TEL4VN

Application Life Cycle management

THAO LUONG
09/2020
https://medium.com/@luongvinhthao

Content

❑ Rolling update and Rollback deployment

❑ Managing and Controlling the Kubernetes Schedule

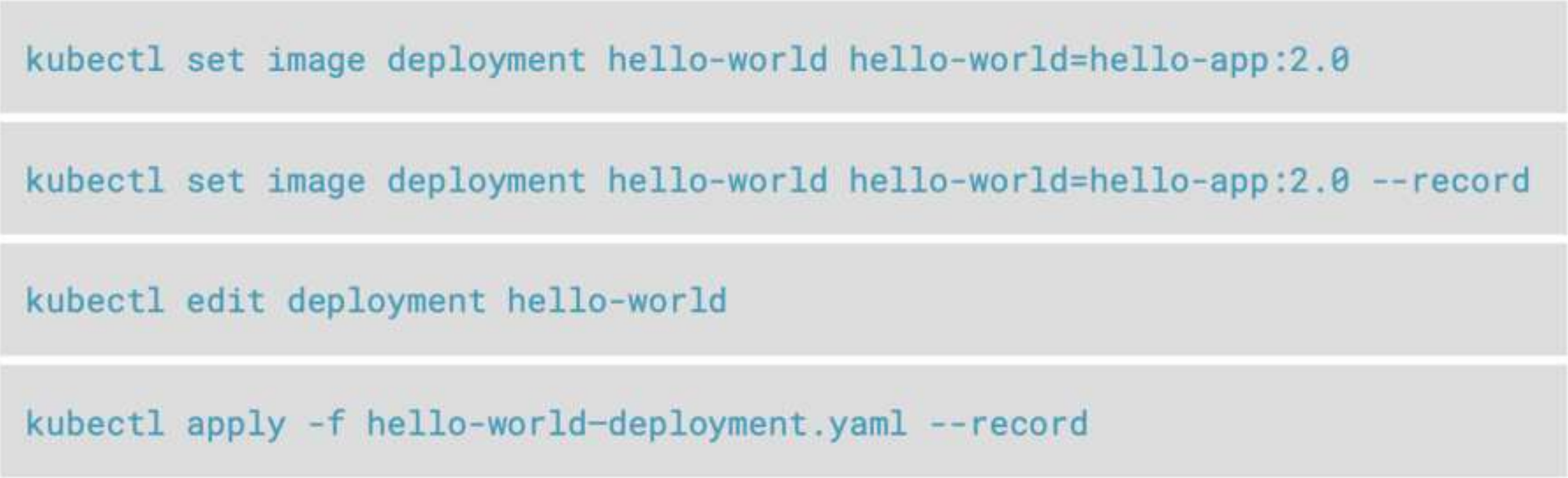
Deploying Application with Deployment

Creating

Updating

Scaling

Update deployment object



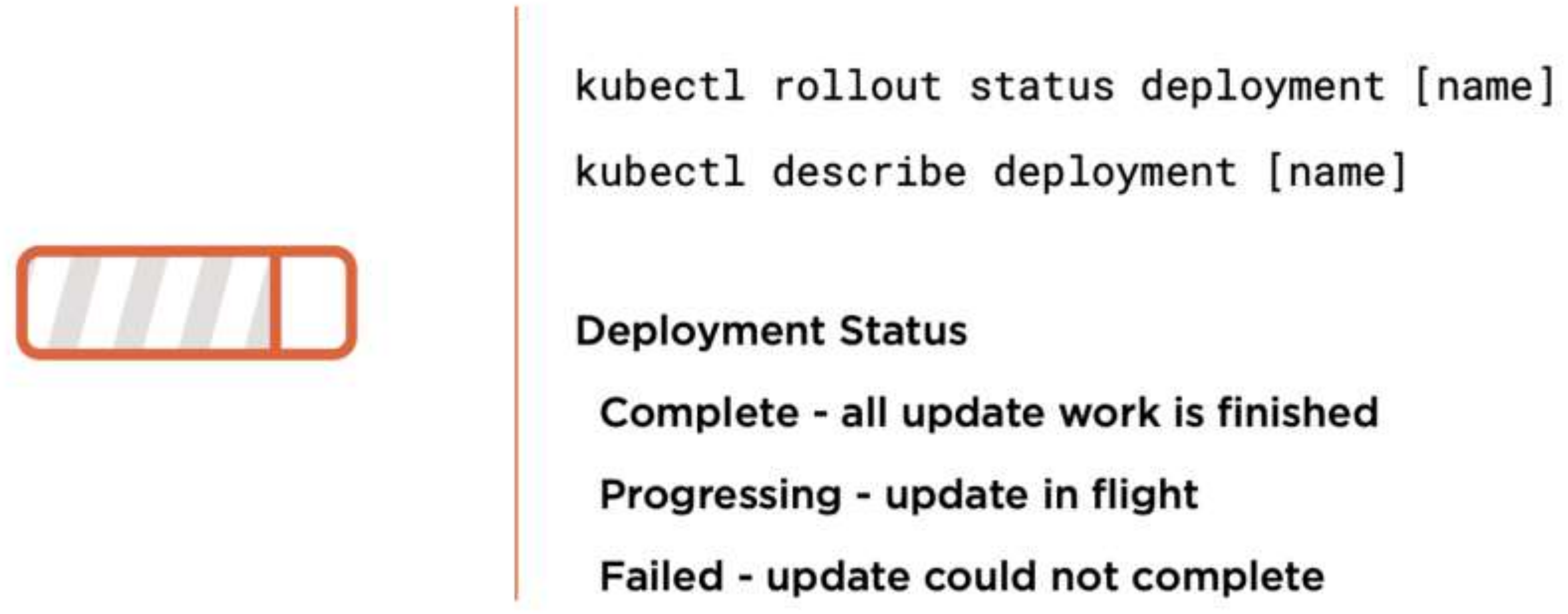
```
kubectl set image deployment hello-world hello-world=hello-app:2.0
```

```
kubectl set image deployment hello-world hello-world=hello-app:2.0 --record
```

```
kubectl edit deployment hello-world
```

```
kubectl apply -f hello-world-deployment.yaml --record
```

Checking deployment status



```
kubectl rollout status deployment [name]

kubectl describe deployment [name]
```

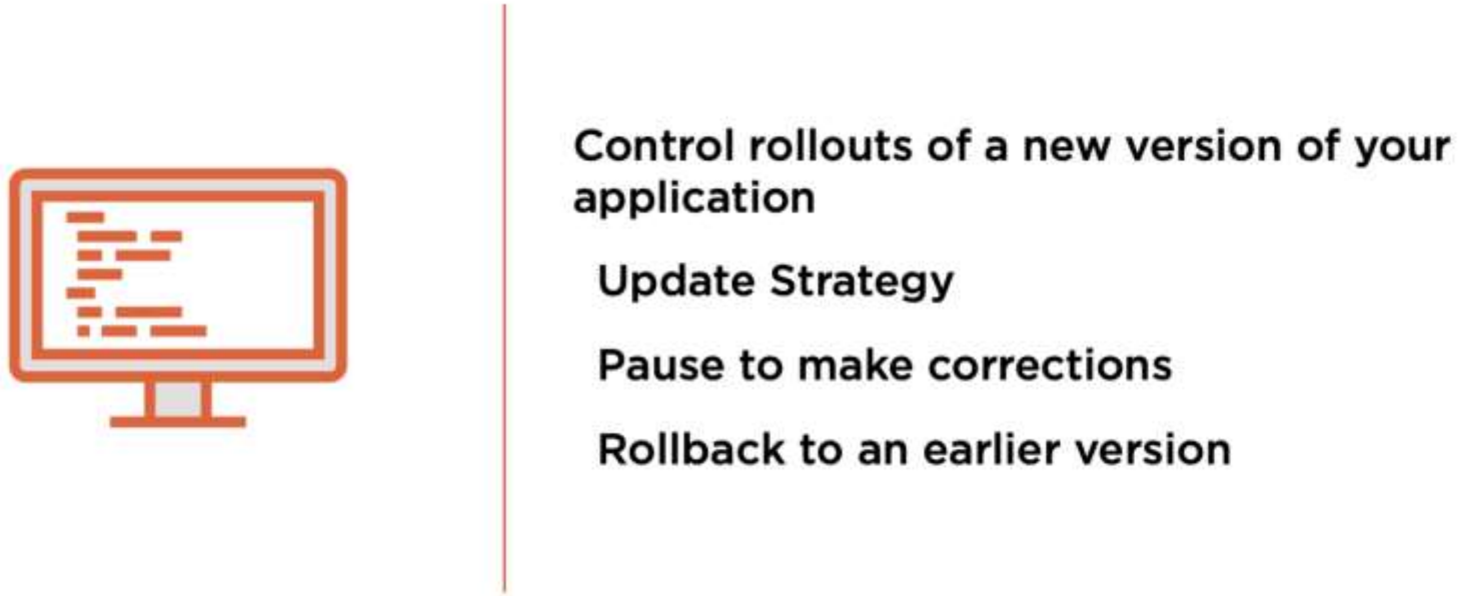
Deployment Status

Complete - all update work is finished

Progressing - update in flight

Failed - update could not complete

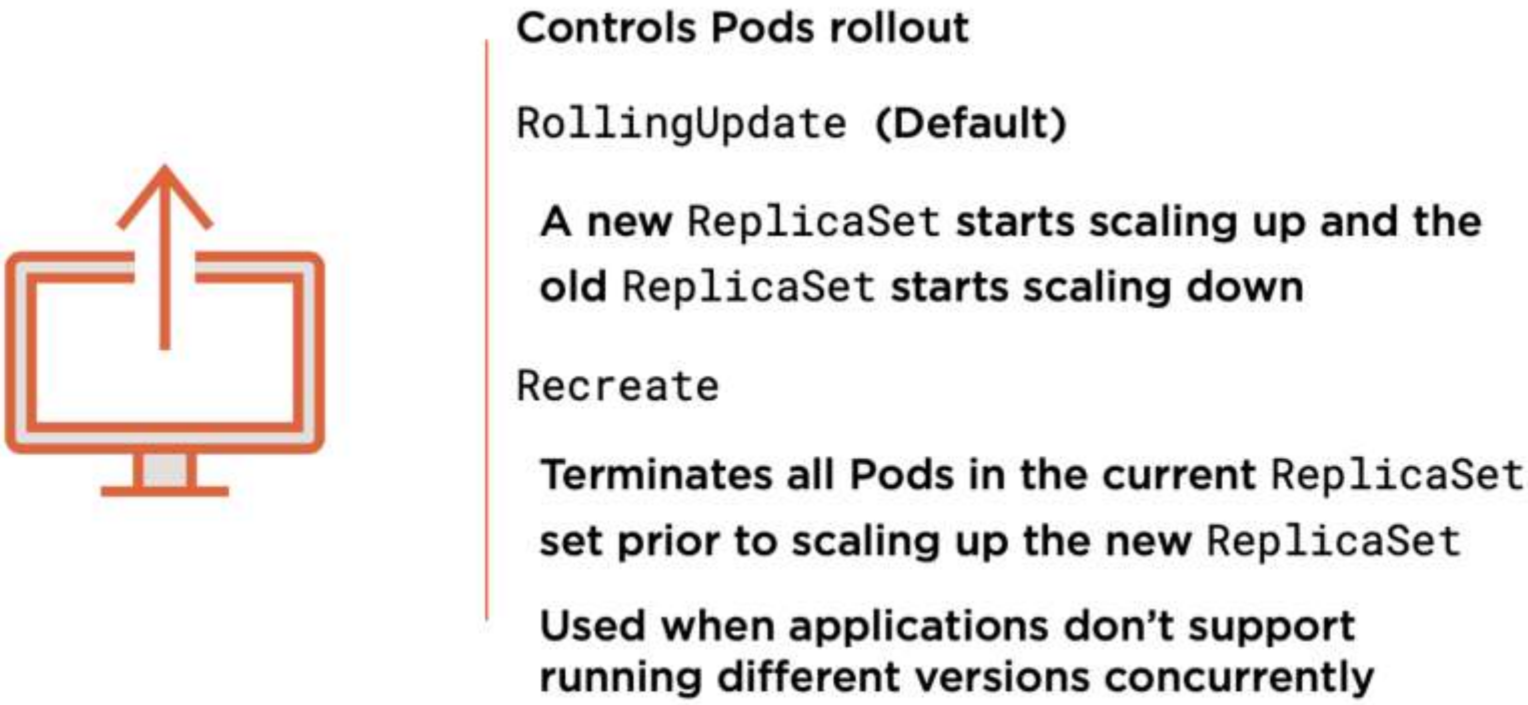
Using Deployments to Change State



Control rollouts of a new version of your application

- Update Strategy
- Pause to make corrections
- Rollback to an earlier version

Controlling Rollouts With Update Strategy



Controls Pods rollout

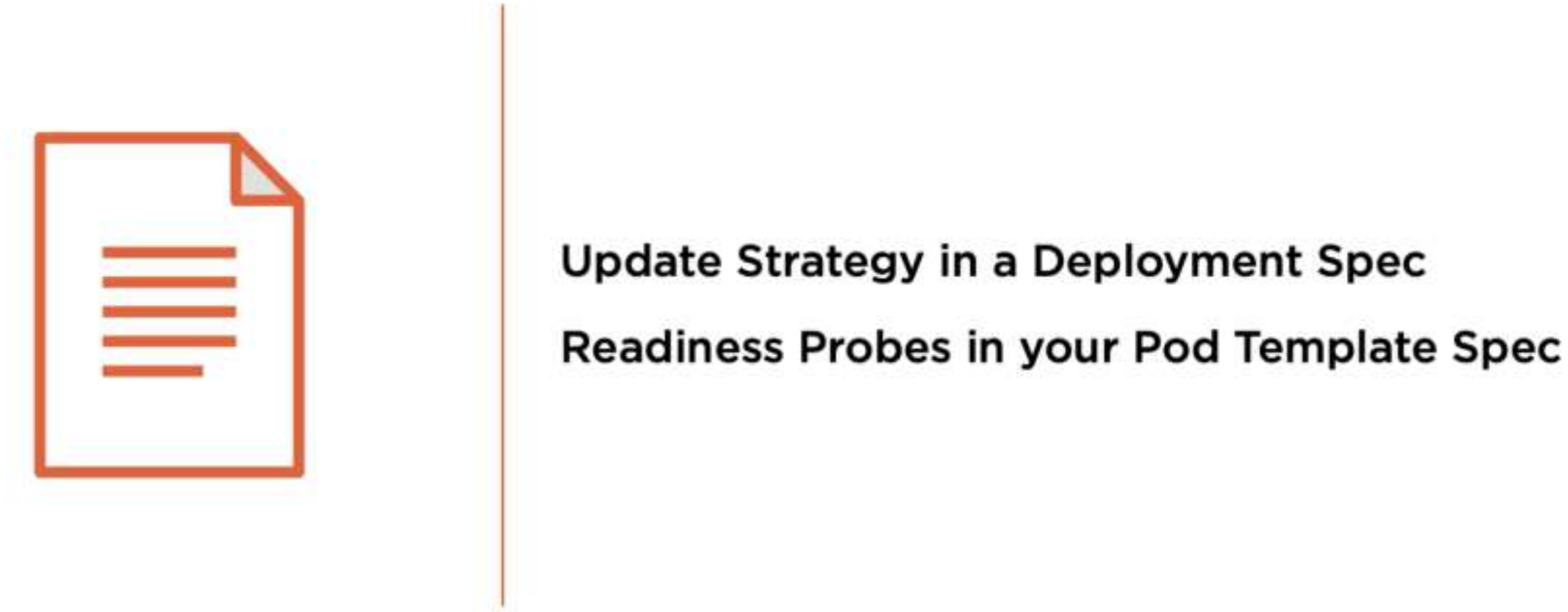
`RollingUpdate` **(Default)**

- **A new** `ReplicaSet` **starts scaling up and the old** `ReplicaSet` **starts scaling down**

`Recreate`

- **Terminates all Pods in the current** `ReplicaSet` **set prior to scaling up the new** `ReplicaSet`
- **Used when applications don't support running different versions concurrently**

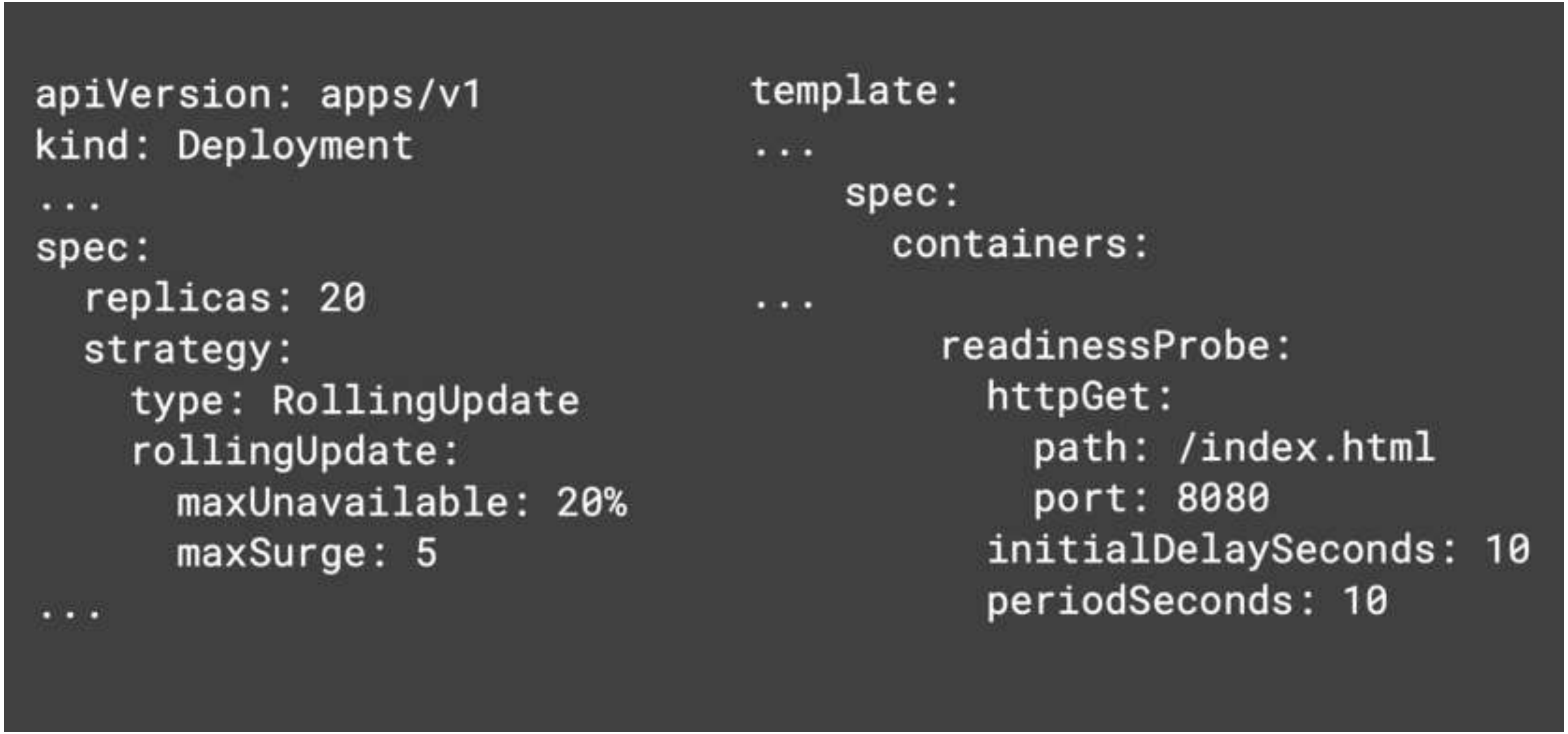
Controlling Deployment Rollouts



Update Strategy in a Deployment Spec

Readiness Probes in your Pod Template Spec

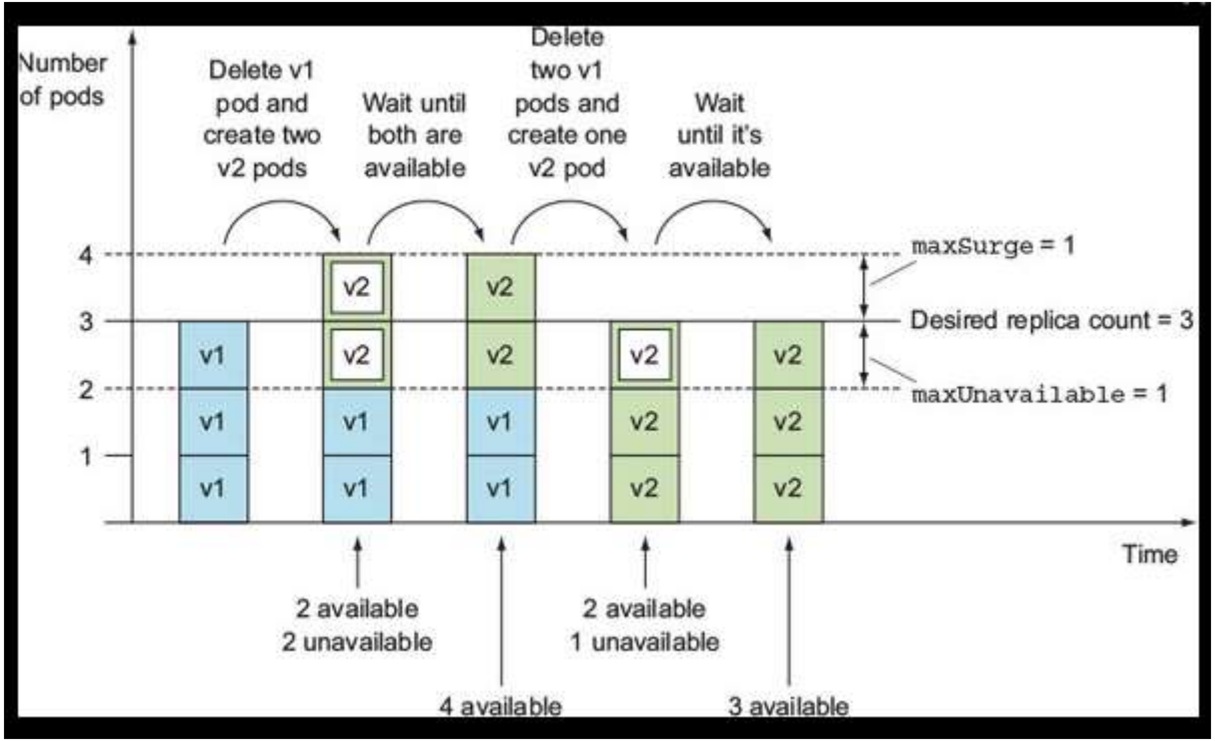
Update Strategy



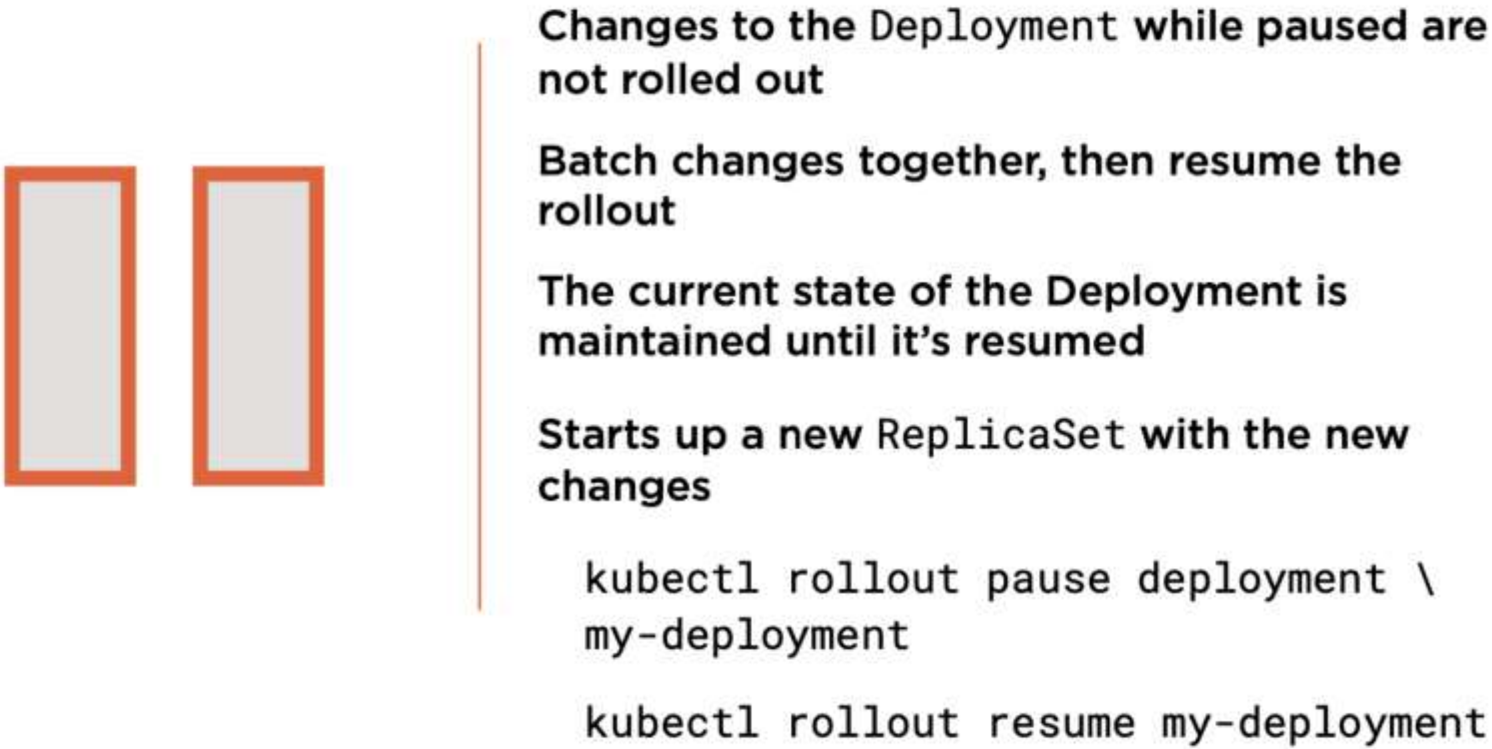
```
apiVersion: apps/v1
kind: Deployment
...
spec:
  replicas: 20
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 20%
      maxSurge: 5
...
```

```
template:
...
    spec:
      containers:
...
        readinessProbe:
          httpGet:
            path: /index.html
            port: 8080
          initialDelaySeconds: 10
          periodSeconds: 10
```

Rolling update with maxSurge = 1 and maxUnavailable =1


Number of pods
Delete v1 pod and create two v2 pods
Wait until both are available
Delete two v1 pods and create one v2 pod
Wait until it's available
4
3
2
1
v1
v2
maxSurge = 1
Desired replica count = 3
maxUnavailable = 1
Time
2 available 2 unavailable
4 available
2 available 1 unavailable
3 available

Pausing and Resuming a Deployment



Changes to the `Deployment` while paused are not rolled out

Batch changes together, then resume the rollout

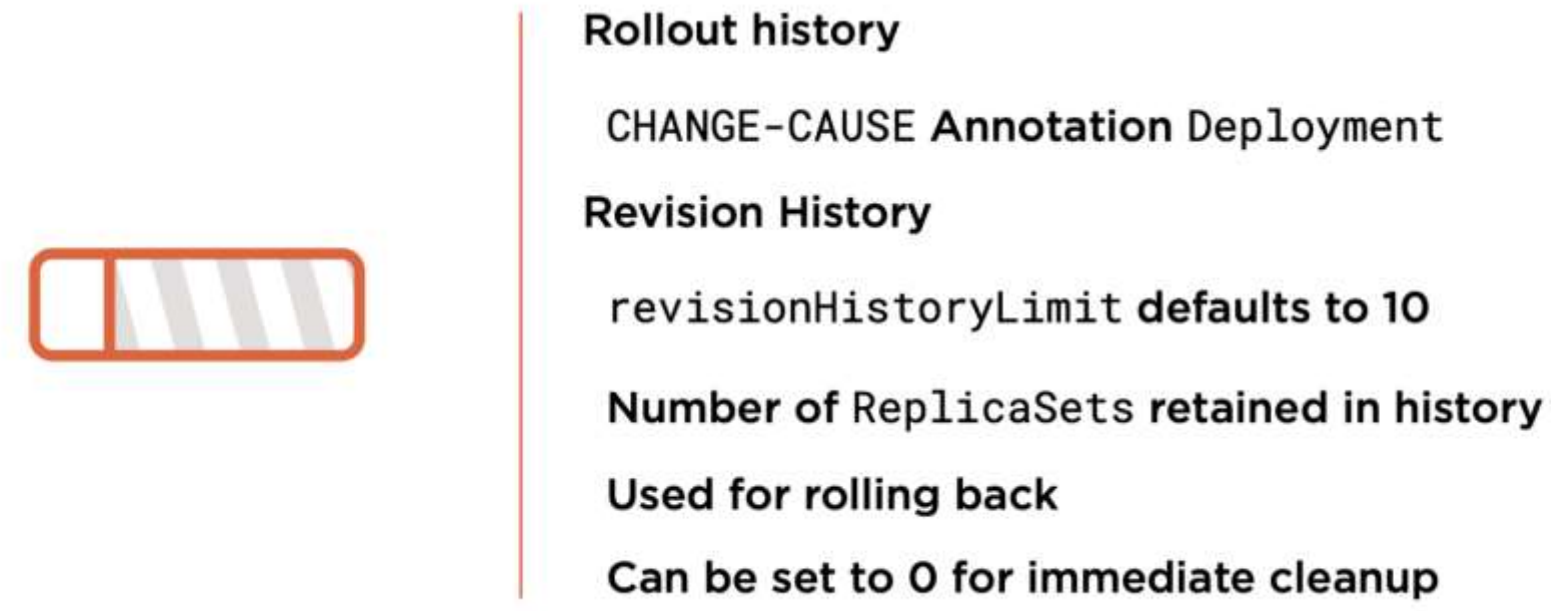
The current state of the Deployment is maintained until it’s resumed

Starts up a new `ReplicaSet` with the new changes

```
kubectl rollout pause deployment \
my-deployment
```

```
kubectl rollout resume my-deployment
```

Rolling Back a Deployment



Rollout history

`CHANGE-CAUSE` **Annotation** `Deployment`

Revision History

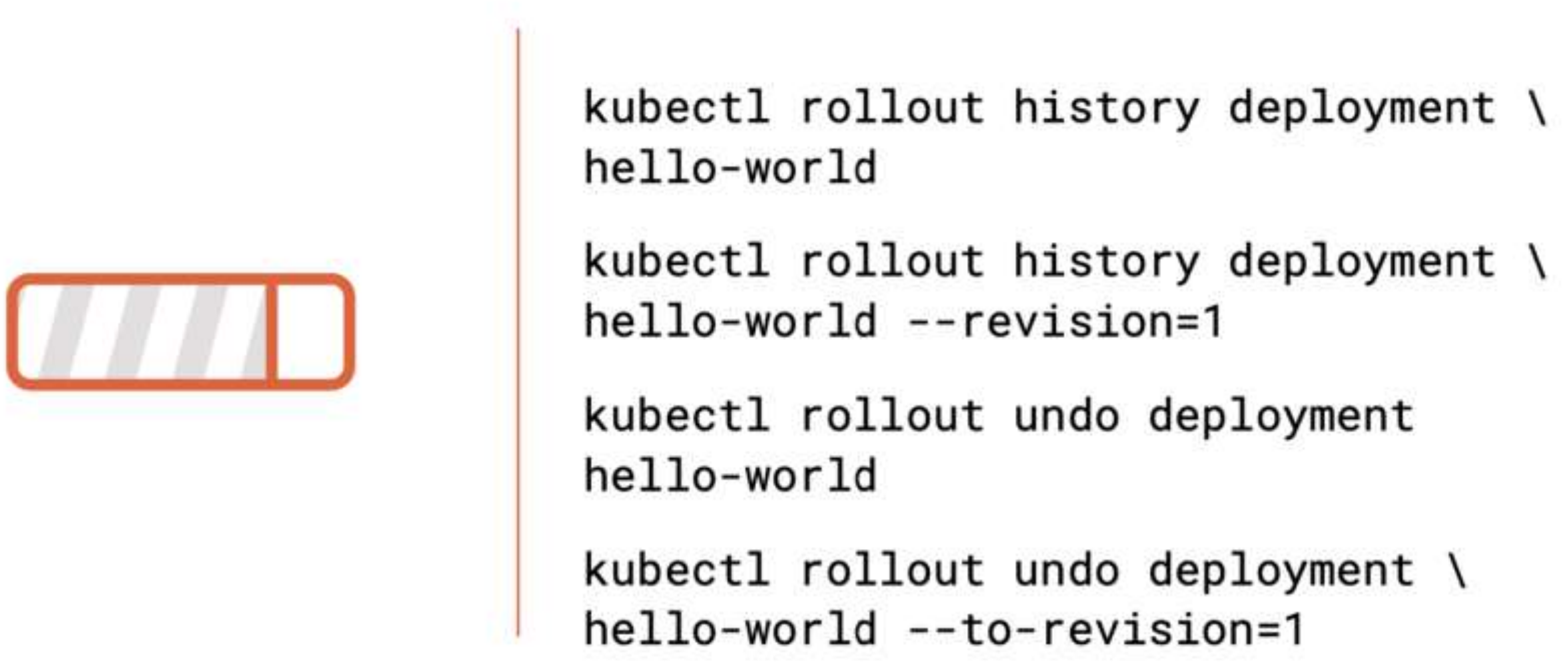
`revisionHistoryLimit` **defaults to 10**

Number of `ReplicaSets` **retained in history**

Used for rolling back

Can be set to 0 for immediate cleanup

Rolling Back a Deployment(cont’)



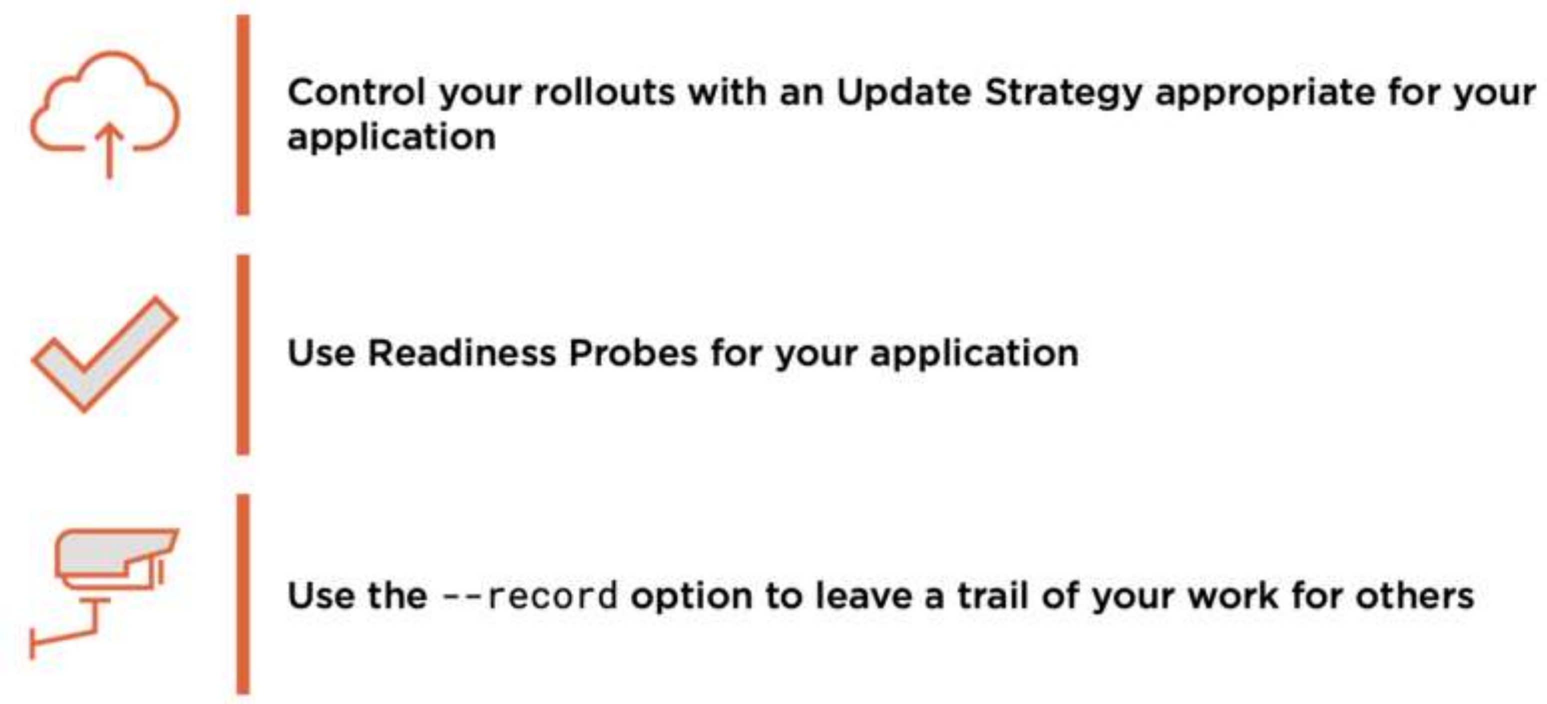
```
kubectl rollout history deployment \
hello-world

kubectl rollout history deployment \
hello-world --revision=1

kubectl rollout undo deployment
hello-world

kubectl rollout undo deployment \
hello-world --to-revision=1
```

Deployment Tips

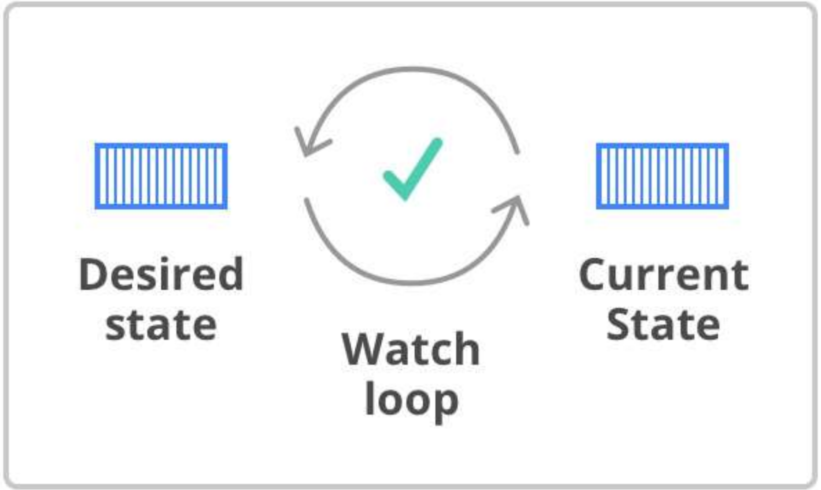


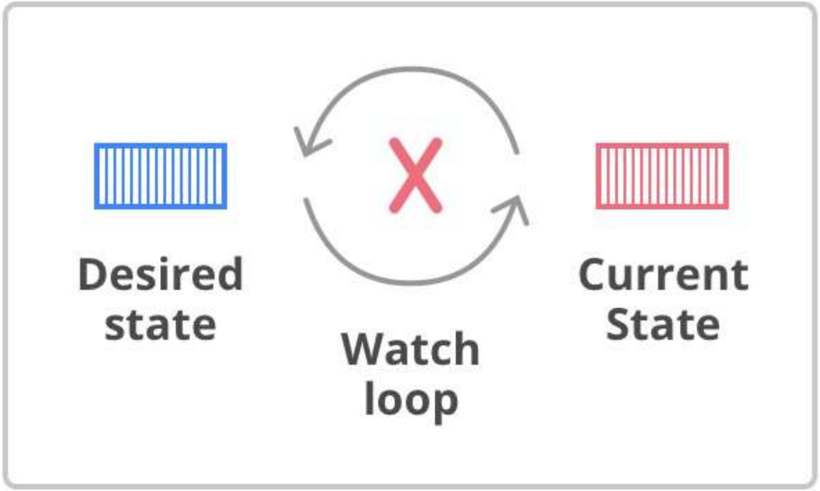
Control your rollouts with an Update Strategy appropriate for your application

Use Readiness Probes for your application

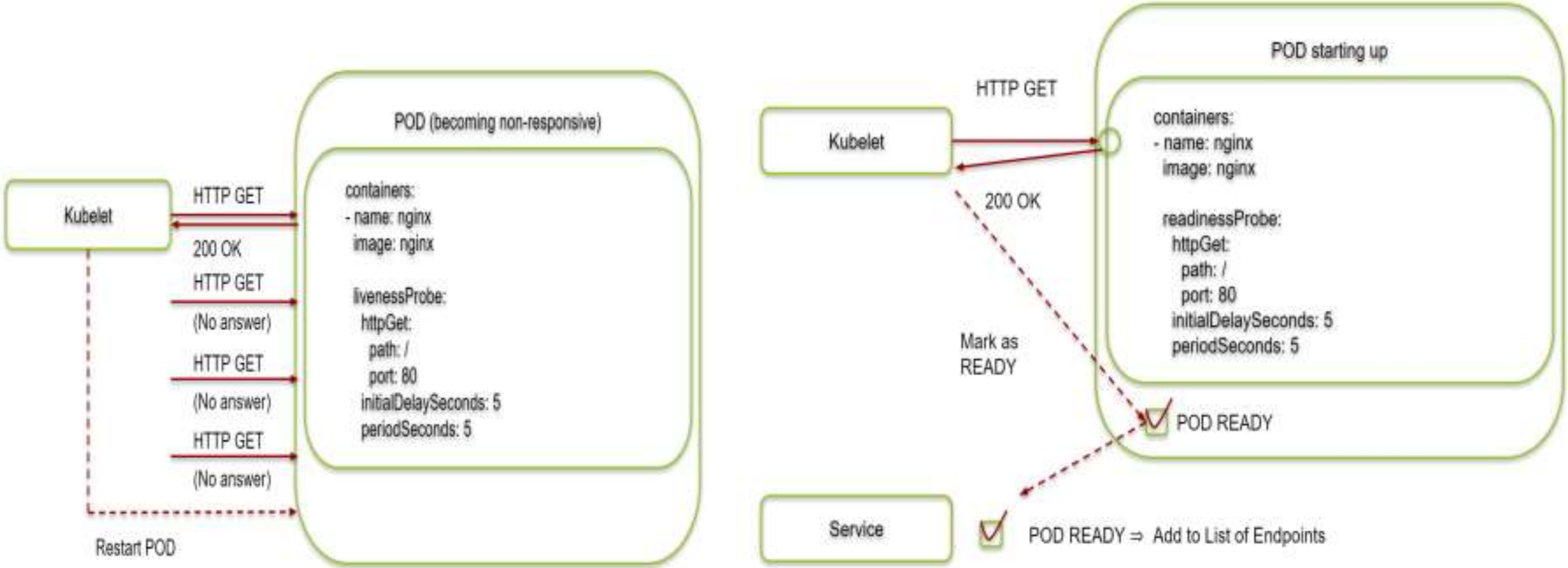
Use the `--record` option to leave a trail of your work for others

Self-Healing concept





Liveness Probe and Readiness Probe



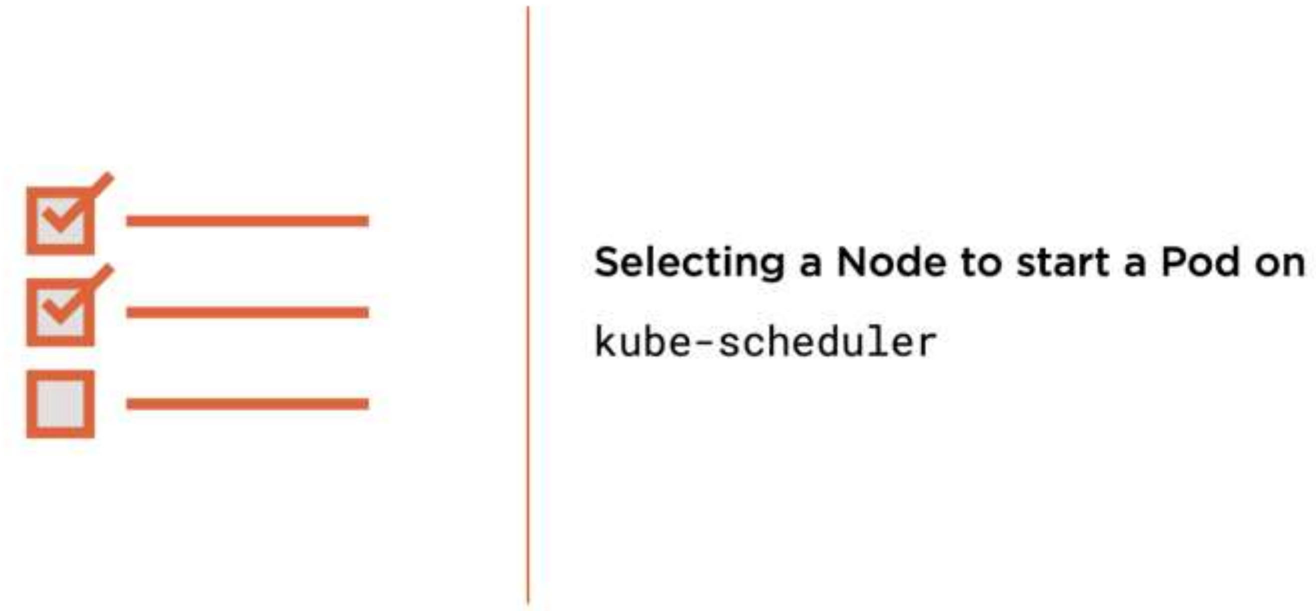
```
containers:
- name: nginx
  image: nginx

  livenessProbe:
   httpGet:
    path: /
    port: 80
   initialDelaySeconds: 5
   periodSeconds: 5
```

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

  readinessProbe:
   httpGet:
    path: /
    port: 80
   initialDelaySeconds: 5
   periodSeconds: 5
```

Scheduling in Kubernetes



Selecting a Node to start a Pod on

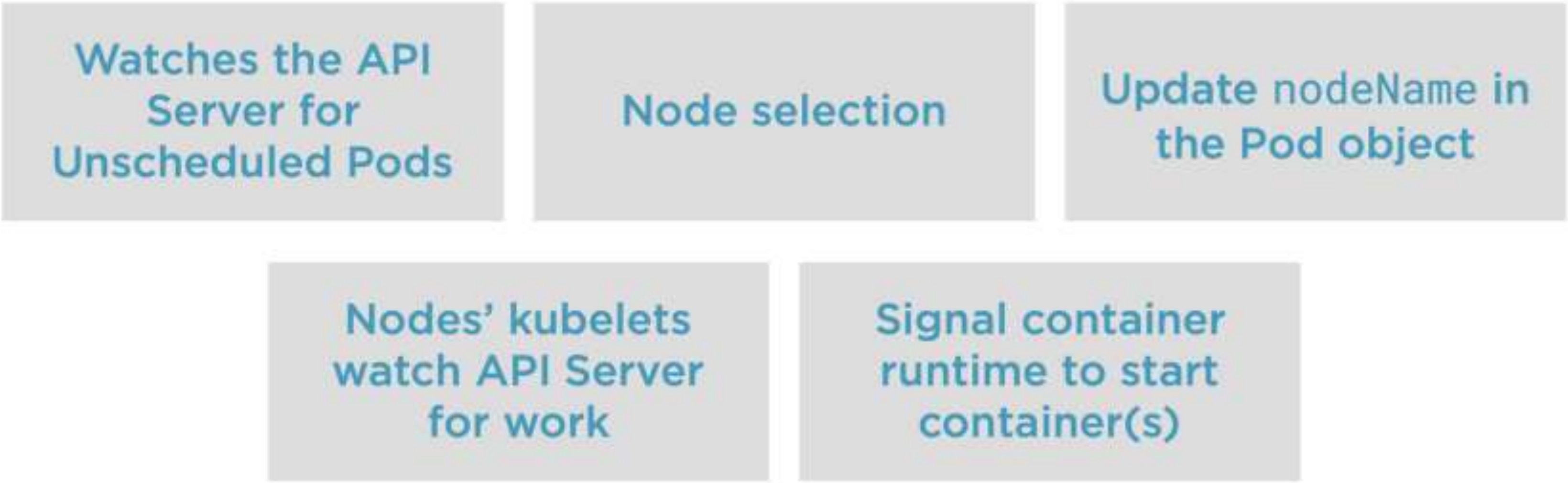
`kube-scheduler`

Scheduling in Kubernetes

Resources

Policy

Scheduling Process



Watches the API Server for Unscheduled Pods

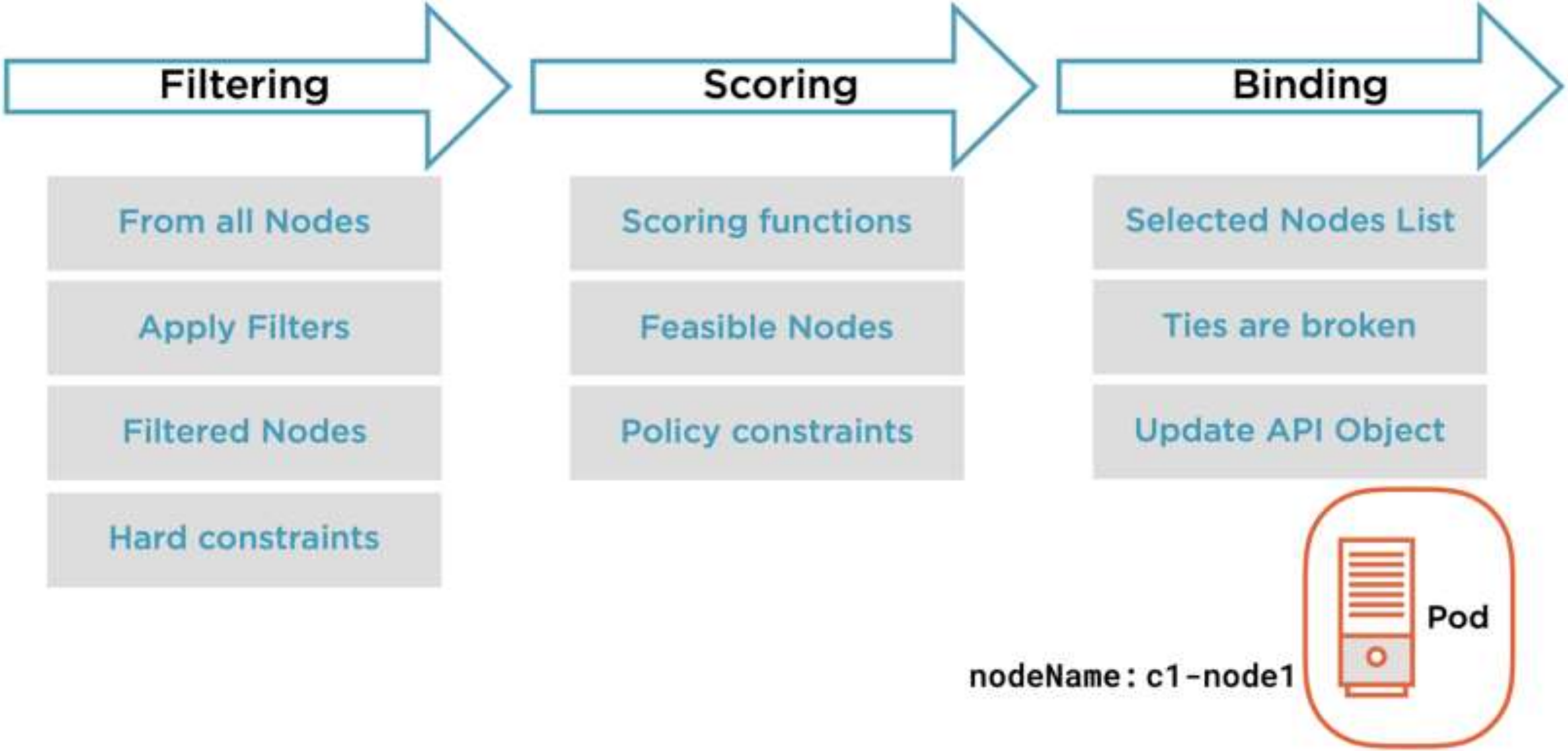
Node selection

Update nodeName in the Pod object

Nodes' kubelets watch API Server for work

Signal container runtime to start container(s)

Node Selection



Filtering

From all Nodes

Apply Filters

Filtered Nodes

Hard constraints

Scoring

Scoring functions

Feasible Nodes

Policy constraints

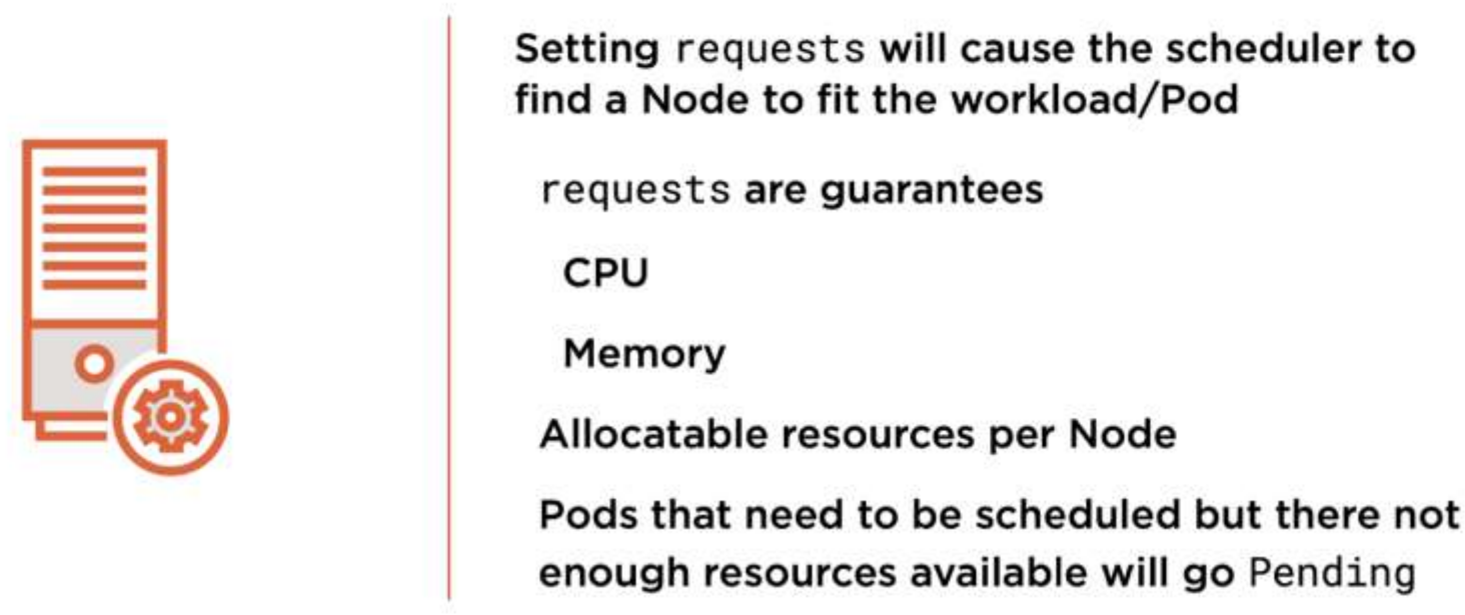
Binding

Selected Nodes List

Ties are broken

Update API Object

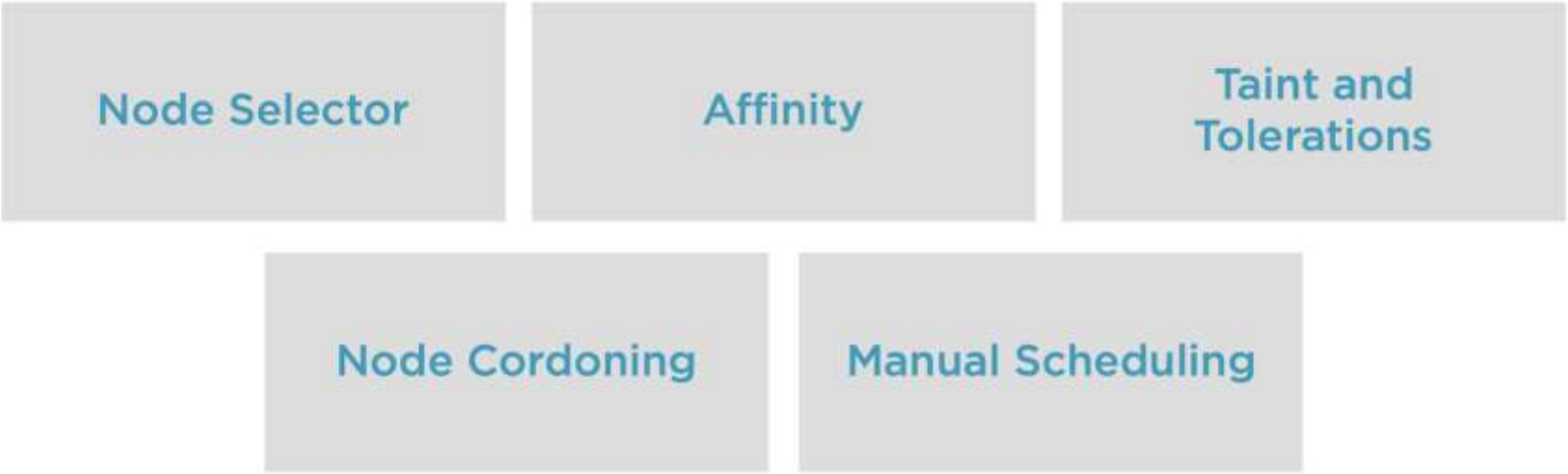
Resource Requests



Setting `requests` will cause the scheduler to find a Node to fit the workload/Pod

- `requests` are guarantees
  - CPU
  - Memory
- Allocatable resources per Node
- Pods that need to be scheduled but there not enough resources available will go `Pending`

Controlling Scheduling



Node Selector

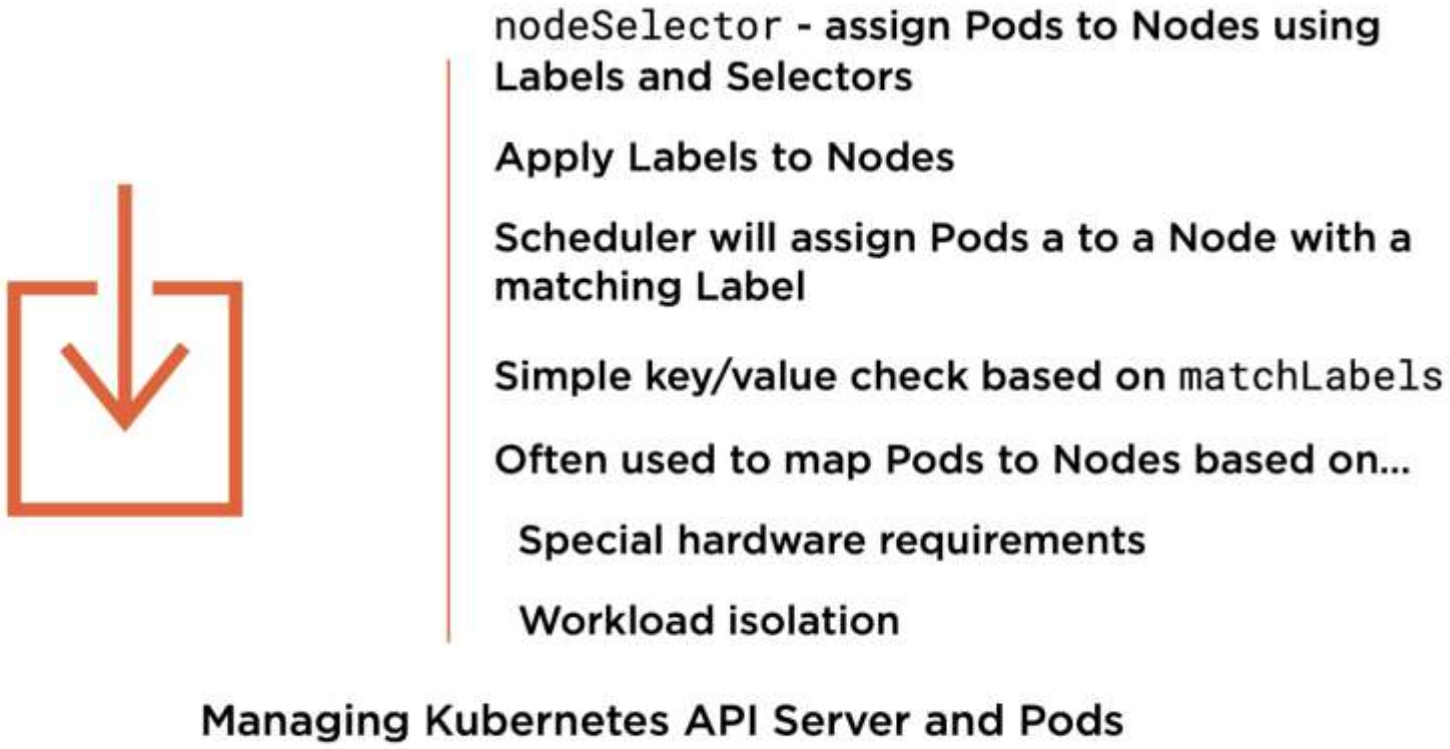
Affinity

Taint and Tolerations

Node Cordoning

Manual Scheduling

Node Selector



`nodeSelector` - assign Pods to Nodes using Labels and Selectors

Apply Labels to Nodes

Scheduler will assign Pods a to a Node with a matching Label

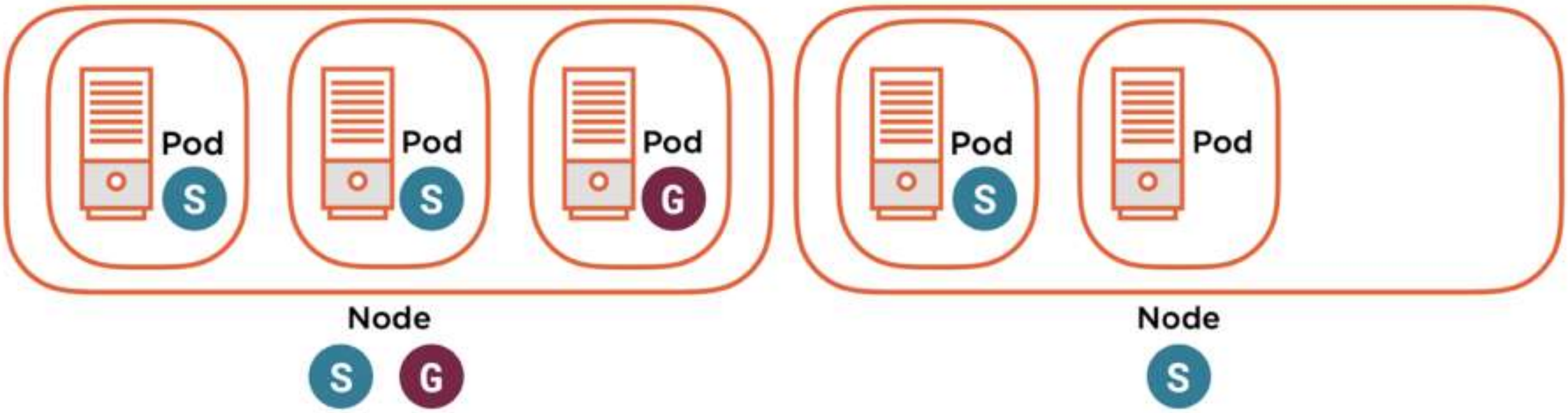
Simple key/value check based on `matchLabels`

Often used to map Pods to Nodes based on...

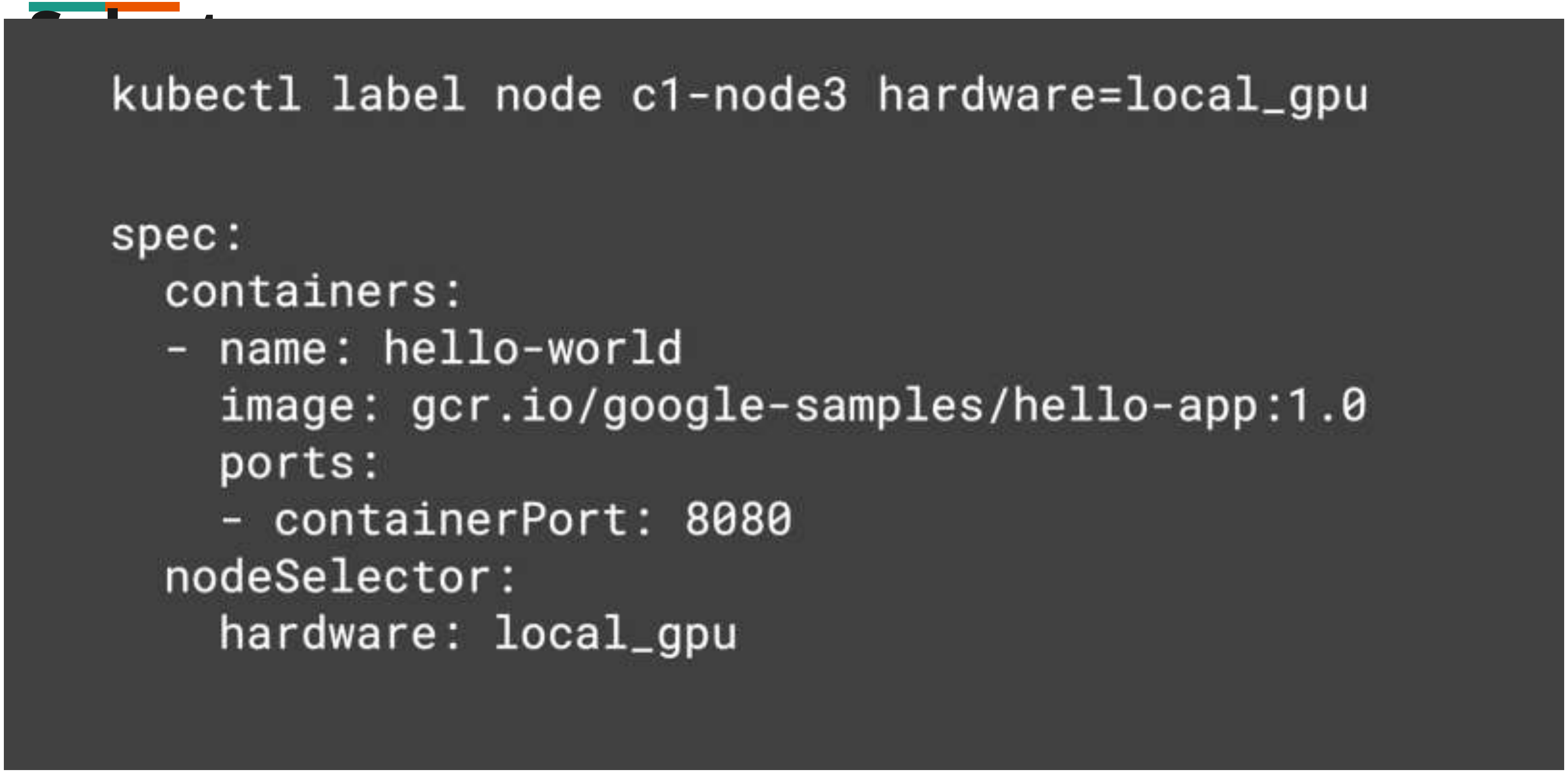
- Special hardware requirements
- Workload isolation

Managing Kubernetes API Server and Pods

Scheduling - Node Selector



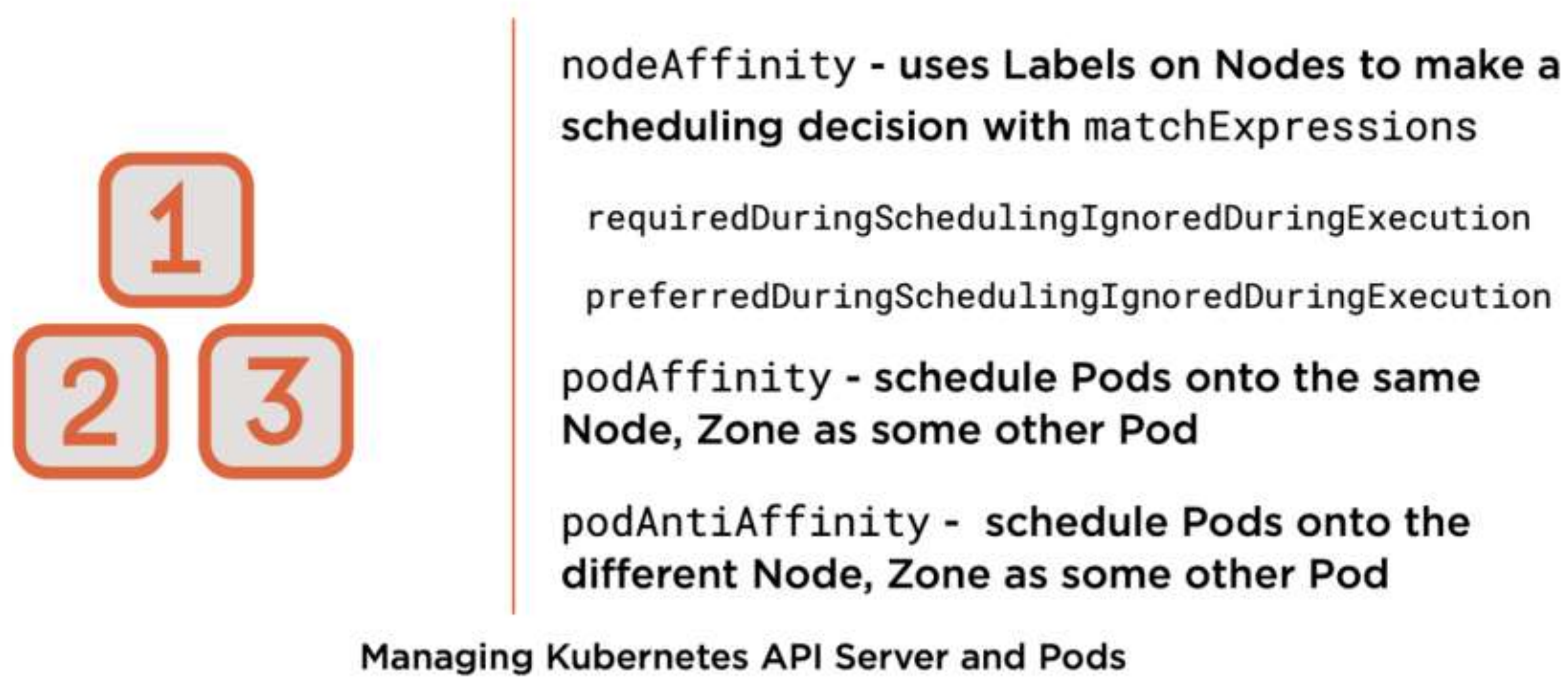
Assigning Pods to Nodes using Node



```
kubectl label node c1-node3 hardware=local_gpu


spec:
  containers:
  - name: hello-world
    image: gcr.io/google-samples/hello-app:1.0
    ports:
    - containerPort: 8080
  nodeSelector:
    hardware: local_gpu
```

Affinity and Anti-Affinity



`nodeAffinity` - **uses Labels on Nodes to make a scheduling decision with** `matchExpressions`

`requiredDuringSchedulingIgnoredDuringExecution`

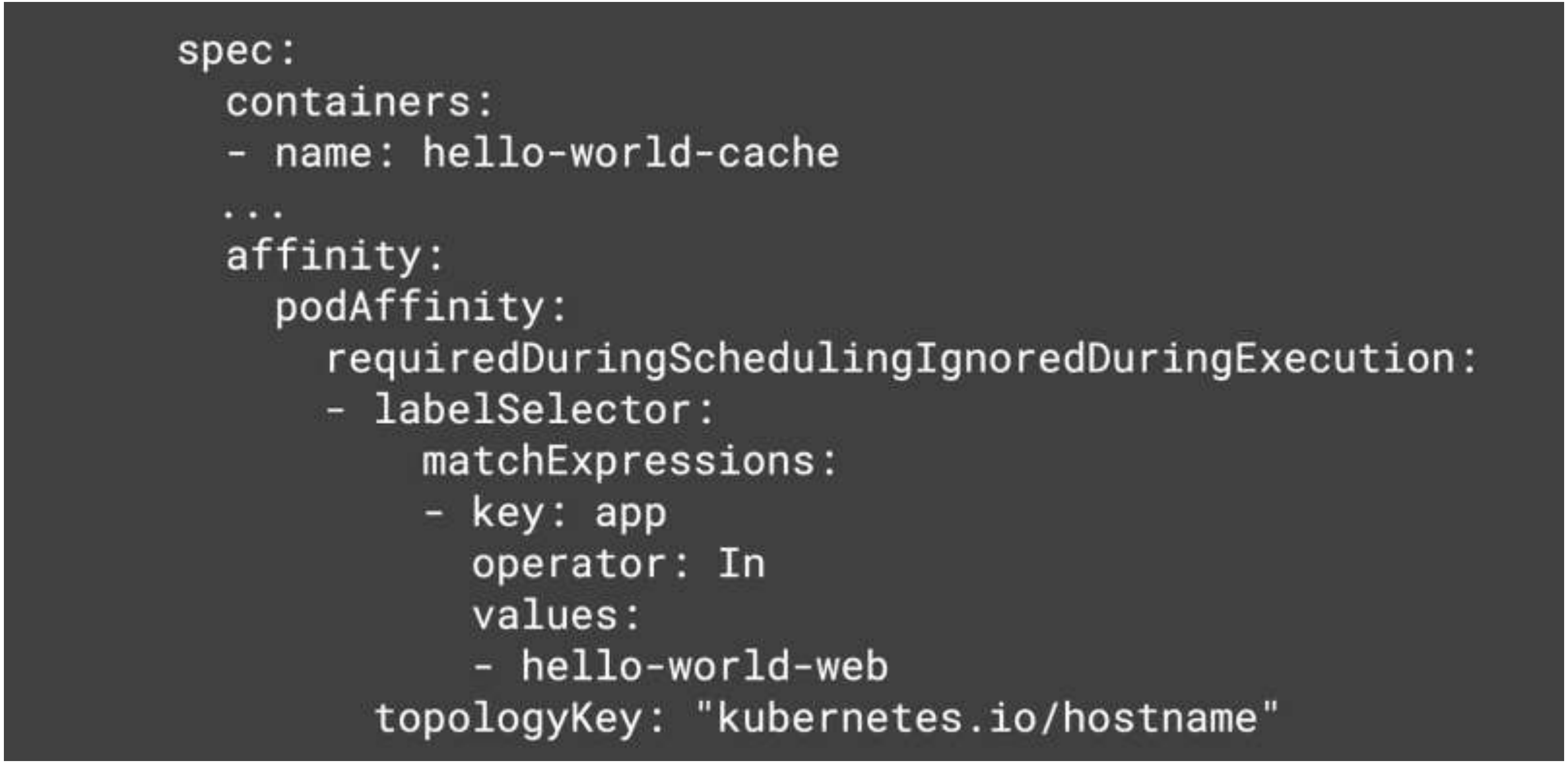
`preferredDuringSchedulingIgnoredDuringExecution`

`podAffinity` - **schedule Pods onto the same Node, Zone as some other Pod**

`podAntiAffinity` - **schedule Pods onto the different Node, Zone as some other Pod**

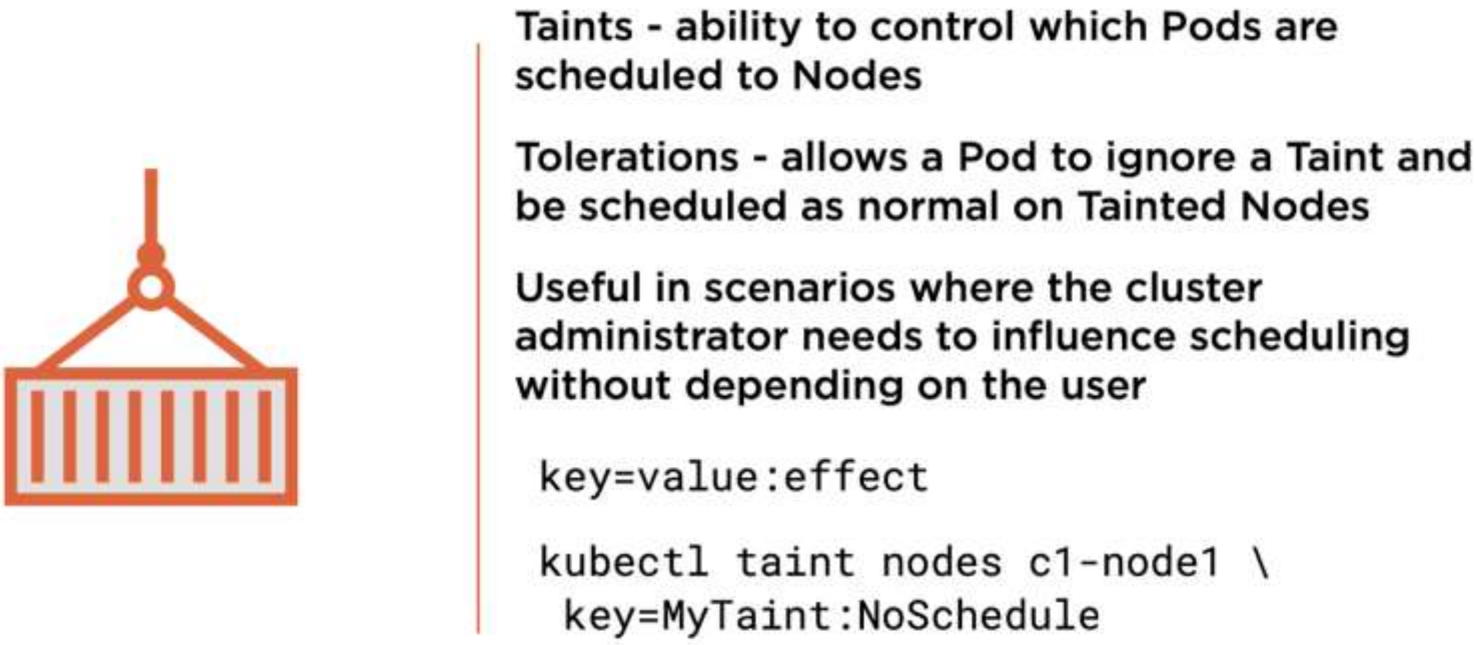
Managing Kubernetes API Server and Pods

Using Affinity to Control Pod Placement



```
spec:
  containers:
  - name: hello-world-cache
  ...
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - hello-world-web
        topologyKey: "kubernetes.io/hostname"
```

Taints and Tolerations



Taints - ability to control which Pods are scheduled to Nodes

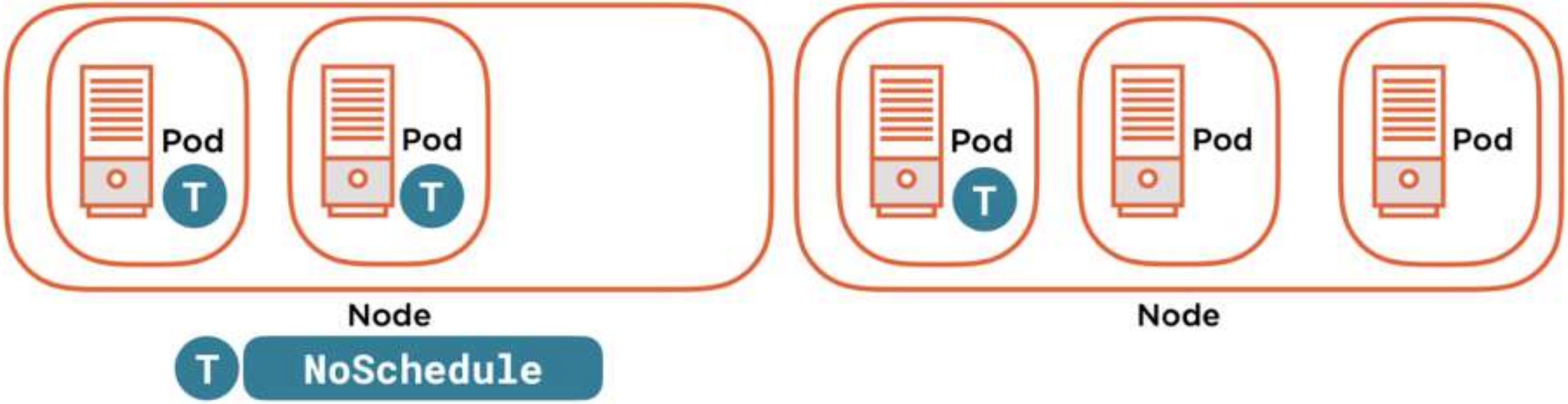
Tolerations - allows a Pod to ignore a Taint and be scheduled as normal on Tainted Nodes

Useful in scenarios where the cluster administrator needs to influence scheduling without depending on the user

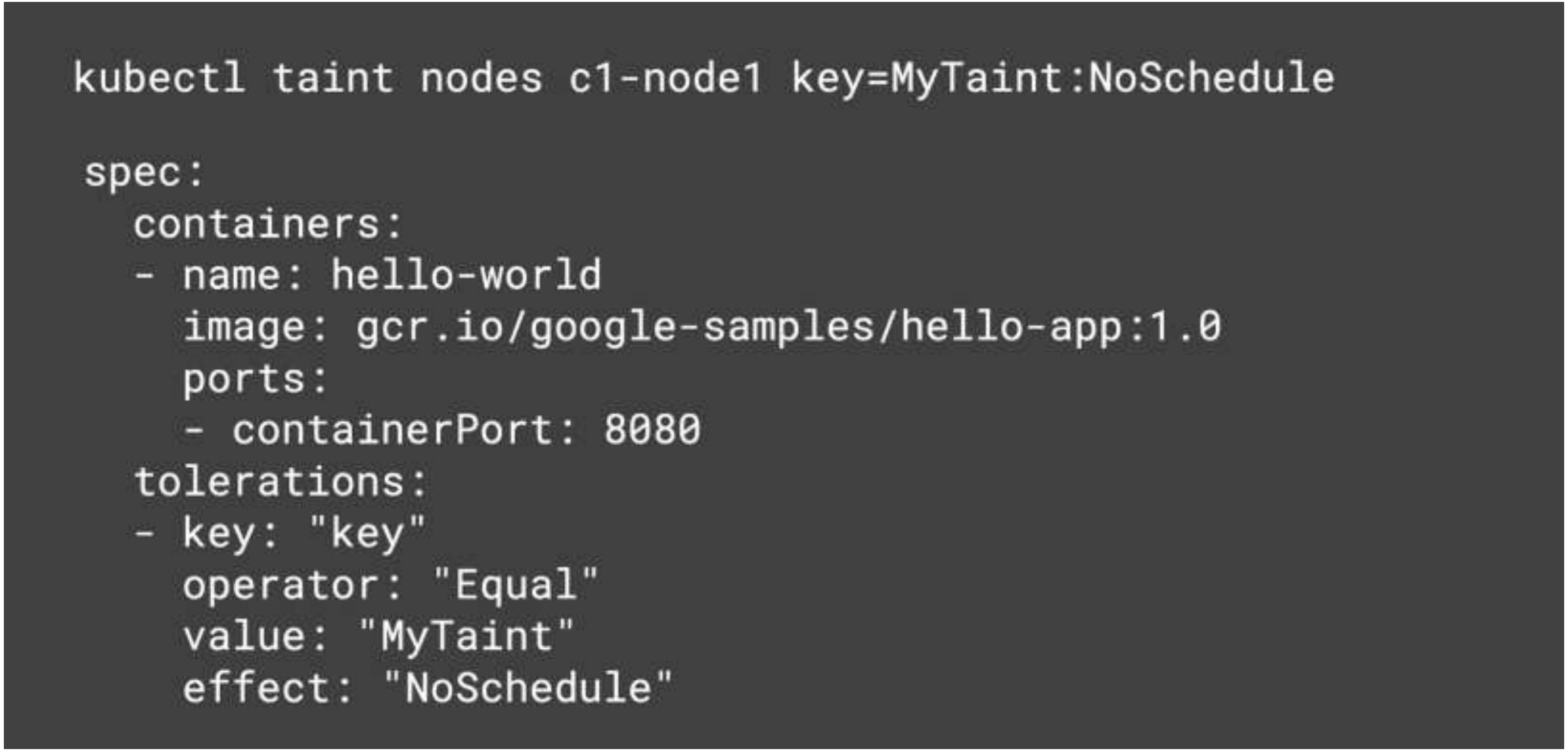
```
key=value:effect
```

```
kubectl taint nodes c1-node1 \
 key=MyTaint:NoSchedule
```

Scheduling - Taints and Tolerations



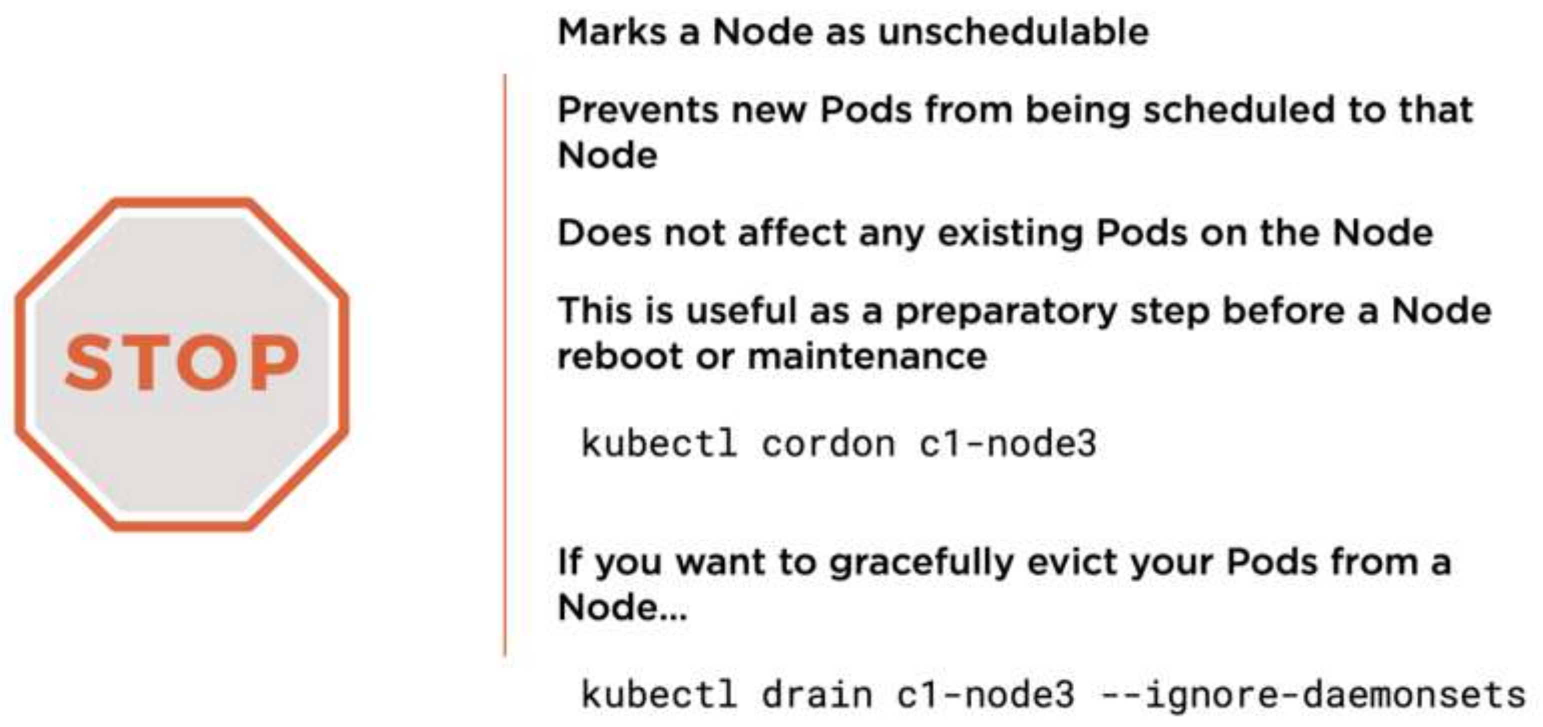
Adding a Taint to a Nodes and a Toleration to a Pod



```
kubectl taint nodes c1-node1 key=MyTaint:NoSchedule

spec:
  containers:
  - name: hello-world
    image: gcr.io/google-samples/hello-app:1.0
    ports:
    - containerPort: 8080
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "MyTaint"
    effect: "NoSchedule"
```

Node Cordoning



Marks a Node as unschedulable

Prevents new Pods from being scheduled to that Node

Does not affect any existing Pods on the Node

This is useful as a preparatory step before a Node reboot or maintenance

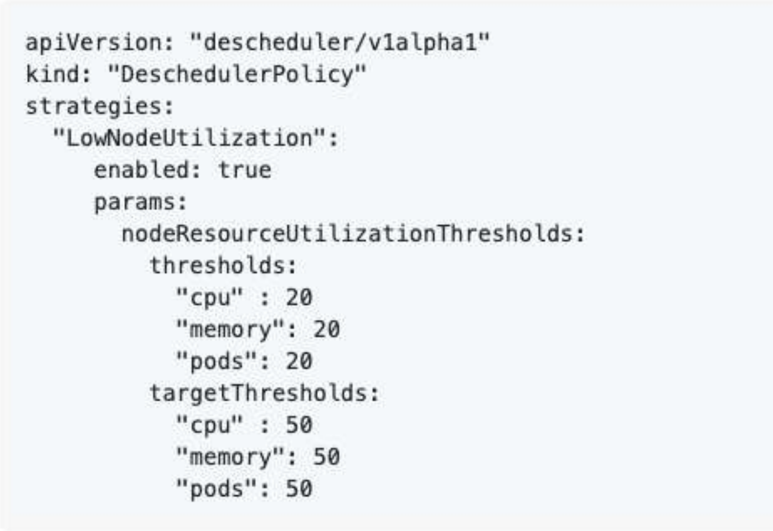
```
kubectl cordon c1-node3
```

If you want to gracefully evict your Pods from a Node...

```
kubectl drain c1-node3 --ignore-daemonsets
```

DeScheduler

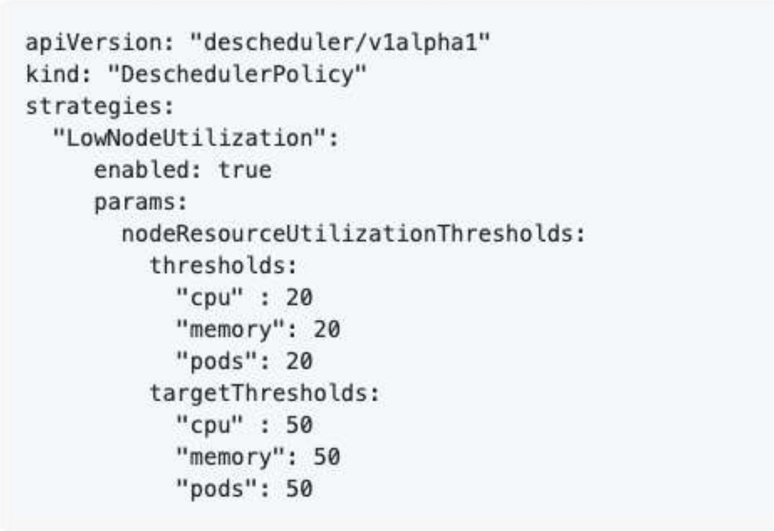
- Some nodes are under or over utilized.
- The original scheduling decision does not hold true any more, as taints or labels are added to or removed from nodes, pod/node affinity requirements are not satisfied any more.
- Some nodes failed and their pods moved to other nodes.
- New nodes are added to clusters.



```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "LowNodeUtilization":
     enabled: true
     params:
       nodeResourceUtilizationThresholds:
         thresholds:
           "cpu" : 20
           "memory": 20
           "pods": 20
         targetThresholds:
           "cpu" : 50
           "memory": 50
           "pods": 50
```

DeScheduler

- Some nodes are under or over utilized.
- The original scheduling decision does not hold true any more, as taints or labels are added to or removed from nodes, pod/node affinity requirements are not satisfied any more.
- Some nodes failed and their pods moved to other nodes.
- New nodes are added to clusters.



```
apiVersion: "descheduler/v1alpha1"
kind: "DeschedulerPolicy"
strategies:
  "LowNodeUtilization":
     enabled: true
     params:
       nodeResourceUtilizationThresholds:
         thresholds:
           "cpu" : 20
           "memory": 20
           "pods": 20
         targetThresholds:
           "cpu" : 50
           "memory": 50
           "pods": 50
```

Q
A