nodes in this order:

Node 1, Node 5, Node 2, Node 6, Node 3, Node 4

//After going over all the Nodes, it goes back to Node 1.
```

Understanding Scheduler Performance Tuning

Problem Statement:

Understand the working of Scheduler Performance Tuning in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Scheduler Performance Tuning in Kubernetes:

1. Checking the status of the scheduler component

Scheduling Policies

Scheduling Policies

A Scheduling Policy may be used to set the Predicates and Priorities that the kube-Scheduler runs to filter and score Nodes respectively.

How to set a Scheduling Policy

Running: kube-scheduler --policy-config-file <filename> or kube-scheduler --policy-configmap <ConfigMap>
and
Using: Policy type

Predicates

Predicates that implement filtering:

1. podFitsHostPorts
2. podFitsHost
3. podFitsResources
4. Matchnodeselector
5. ChecknodePIDPressure
6. ChecknodeDiskPressure
7. NoVolumeZoneConflict
8. NoDiskConflict
9. MaxCSIVolumeCount
10. ChecknodeMemoryPressure
11. ChecknodeCondition
12. podToleratesnodeTaints

Priorities

Priorities that implement scoring:

1. SelectorSpreadPriority
2. InterpodAffinityPriority
3. LeastRequestedPriority:
4. MostRequestedPriority
5. RequestedToCapacityRatioPriority
6. BalancedResourceAllocation
7. nodePreferAvoidpodPriority:
8. TaintTolerationPriority
9. ImageLocalityPriority
10. ServiceSpreadingPriority
11. nodeAffinityPriority
12. EqualPriority
13. EvenpodspreadPriority

Understanding Scheduling Policies

Problem Statement:

Understand the scheduling policies in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Scheduling Policies in Kubernetes:

1. Defining Scheduling Policies

Scheduling Profiles

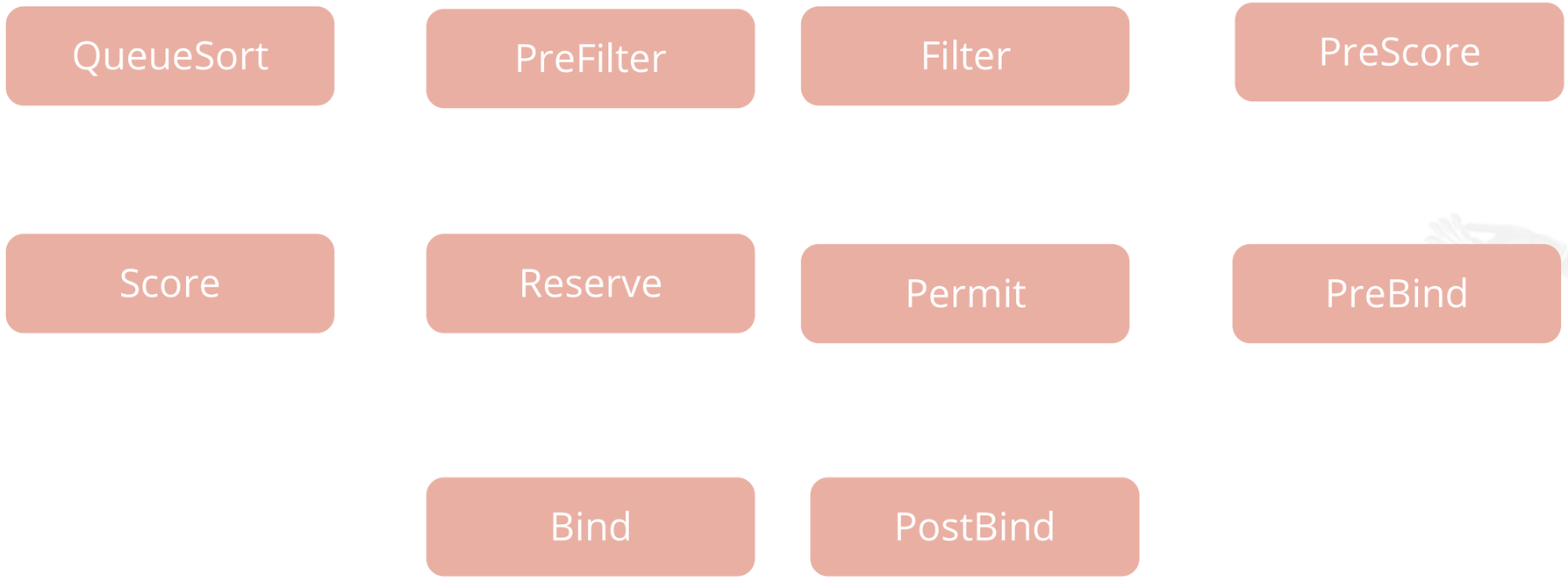
Introduction

A Scheduling Profile permits configuration of different stages of Scheduling in the kube-Scheduler. Each stage is exposed at an Extension Point. Plug-ins provide Scheduling behaviors via the implementation of one or more Extension Points.

A single instance of kube-Scheduler may be configured to run multiple profiles.

Extension Points

Scheduling happens in stages. These are exposed through the following Extension Points:



Extension Points

For each Extension Point, a user may disable specific default plug-ins or enable one.

Here is an example:

Demo

```
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
profiles:
  - plugins:
      score:
        disabled:
        - name: nodeResourcesLeastAllocated
        enabled:
        - name: MyCustomPluginA
          weight: 2
        - name: MyCustomPluginB
          weight: 1
```

Scheduling Plug-ins

The following plug-ins, that implement one or more Extension Points are enabled by default:

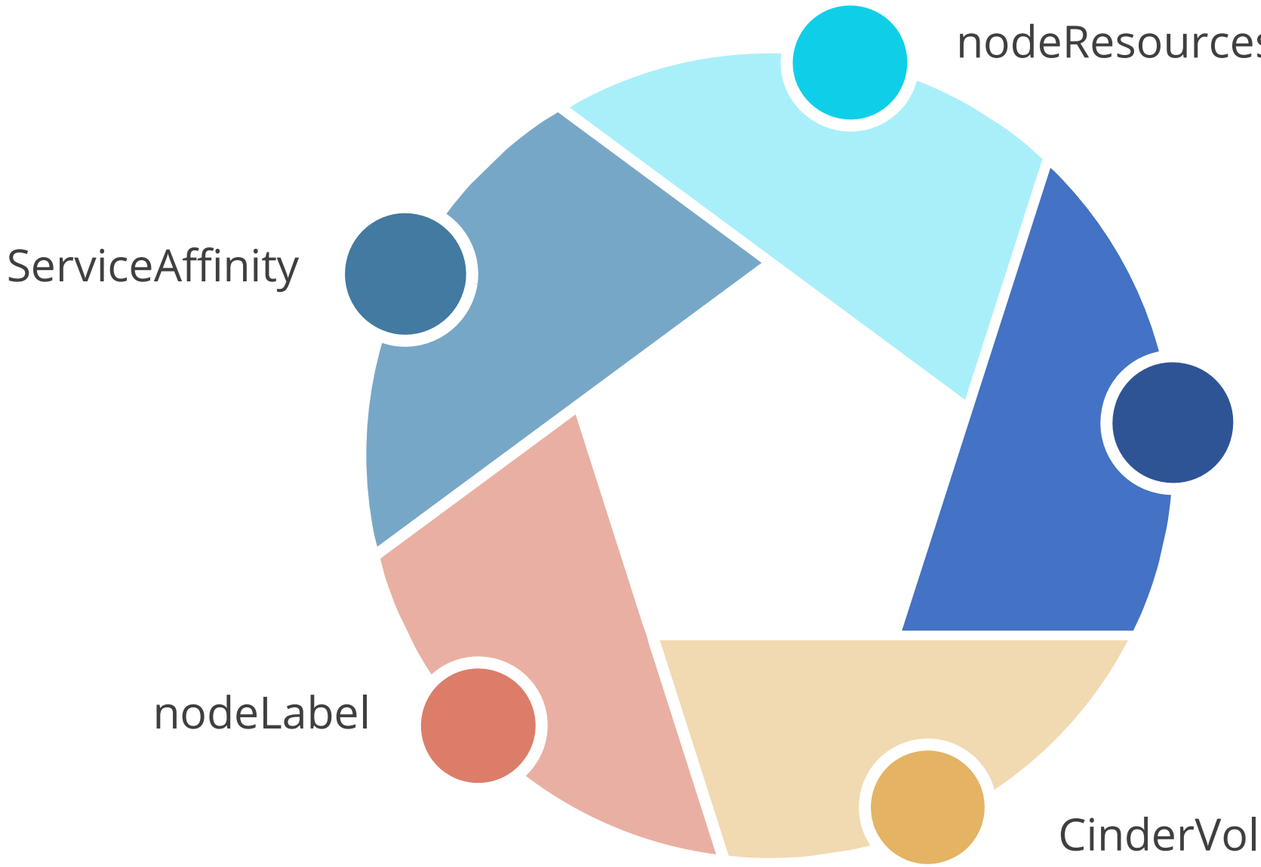
SelectorSpread	podTopologySpread	nodePreferAvoidpod
ImageLocality	nodeName	nodeAffinity
TaintToleration	nodePorts	nodeResourcesBalancedAllocation
points	nodeUnschedulable	nodeResourcesFit

Scheduling Plug-ins

nodeResourcesLeastAllocated	EBSLimits	DefaultBinder
nodeVolumeLimits	PrioritySort	VolumeZone
InterpodAffinity	VolumeRestrictions	AzureDiskLimits
VolumeBinding	GCEPDLimits	NDefaultPreemptionocation

Scheduling Plug-ins

Some plug-ins are not enabled by default and will need to be enabled through the component config APIs.



Multiple Profiles

With a configuration as seen in the sample below, the Scheduler will run with two profiles, one with default plug-ins and the other with all scoring plug-ins disabled.

Demo

```
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: default-scheduler
  - schedulerName: no-scoring-scheduler
    plugins:
      preScore:
        disabled:
        - name: '*'
      score:
        disabled:
        - name: '*'
```


Note

A Pod that must be scheduled according to a specific profile must include the corresponding Scheduler name in its .spec.schedulerName.

Understanding Scheduling Profiles

Problem Statement:

Understand and define scheduling profiles in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Scheduling Profiles in Kubernetes:

1. Define Scheduling Profiles

Topology Management Policies

Topology Manager: Overview

An increasing number of systems are leveraging a combination of CPUs and hardware accelerators to support latency-critical execution and high-throughput parallel computation.

Optimizations in CPU isolation, device locality, and memory are required to bring out the best performance.

Topology Manager is a Kubelet component that coordinates the set of components responsible for optimizations.

How Topology Manager Works

The Topology Manager is a Kubelet component. It acts as a source of truth, supporting other Kubelet components, in making Topology-aligned choices for resource allocation.

Topology Manager provides an interface for components, called Hint Providers, to send and receive Topology information.

It receives Topology information from the Hint Providers as a bitmask denoting NUMA Nodes available and a preferred allocation indication.

Scopes and Policies

The Topology Manager aligns:

- Pods of all QoS classes
- Requested resources that the Hint Provider provides Topology hints for

The scope defines the granularity of resource alignment.

Scopes

The Topology Manager can manage the alignment of resources in two distinct scopes:

1. Container Scope (default)
2. Pod Scope

Policies

The Topology Manager supports four allocation policies:

1. none policy
2. restricted policy
3. single-numa-node policy
4. best-effort policy

Pod Interactions with Policies

In the specs shown below, the Pod runs in the Burstable QoS class. This is due to the requests being less than the limits.

Review the Containers in the following Pod specs:

Demo

```
spec:
  containers:
  - name: nginx
    image: nginx

This Pod runs in the BestEffort QoS class because no resource
requests or limits are specified.

spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

Pod Interactions with Policies

If the selected policy is anything other than none, the Topology Manager will pick up the Pod specifications as shown here:

Demo

```
  spec:
containers:
- name: nginx
  image: nginx
  resources:
    limits:
      memory: "200Mi"
      cpu: "2"
      example.com/device: "1"
    requests:
      memory: "200Mi"
      cpu: "2"
      example.com/device: "1"
```

Pod Interactions with Policies

The Pod with integral CPU requests runs in the Guaranteed QoS class as limits and requests are equal.

Demo

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "300m"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "300m"
        example.com/device: "1"
```

Pod Interactions with Policies

This Pod by sharing CPU request runs in the Guaranteed QoS class. This is because requests are equal to limits.

Demo

```
  spec:
containers:
- name: nginx
  image: nginx
  resources:
    limits:
      example.com/deviceA: "1"
      example.com/deviceB: "1"
    requests:
      example.com/deviceA: "1"
      example.com/deviceB: "1"
```


This Pod runs in the BestEffort QoS class because there are no CPU and memory requests.

Known Limitations

1. The maximum number of NUMA Nodes allowed by the Topology Manager is eight.

2. The Scheduler is not Topology-aware. It is possible to be scheduled on a Node and fail on the Node because of the Topology Manager.

Understanding Topology Management Policies

Problem Statement:

Understand the functioning of Topology Management Policies in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

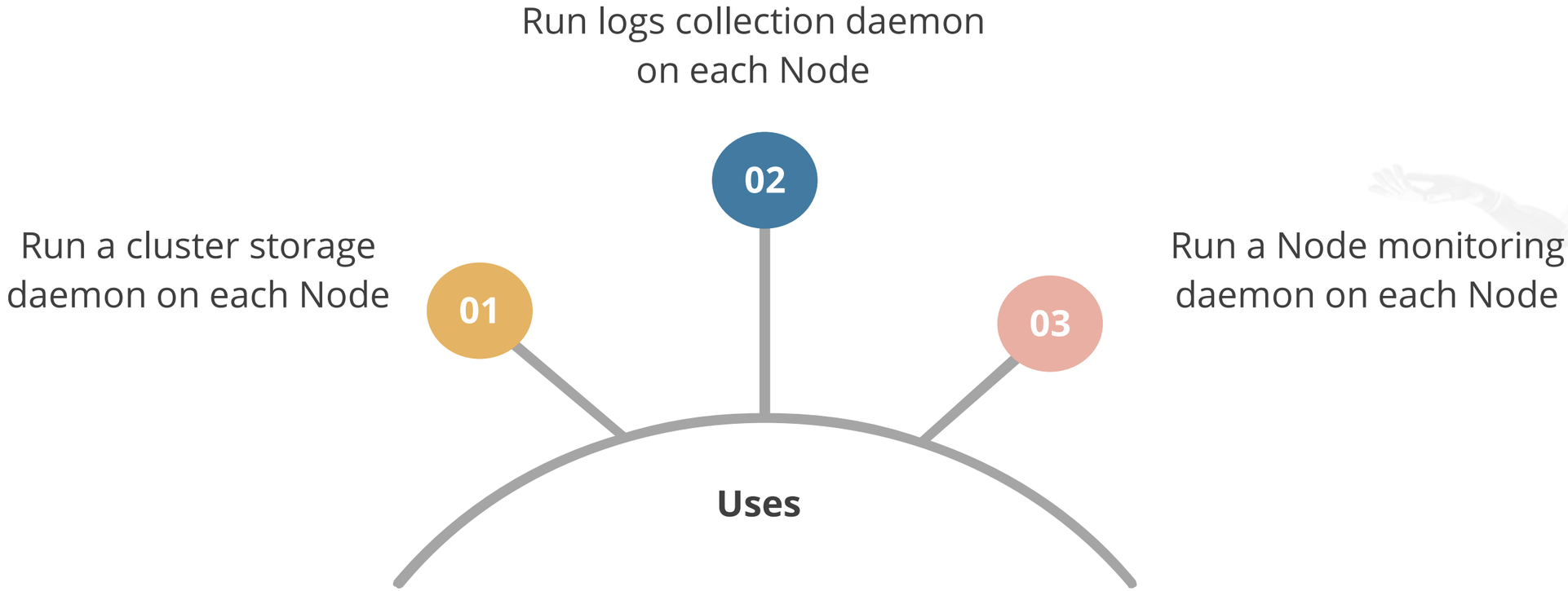
Steps to demonstrate Topology Management Policies in Kubernetes:

1. Setting policy flag in kubelet
2. Restarting the kubelet

DaemonSet

Introduction

A DaemonSet ensures that Nodes run a copy of a Pod.



Writing a DaemonSet Spec

The daemonset.yaml file shown here describes a DaemonSet that runs the fluentd-elasticsearch Docker image:

Demo

```
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:
```

Writing a DaemonSet Spec

Demo

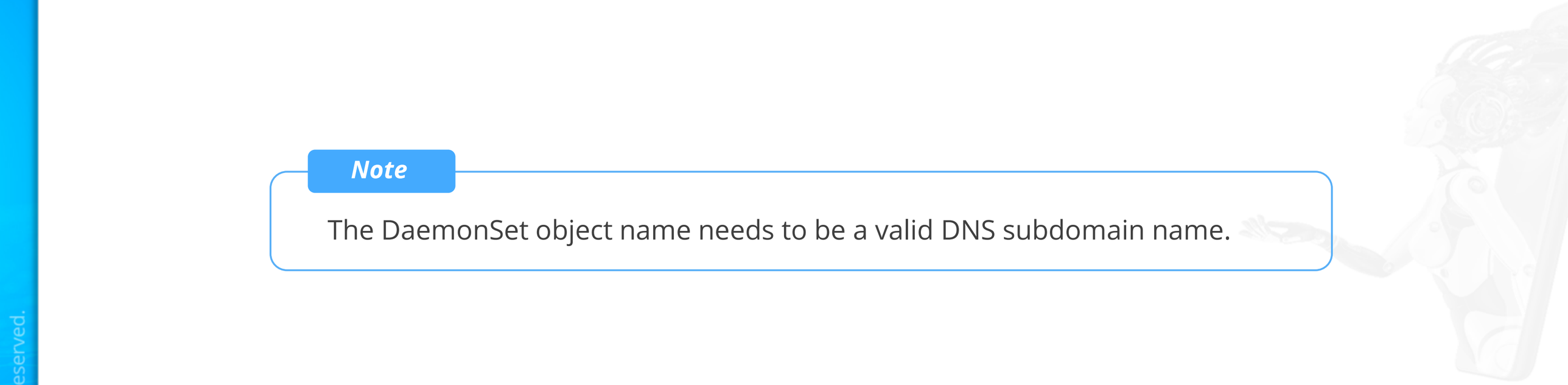
```
# this toleration is to have the daemonset runnable
on master Nodes; remove it if your masters can't run
Pod
- key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image:
quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

Demo

```
volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Required Fields

As with all other Kubernetes config, a DaemonSet needs apiVersion, kind, and metadata fields. A DaemonSet also needs a .spec section.



Note

The DaemonSet object name needs to be a valid DNS subdomain name.

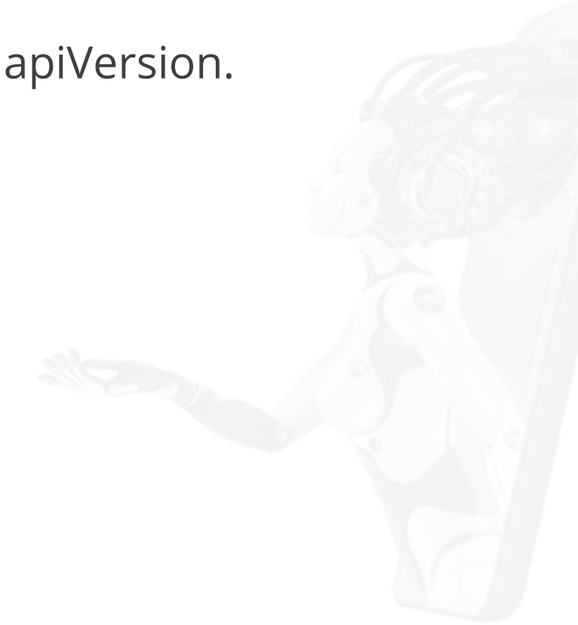
Pod Template

The **.spec.template** is a Pod template.

This template has the same schema as a Pod. It is nested and does not have an apiVersion.

A Pod template in a DaemonSet must set relevant labels as well as necessary fields.

The RestartPolicy must be unspecified or set to Always.



Pod Selector

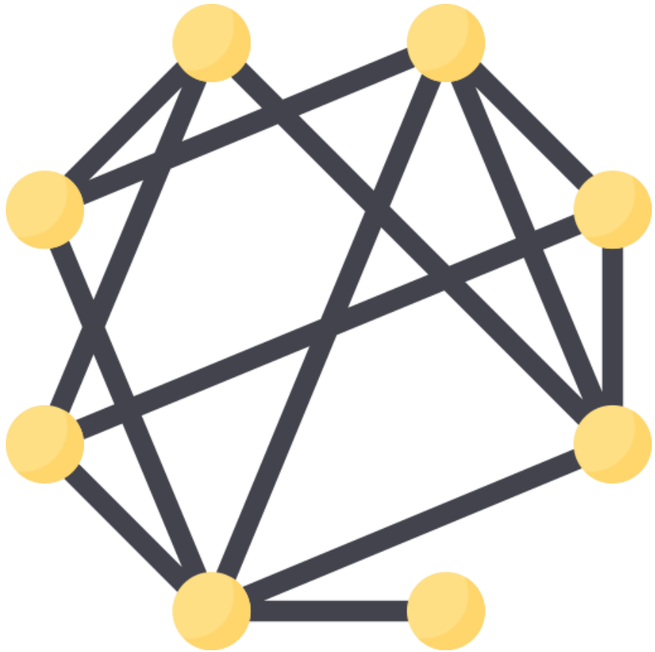
The **.spec.selector** field is a Pod Selector. The .spec.selector object consists of two fields:

1. matchLabels
2. matchExpressions

The result is ANDed when both the fields are specified.

Running Pods on Select Nodes

If **.spec.template.spec.nodeselector** is set, the DaemonSet Controller will create Pods on Nodes, matching the Node Selector.



How Daemon Pods Are Scheduled

The DaemonSet Controller creates and schedules a DaemonSet.

It results in:

1. Inconsistent Pod behavior

2. Loss of control over decisions when preemption is enabled (w.r.t. Scheduling)

Running Pods on Select Nodes

The DaemonSet Controller schedules DaemonSets and binds the Pod to the target host only when creating or modifying DaemonSet Pod. Changes are not made to the spec.template of the DaemonSet.

Demo

```
nodeAffinity:

necessaryDuringSchedulingIgnoredDuringExecution:
    nodeselectorTerms:
    - matchFields:
      - key: metadata.name
        operator: In
        values:
        - target-host-name
```

Taints and Tolerations

The following tolerations are added to DaemonSet Pods automatically that depends on the related features:

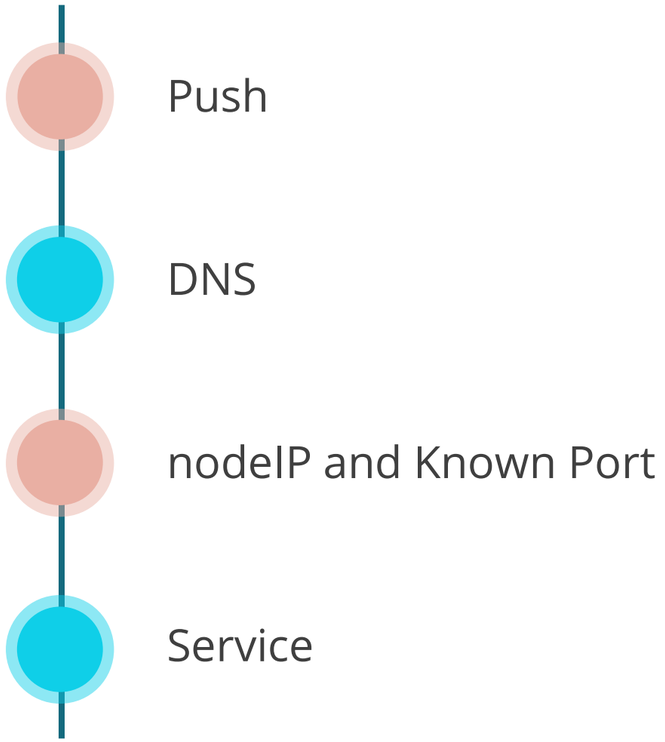
Toleration Key	Effect	Version	Description
node.kubernetes.io/not-ready	NoExecute	1.13+	DaemonSet Pods will not be evicted when there are Node-related problems such as a network partition.
node.kubernetes.io/unreachable	NoExecute	1.13+	DaemonSet Pods will not be evicted when there are Node-related problems such as a network partition.

Taints and Tolerations

Toleration Key	Effect	Version	Description
node.kubernetes.io/disk-pressure	NoSchedule	1.8+	DaemonSet Pods tolerate disk-pressure attributes by default Scheduler.
node.kubernetes.io/memory-pressure	NoSchedule	1.8+	DaemonSet Pods tolerate memory-pressure attributes by default Scheduler.
node.kubernetes.io/unschedulable	NoSchedule	1.12+	DaemonSet Pods tolerate unschedulable attributes by default Scheduler.
node.kubernetes.io/network-unavailable	NoSchedule	1.12+	DaemonSet Pods, which uses host network, tolerate network-unavailable attributes by default Scheduler.

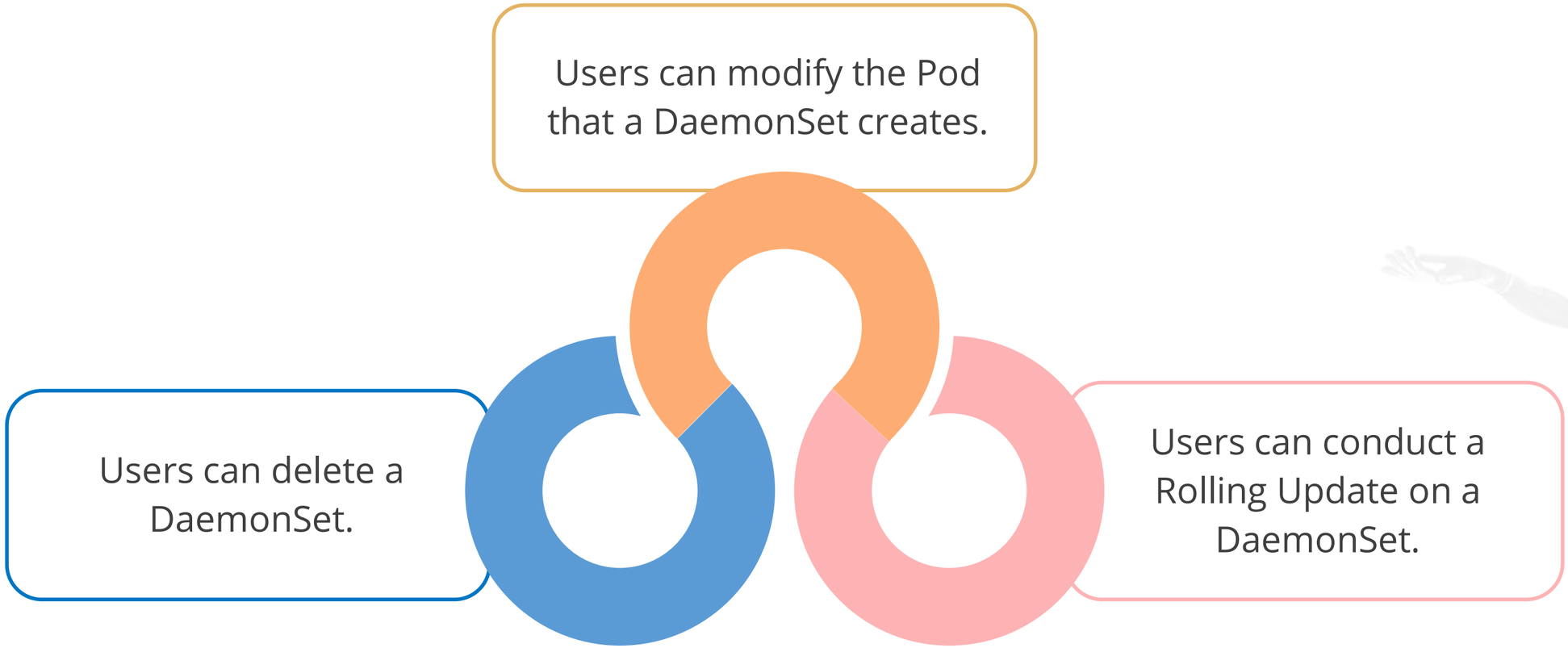
Communicating with Daemon Pod

Here are some patterns for communicating with a Pod in a DaemonSet:



Updating a DaemonSet

The DaemonSet will instantly add Pods to newly matching Nodes if the Node labels are changed. It will also delete Pods from newly not-matching Nodes.



Alternatives to DaemonSet

There are four options to a DaemonSet:

- Init scripts
- Bare Pods
- Static Pods
- Deployments

Understanding DaemonSet

Problem Statement:

Create and know the working of DaemonSets in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate DaemonSet in Kubernetes:

1. Creating a Daemon Set

Pod Overhead

Introduction

Pod Overhead accounts for resources consumed by the Pod infrastructure.

(beyond Container requests and limits)

Pod Overhead is set at admission time
(based on the overhead associated with a Pod's RuntimeClass).

When enabled, it is considered along with the total of all Container resource requests made when scheduling a Pod.

Using Pod Overhead

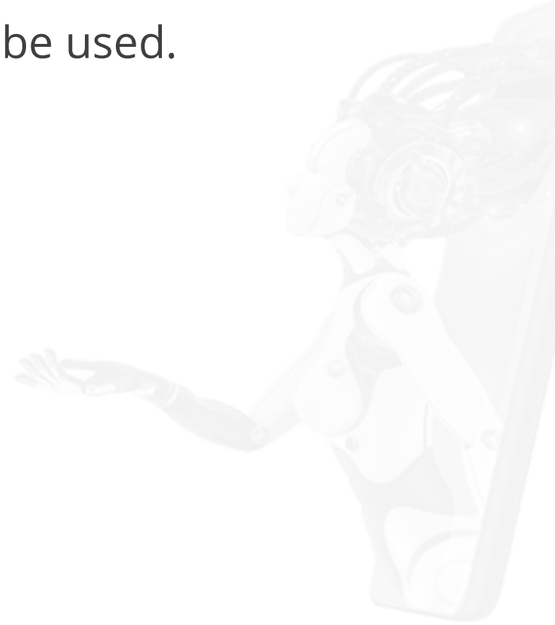
To use the Pod Overhead feature, a RuntimeClass that defines the overhead field is necessary.

Here is a RuntimeClass definition with a virtualizing Container runtime that could be used.

It uses around 120MiB per Pod for the VM and the guest OS.

Demo

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata-fc
handler: kata-fc
overhead:
  podFixed:
    memory: "120MiB"
      cpu: "250m"
```



Using Pod Overhead

Workloads that set the kata-fc RuntimeClass handler factor will take the cpu and memory overheads while calculating resource quotas, Node Scheduling, as well as Pod cgroup sizing.

Here is an example workload, test-pod:

Demo

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  runtimeClassName: kata-fc
  containers:
  - name: busybox-ctr
    image: busybox
    stdin: true
    tty: true
    resources:
```

Demo

```
limits:
        cpu: 500m
        memory: 100Mi
  - name: nginx-ctr
    image: nginx
    resources:
      limits:
        cpu: 1500m
        memory: 100MiB
```

The workload's podpec is updated by the RuntimeClass Admission Controller to include the overhead as described in the RuntimeClass.

Using Pod Overhead

After the RuntimeClass Admission Controller, the user can check the updated podpec.

Demo

```
kubectl get Pod test-pod
 -o jsonpath='{.spec.overhead}'


The output is:

map[cpu:250m memory:120Mi]
```

If a ResourceQuota is defined, the sum of Container requests and overhead field is calculated.

Using Pod Overhead

In this example, verify Container requests are done for the workload:

Demo

```
kubectl get Pod test-pod -o jsonpath='{.spec.containers[*].resources.limits}'

//The total container requests are 2000m CPU and 200MiB of memory:

map[cpu: 500m memory:100Mi] map[cpu:1500m memory:100Mi]
```

To check this against what is being observed by the Node, use:

Demo

```
kubectl describe Node | grep test-pod -B2

The output shows 2250m CPU and 320MiB of memory are requested, which includes podOverhead:

 Namespace                   Name                CPU Requests  CPU Limits   Memory Requests  Memory Limits  AGE
 ---------                   ----                ------------  ----------   ---------------  -------------  ---
 default                     test-pod            2250m (56%)   2250m (56%)  320Mi (1%)       320Mi (1%)     36m
```

Verify Pod Cgroup Limits

You can check the Pod's memory cgroups on the Node where the workload is in operation.

crictl is used on the Node, which provides a CLI for CRI-compatible Container runtimes.

Here’s how crictl is used:

Demo

```
Pod First, on the particular Node, determine the identifier:
# Run this on the Node where the Pod is scheduled
pod_ID="$(sudo crictl Pod --name test-pod -q)"

From this, you can determine the cgroup path for the Pod:
# Run this on the Node where the Pod is scheduled
sudo crictl inspectp -o=json $pod_ID | grep cgroupsPath

The resulting cgroup path includes the pod's pause container. The Pod level cgroup is one
directory above.
       "cgroupsPath": "/kubepod/podd7f4b509-cf94-4951-9417-
d1087c92a5b2/7ccf55aee35dd16aca4189c952d83487297f3cd760f1bbf09620e206e7d0c27a"
```

Verify Pod Cgroup Limits

Here's how you can use **crictl:**

Demo

```
In this specific case, the Pod cgroup path is kubepod/podd7f4b509-cf94-4951-9417-d1087c92a5b2.
Verify the Pod level cgroup setting for memory:

# Run this on the Node where the Pod is scheduled.
# Also, change the name of the cgroup to match the cgroup allocated for your Pod.
 cat /sys/fs/cgroup/memory/kubepod/podd7f4b509-cf94-4951-9417-d1087c92a5b2/memory.limit_in_bytes

This is 320 MiB, as expected:
335544320
```

Observability

A kube_pod_overhead metric aids in corroborating whether PodOverhead is being utilized. It also helps to observe the stability of workloads that run with a defined Overhead.

Observability is not available in the 1.9 release of kube-state-metrics.

Understanding Pod Overheads

Problem Statement:

Understand the working of Pod overhead in Kubernetes

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Pod Overhead in Kubernetes:

1. Defining resource quota
2. Creating a deployment using the defined resource quota

Performance Tuning

Introduction

Performance Tuning maximizes infrastructure utilization, as well as cost reduction, instead of merely scaling up in reaction to the demands of the environment.

Follow these ten steps to tune the performance of your application for best results:

1. Build container-optimized images
2. Define resource profile to match application requirements
3. Configure Node affinities
4. Configure Pod affinities
5. Configure Tolerances and Taints
6. Configure Pod priorities
7. Configure Kubernetes features
8. Optimize etcd cluster
9. Deploy the Kubernetes cluster in proximity to your customers
10. Gain insights from metrics

Build Container-Optimized Images

A Container-Optimized image reduces Container image size, facilitating fast retrievals. Now, Kubernetes can run the resultant Container more efficiently.

A Container-Optimized image must:

- Have a single application or perform one operation
- Have only small images
- Use Container-friendly OSes
- Use multistage builds to keep the app small
- Have health and readiness check Endpoints

Define Resource Profile to Match Application Requirements

Requests must be defined to support Kubernetes as it schedules fine-tuned image efficiently on appropriate Nodes. Limits must also be set on memory, CPU, and other resources.

For instance, to use a Go-based microservice as an email server, designate this resource profile:

Demo

```
resources:
   requests:
     memory: 1Gi
     cpu: 250m
   limits:
     memory: 2.5Gi
     cpu: 750m
```

Configure Node Affinities

It helps in Performance Tuning when the Pod placement is specified.

Node Affinity helps define where the Pods must be deployed. This can be done in two ways:

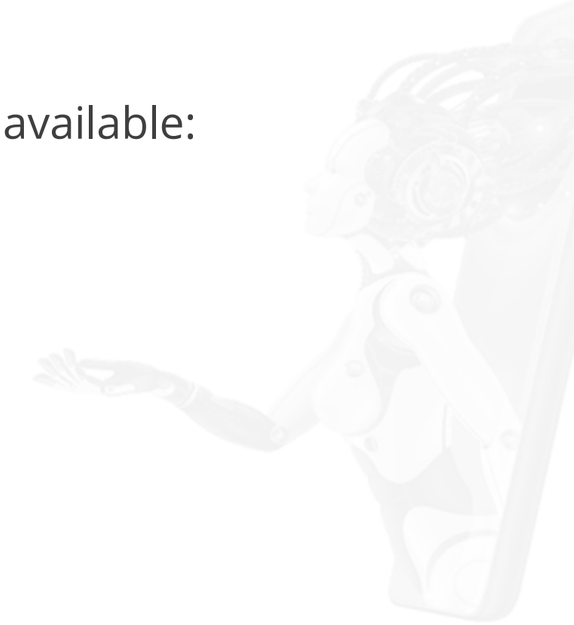
1. Use a nodeSelector with relevant label in the spec section
2. Use nodeAffinity of the Affinity field in the spec section

Configure Pod Affinities

Kubernetes aids in changing and rechanging Pod Affinity configurations in terms of currently operational Pods.

Under **podAffinity** of the Affinity field in the spec section, there are two fields available:

- necessaryDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution

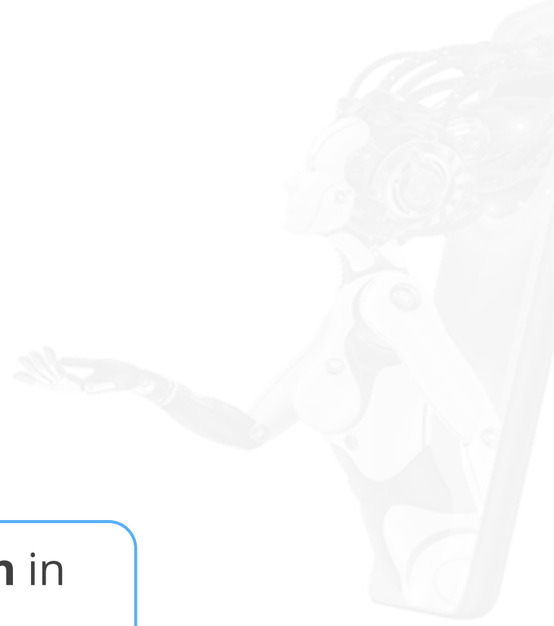


Configure Taints and Tolerations

The behavior of taints is perpendicular to that of Affinity rules. Taints provides certain rules to prevent events such as non-deployment of some Containers to specific Nodes. Taint option can be applied to a certain Node using kubectl.

Demo

```
$ kubectl taint Nodes backup1=backups-only:NoSchedule
```



An exception for a certain Pod can be provided by including **Toleration** in the PodSpec.

Configure Taints and Tolerations

Schedule backups on tainted Nodes by adding these fields in the Pod Spec:

Demo

```
spec:
   tolerations:
     - key: "backup1"
       operator: "Equal"
       value: "backups-only"
       effect: "NoSchedule"
```

Configure Pod Priorities

Kubernetes PriorityClass has a method to define and enforce a certain order.

Here's how a priority class can be created:

Demo

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: highPriority
value: 1000000
globalDefault: false

description: "This priority class should be used for
initial Node function validation."
```


- A high priority Pod is **assigned** a higher value number.
- A priorityClassName may be added under the Pod spec **priorityClassName: highPriority**

Configure Kubernetes Features

Kubernetes uses a feature gate framework, which allows administrators to enable or disable environment features.

Here are the features that boost scaling performance:

CPUManager

podOverhead

Accelerators

Optimize Etcd Cluster

Etcd is the brain of Kubernetes, in the form of a distributed **key=value** database.

Note

Deploying Nodes with solid state disks (SSDs) with low I/O latency and high throughput will optimize database performance.

Deploy Kubernetes Clusters

Deploy Kubernetes clusters in geographical zones closer to end users to ensure low latency.

Note

Kubernetes clusters can be deployed locally.

Feedback from Metrics

Quantitative feedback on system performance supports in tuning the performance of Kubernetes Managed Systems.

Managing Resources

Organizing Resource Configurations

Simplify management of multiple resources by grouping them in the same file. In a YAML file, this can be separated by --- , as shown here:

Demo

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
```

Demo

```yaml
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Organizing Resource Configurations

This code snippet shows you how to create multiple resources:

Demo

```
kubectl apply -f https://k8s.io/examples/application/nginx-app.yaml

service/my-nginx-svc created
deployment.apps/my-nginx created

kubectl apply also accepts multiple -f arguments:

kubectl apply -f
https://k8s.io/examples/application/nginx/nginx-svc.yaml -f
https://k8s.io/examples/application/nginx/nginx-deployment.yaml

And a directory can be specified rather than or in addition to individual files:

kubectl apply -f https://k8s.io/examples/application/nginx/
```

Organizing Resource Configurations

A URL can also be set as a configuration source. This is handy for directly deploying from configuration files checked into GitHub.

Demo

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/website/master/content/en/exam
ples/application/nginx/nginx-deployment.yaml

deployment.apps/my-nginx created
```

Bulk Operations in Kubectl

Kubectl performs several bulk operations, including creating resource, extracting resource names from config files, deleting resources, and so on. Here's how:

Demo

```
kubectl delete -f https://k8s.io/examples/application/nginx-app.yaml

deployment.apps "my-nginx" deleted
service "my-nginx-svc" deleted

In the case of two resources, you can set both resources on the command line using the
resource/name syntax:

kubectl delete deployments/my-nginx services/my-nginx-svc

For larger numbers of resources, you'll find it easier to set the selector (label
query) specified using -l or --selector, to filter resources by their labels:

kubectl delete deployment,services -l app=nginx

deployment.apps "my-nginx" deleted
service "my-nginx-svc" deleted
```

Bulk Operations in Kubectl

Kubectl outputs resource names in the same syntax that it accepts. These chain operations can be executed using $() or xargs.

Demo

```
kubectl get $(kubectl create -f docs/concepts/cluster-administration/nginx/ -o
name | grep service)
kubectl create -f docs/concepts/cluster-administration/nginx/ -o name | grep
service | xargs -i kubectl get {}

NAME           TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
my-nginx-svc   LoadBalancer   10.0.0.208   <pending>     80/TCP       0s
```

Bulk Operations in Kubectl

If resources are organized across several subdirectories within a particular directory, we can perform recursive operations on subdirectories. Do this via specification of **--recursive** or **-R** alongside the **--filename**, **-f flag.**

Demo

```
project/k8s/development
├── configmap
│   └── my-configmap.yaml
├── deployment
│   └── my-deployment.yaml
└── pvc
    └── my-pvc.yaml
```

Demo

```
//Creating the resources in this directory using the
following command will throw an error:
kubectl apply -f project/k8s/development

error: you must provide one or more resources by
argument or filename (.json|.yaml|.yml|stdin)

//Instead, set the --recursive or -R flag with the --
filename,-f flag as such:
kubectl apply -f project/k8s/development --recursive

configmap/my-config created
deployment.apps/my-deployment created
persistentvolumeclaim/my-pvc created
```

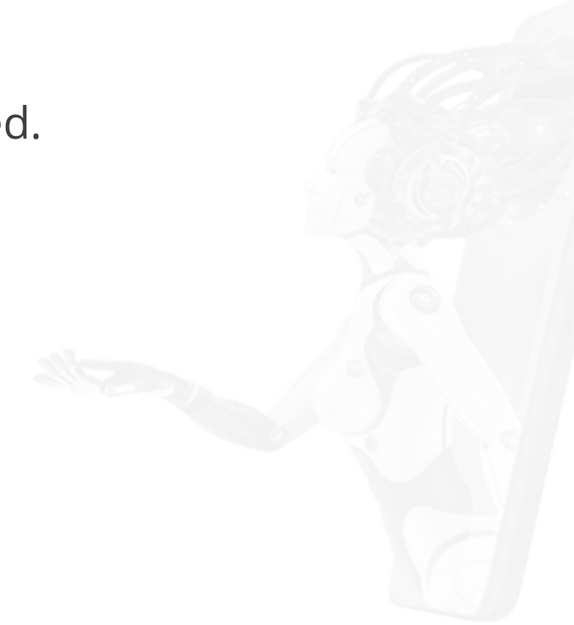
Bulk Operations in Kubectl

The --recursive flag works with any operation that accepts the --filename,-f flag such as Kubectl {create,get,delete,describe,rollout} etc.

The --recursive flag also works when multiple -f arguments are provided.

Demo

```
kubectl apply -f project/k8s/namespaces -f project/k8s/development --
recursive

namespace/development created
namespace/staging created
configmap/my-config created
deployment.apps/my-deployment created
persistentvolumeclaim/my-pvc created
```



Using Labels

Here are a few scenarios that use many labels.

It shows a guestbook that requires each tier to be distinguished.

Demo

```
//Frontend labels
labels:
        app: guestbook
        tier: frontend

//while the Redis master and slave would have different tier labels, and
//perhaps even an additional role label:
    labels:
        app: guestbook
        tier: backend
        role: master
and
    labels:
        app: guestbook
        tier: backend
        role: slave
```

Using Labels

Labels allow slicing and dicing resources along any dimension specified by a label.

Demo

```
kubectl apply -f examples/guestbook/all-in-one/guestbook-all-in-one.yaml
kubectl get Pod -Lapp -Ltier -Lrole

NAME                              READY     STATUS    RESTARTS   AGE       APP         TIER       ROLE
guestbook-fe-4nlpb                1/1       Running   0          1m        guestbook   frontend   <none>
guestbook-fe-ght6d                1/1       Running   0          1m        guestbook   frontend   <none>
guestbook-fe-jpy62                1/1       Running   0          1m        guestbook   frontend   <none>
guestbook-redis-master-5pg3b      1/1       Running   0          1m        guestbook   backend    master
guestbook-redis-slave-2q2yf       1/1       Running   0          1m        guestbook   backend    slave
guestbook-redis-slave-qgazl       1/1       Running   0          1m        guestbook   backend    slave
my-nginx-divi2                    1/1       Running   0          29m       nginx       <none>     <none>
my-nginx-o0ef1                    1/1       Running   0          29m       nginx       <none>     <none>

kubectl get Pod -lapp=guestbook,role=slave

NAME                              READY     STATUS    RESTARTS   AGE
guestbook-redis-slave-2q2yf       1/1       Running   0          3m
guestbook-redis-slave-qgazl       1/1       Running   0          3m
```

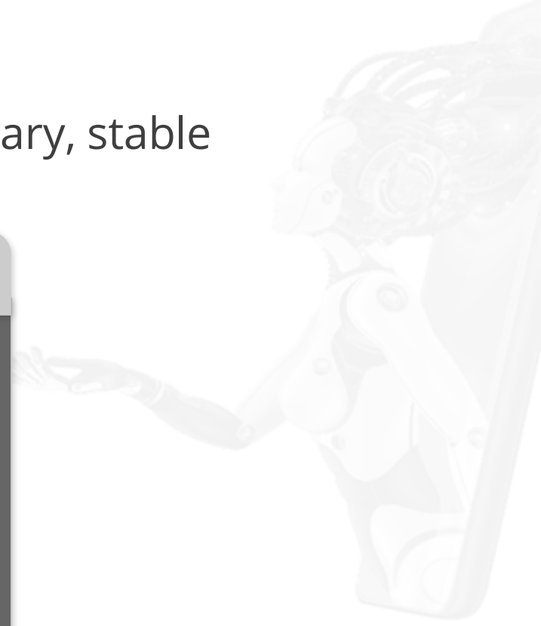
Canary Deployments

Multiple labels may be necessary to identify Deployments of different releases or configurations of the same component. Then, the **Canary** of a new application release is deployed along with the previous release.

For example, a track label can be used to differentiate different releases. The primary, stable release would have a track label with value as stable, as shown below:

Demo

```
name: frontend
     replicas: 3
     ...
     labels:
        app: guestbook
        tier: frontend
        track: stable
     ...
     image: gb-frontend:v3
//and then create a new release of the guestbook frontend that carries the track
label with different value so that two sets of Pod would not overlap:
    name: frontend-canary
     replicas: 1
```



Canary Deployments

A new release of the guestbook frontend, carrying the track label with a different value (Canary), may be created to avoid overlap. Here's how:

Demo

```
...
    labels:
       app: guestbook
       tier: frontend
       track: canary
    ...
    image: gb-frontend:v4
```

The frontend service spans both sets of replicas by selecting the common subset of their labels to ensure that traffic is redirected to both applications.

Demo

```
selector:
    app: guestbook
    tier: frontend
```

Updating Labels

Labels are updated whenever there is a need for an existing Pod and other resources to be relabeled before new resources are created. This can be done by using **kubectl** label.

If all the nginx Pods must be labelled, run the following:

Demo

```
kubectl label Pod -l app=nginx tier=fe

pod/my-nginx-2035384211-j5fhi labeled
pod/my-nginx-2035384211-u2c7e labeled
pod/my-nginx-2035384211-u3t6x labeled

//This first filters all Pod with the label "app=nginx", and then labels them with the
"tier=fe". To see the Pod you labeled, run:

kubectl get Pod -l app=nginx -L tier

NAME                          READY     STATUS    RESTARTS   AGE       TIER
my-nginx-2035384211-j5fhi     1/1       Running   0          23m       fe
my-nginx-2035384211-u2c7e     1/1       Running   0          23m       fe
my-nginx-2035384211-u3t6x     1/1       Running   0          23m       fe
```

Updating Annotations

Annotations are arbitrary non-identifying metadata for retrieval by API clients such as tools, libraries, etc. This can be accomplished by using **kubectl annotate**, as shown below:

Demo

```
kubectl annotate Pod my-nginx-v4-9gw19 description='my frontend running nginx'
kubectl get Pod my-nginx-v4-9gw19 -o yaml

apiVersion: v1
kind: Pod
metadata:
  annotations:
    description: my frontend running nginx
```

Scaling Applications

An application must be scaled when the load on the application grows or shrinks.

Here's how:

Demo

```
// For example, to decrease the number of nginx replicas from 3 to 1 perform:
kubectl scale deployment/my-nginx --replicas=1

deployment.apps/my-nginx scaled

//Now you only have one Pod managed by the deployment.

kubectl get Pod -l app=nginx

NAME                          READY     STATUS    RESTARTS   AGE
my-nginx-2035384211-j5fhi     1/1       Running   0          30m

//To have the system automatically choose the number of nginx replicas as needed, ranging
from 1 to 3, do:

kubectl autoscale deployment/my-nginx --min=1 --max=3

horizontalpodautoscaler.autoscaling/my-nginx autoscaled
```

Kubectl Apply

- Kubectl apply pushes configuration changes to the cluster.
- It compares the version of the configuration that is being pushed with the previous version.
- It applies changes made without overwriting any automated changes to properties that have not been specified.

Demo

```
kubectl apply -f https://k8s.io/examples/application/nginx/nginx-deployment.yaml
deployment.apps/my-nginx configured
```

Kubectl Edit

Resources can be updated using **Kubectl edit**.

Here's how:

Demo

```
kubectl edit deployment/my-nginx

//This is equivalent to first get the resource, edit it in text editor, and then
//apply the resource with the updated version:

kubectl get deployment my-nginx -o yaml > /tmp/nginx.yaml
vi /tmp/nginx.yaml
# do some edit, and then save the file

kubectl apply -f /tmp/nginx.yaml
deployment.apps/my-nginx configured

rm /tmp/nginx.yaml
```

Disruptive Updates

replace --force changes resource fields that cannot be updated once initialized or to implement recursive changes. It deletes and re-creates the resource.

Here's how you can modify the original configuration file:

Demo

```
kubectl replace -f https://k8s.io/examples/application/nginx/nginx-deployment.yaml --
force

deployment.apps/my-nginx deleted
deployment.apps/my-nginx replaced
```

Managing Resources for Workloads

Problem Statement:

Understand the working of managing resources in Kubernetes

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Managing Resources in Kubernetes:

1. Defining multiple resources in a single configuration file

Key Takeaways

- Scheduling ensures that Pods are matched to Nodes so that Kubelet can run them.
- A DaemonSet ensures all eligible Nodes run a copy of a Pod. DaemonSet Pods are created and scheduled by the DaemonSet Controller.
- Kubernetes empowers the user to change and rechange the Pod Affinity configurations in terms of the current running Pod.
- Performing a bulk operation on project/k8s/development will stop at the first level of the directory by default.

Schedule Deployments and Rollout and Rollback Updates

Problem Statement:

For scheduling an application on a specific Node, you have been asked to add labels to Nodes for Node selection. You also have to roll out the updates for the deployed application and rollback the changes in case of any errors or issues.

Steps to Perform:

1. Configuring Node selection
2. Configuring Rolling updates
3. Configuring Rollbacks

LESSON-END PROJECT