Kubernetes Housekeeping

How K8s manages resources and keeps its house clean!

Hello! I am Damini Satya

Software Engineer at Salesforce
Open Source contributor - @kubernetes

Twitter - @Daminisatya

salesforce

Hello! I am Mitesh Jain

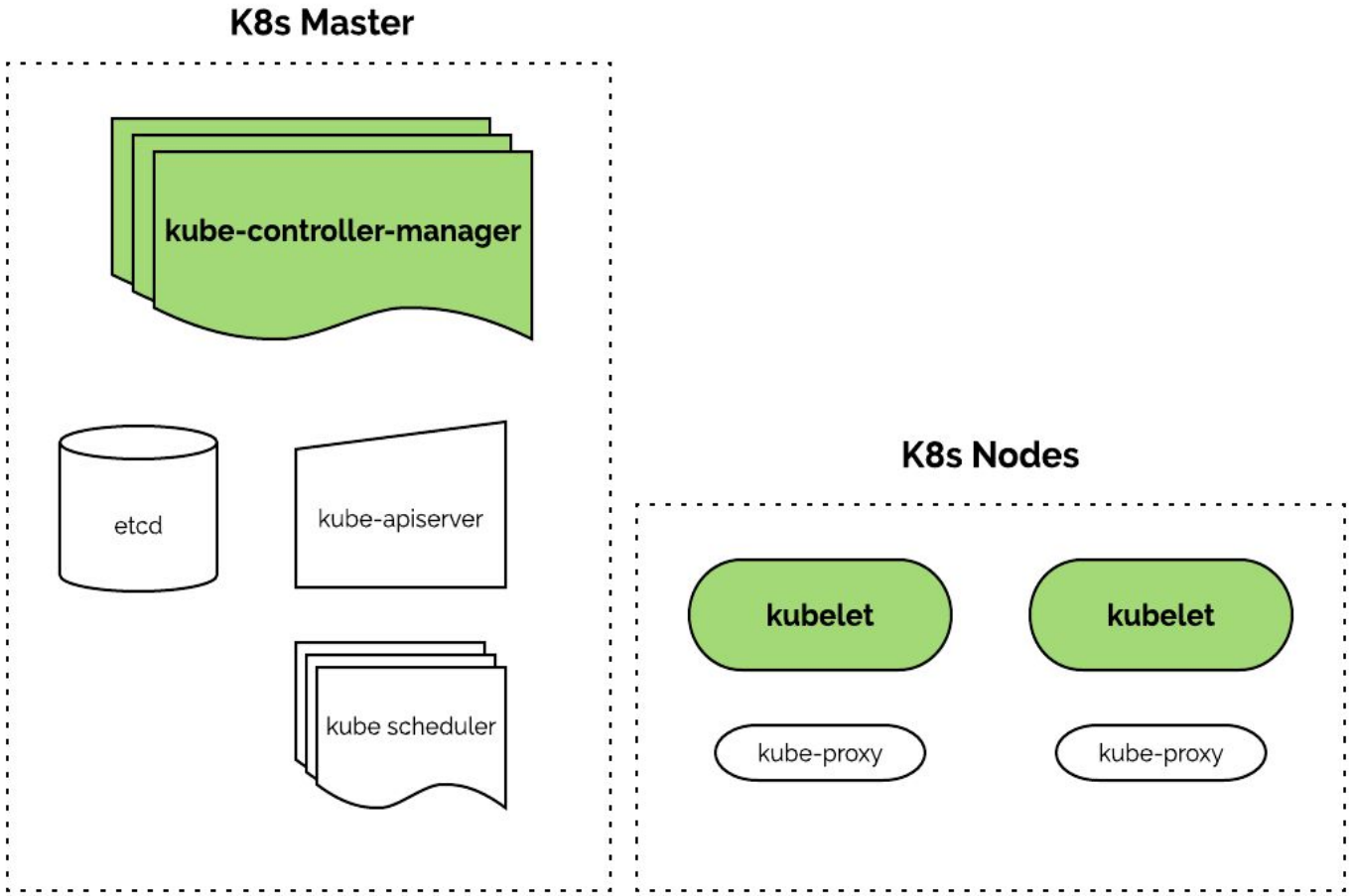
Lead Systems Engineer
Open Source deployments in public and private clouds
Ex - RedHat, GE, Wipro and Salesforce.com

Twitter - @MiteshSKJ

Agenda

- Introduction
- Garbage Collector Controller
- Kubelet
- Eviction Manager
- Node conditions
- Reclaim
- Best practices

Garbage Collection in K8s


K8s Master
kube-controller-manager
etcd
kube-apiserver
kube scheduler
K8s Nodes
kubelet
kubelet
kube-proxy
kube-proxy

Garbage Collection in K8s Master

kube-controller-manager

- Component on the master that runs controllers.

```
kube-controller-manager [flags]
```

- [flags]
  - --enable-garbage-collector
  - --terminated-pod-gc-threshold
  - --concurrent-gc-syncs

Garbage Collector Controller

- Graph builder
- attemptToDelete & attemptToOrphan queues

Garbage Collector Controller Configuration

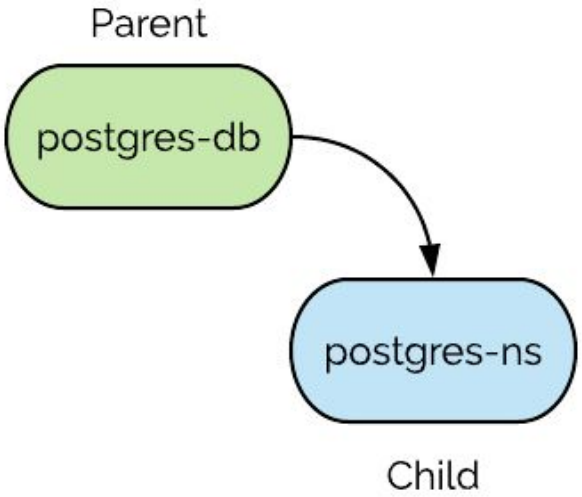
- *enable-garbage-collector - Default: true*
- *concurrent-gc-syncs - Default: 20*
- *terminated-pod-gc-threshold - Default: 12500*

```
kube-controller-manager [flags]
```

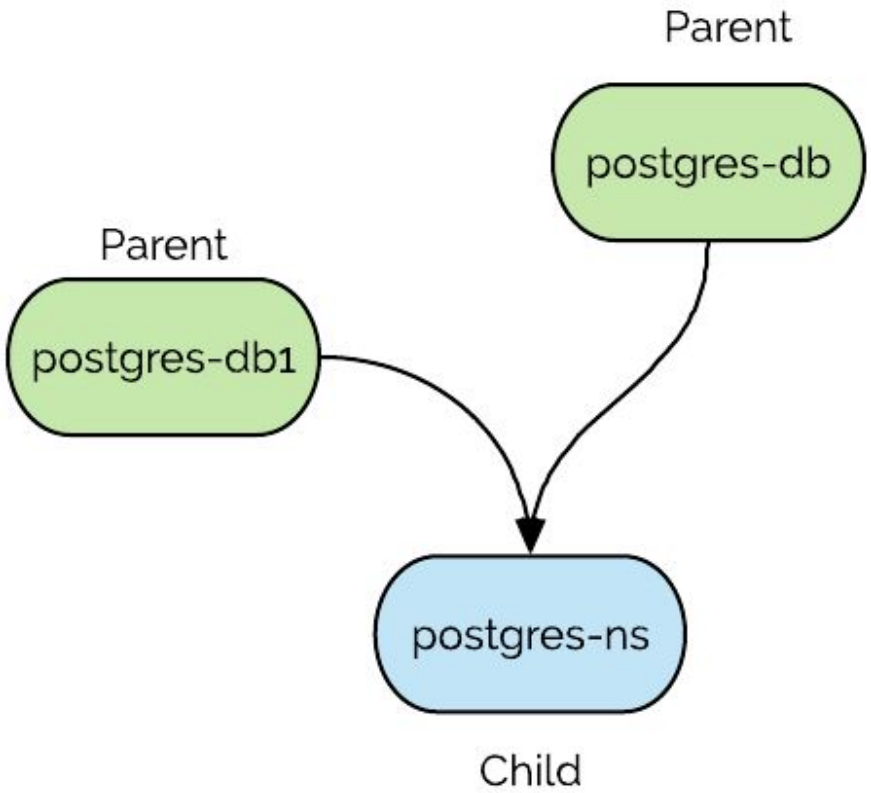
Owners and Dependents

- Owners (Parent)
- Dependents (Child)
- metadata.ownerReferences
- Multiple ownerReferences

```yaml
apiVersion: v1
kind: Namespace
metadata:
   name: postgres-ns
   ownerReferences:
    - apiVersion: v1
      controller: true
      blockOwnerDeletion: true
      kind: Database
      name: postgres-db
      uid: 1552c2c2-8ea1-11e8-8289-02ee24bb8af6
```


Parent
postgres-db
postgres-ns
Child

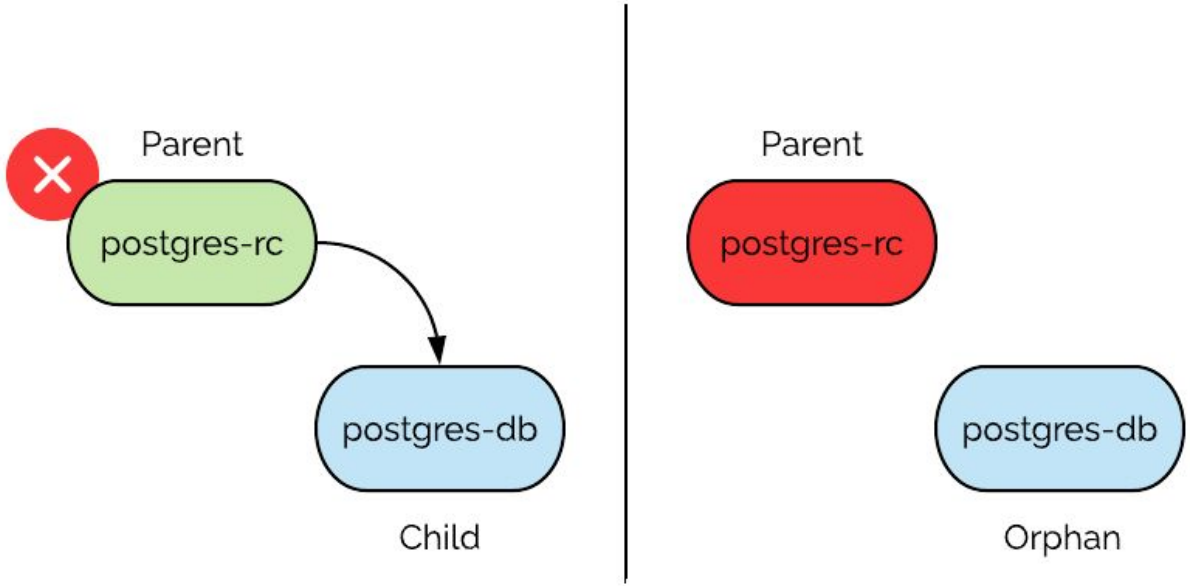
```yaml
apiVersion: v1
kind: Namespace
metadata:
   name: postgres-ns
   ownerReferences:
   - apiVersion: v1
     controller: true
     blockOwnerDeletion: true
     kind: Database
     name: postgres-db
     uid: 0eccecf0-8ea4-11e8-8289-02ee24bb8af6
   - apiVersion: v1
     controller: false
     blockOwnerDeletion: true
     kind: Database
     name: postgres-db1
     uid: 11e140d9-8ea4-11e8-8289-02ee24bb8af6
```


Parent
postgres-db
Parent
postgres-db1
postgres-ns
Child

Delete dependents

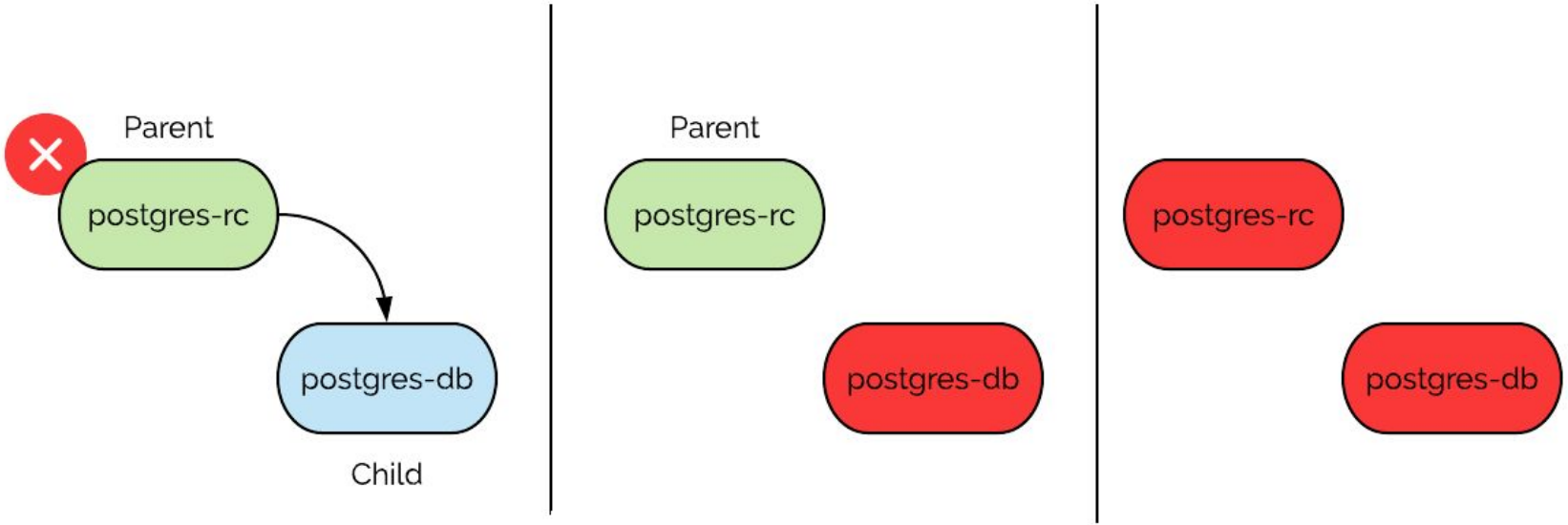
- Orphan
- Foreground cascading deletion
- Background cascading deletion

Orphan



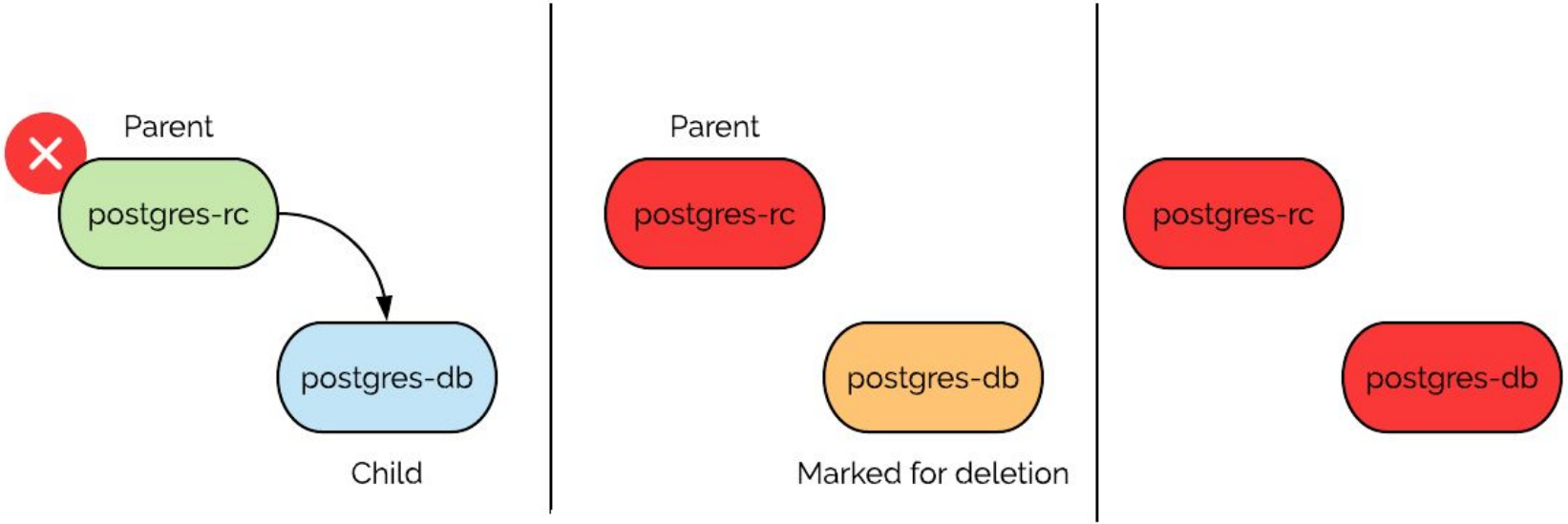

```
curl -X DELETE localhost:8080/apis/apps/v1/namespaces/default/replicasets/postgres-rc \
  -d '{"kind":"DeleteOptions","apiVersion":"v1","propagationPolicy":"Orphan"}' \
  -H "Content-Type: application/json"
```

Foreground cascading deletion




```
curl -X DELETE localhost:8080/apis/apps/v1/namespaces/default/replicasets/postgres-rc \
  -d '{"kind":"DeleteOptions","apiVersion":"v1","propagationPolicy":"Foreground"}' \
  -H "Content-Type: application/json"
```

Background cascading deletion




```
curl -X DELETE localhost:8080/apis/apps/v1/namespaces/default/replicasets/postgres-rc \
  -d '{"kind":"DeleteOptions","apiVersion":"v1","propagationPolicy":"Background"}' \
  -H "Content-Type: application/json"
```

Garbage Collection in K8s Node

Kubelet

An agent that runs on each node in the cluster. Apart from making sure that containers are running in a Pod it also performs the task of monitoring and managing resource on the nodes.

Kubelet Configuration

- CLI options (KUBELET_EXTRA_ARGS)
- Config file (/var/lib/kubelet/config)
- Dynamic Configuration

Resource Handling

- ***Trigger configuration*** - Defines threshold for resources which will trigger garbage collection or eviction.

- ***Policy configuration*** - Defines parameters used to govern when and how a resource is evicted/managed.

Image Garbage Collector (to be deprecated)

- Image Garbage Collector deletes the unused images in the nodes.
- Set for deprecation in favour of Eviction Manager
- Flags
  - *imageGCHighThresholdPercent: 85*
  - *imageGCLowThresholdPercent: 80*

Image Garbage Collector additional flags

image-gc-high-threshold	In favour of ***eviction-hard*** or ***eviction-soft*** which can trigger image garbage collection.
image-gc-low-threshold	In favour of ***eviction-minimum-reclaim*** which achieves the same behavior.
low-diskspace-threshold-mb	In favour of ***eviction-hard*** or ***eviction-soft*** as eviction generalizes disk thresholds to other resources.
outofdisk-transition-frequency	In favour of ***eviction-pressure-transition-period*** as eviction generalizes disk pressure transition to other resources.
maximum-dead-containers	deprecated once old logs are stored outside of container's context
maximum-dead-containers-per-container	deprecated once old logs are stored outside of container's context
minimum-container-ttl-duration	deprecated once old logs are stored outside of container's context

Eviction Manager

Eviction Manager is a sub process of kubelet which monitors the system resources and takes action based on the configured thresholds and policies.

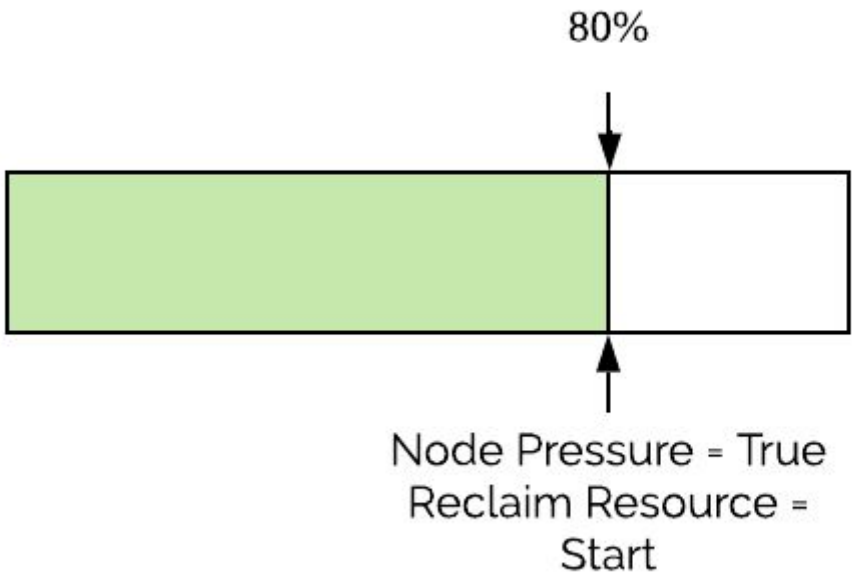
Eviction Monitoring Interval

- *housekeeping-interval*

Eviction Manager Policies

- Hard Eviction Thresholds
- Soft Eviction Thresholds
- Oscillation of node conditions
- Minimum eviction reclaim

Eviction Manager - Hard Eviction Thresholds

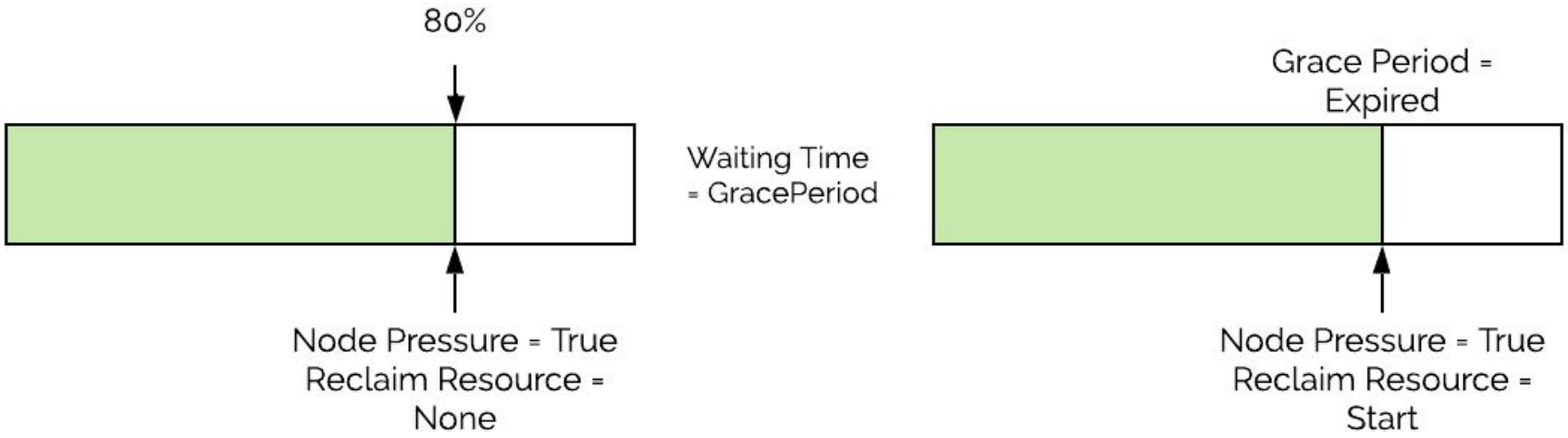


Eviction Manager - Hard Eviction Thresholds

- Flags
  - ***eviction-hard*** - describes a set of eviction thresholds.

- Example
  - *--eviction-hard="imagefs.available<15%,memory.available<600Mi"*

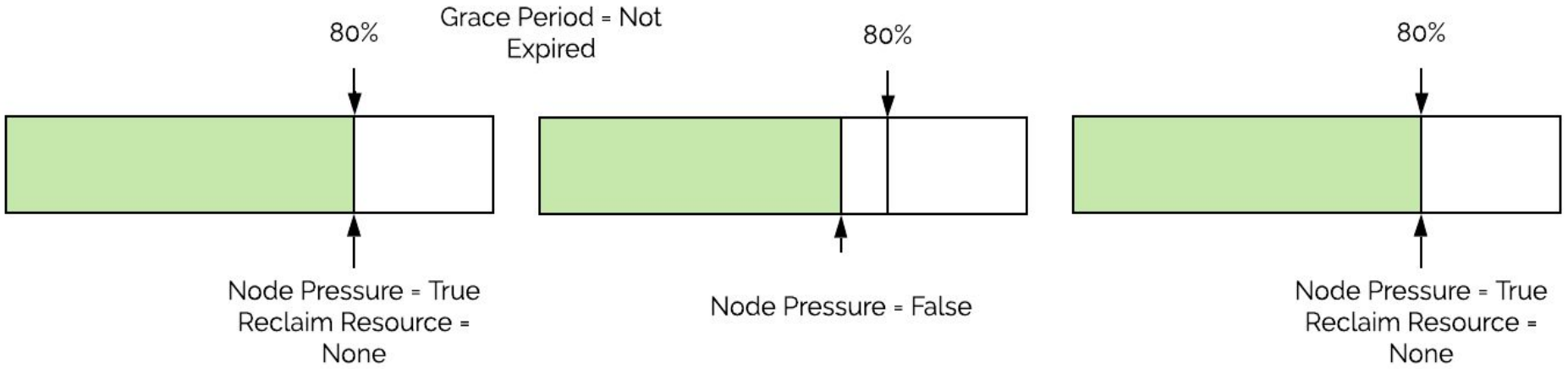
Eviction Manager - Soft Eviction Thresholds



Eviction Manager - Soft Eviction Thresholds

- Pairs an eviction threshold with a required grace period.
- Flags
  - ***eviction-soft*** - describes a set of eviction thresholds
  - ***eviction-soft-grace-period*** - describes a set of eviction grace periods
  - ***eviction-max-pod-grace-period*** - describes the maximum allowed grace period (in seconds) to use when terminating Pods in response to a soft eviction threshold being met.
- Example
  - *--eviction-soft="memory.available<600Mi"*
    --eviction-soft-grace-period="memory.available=1m30s"

Eviction Manager - Oscillation of Node Conditions

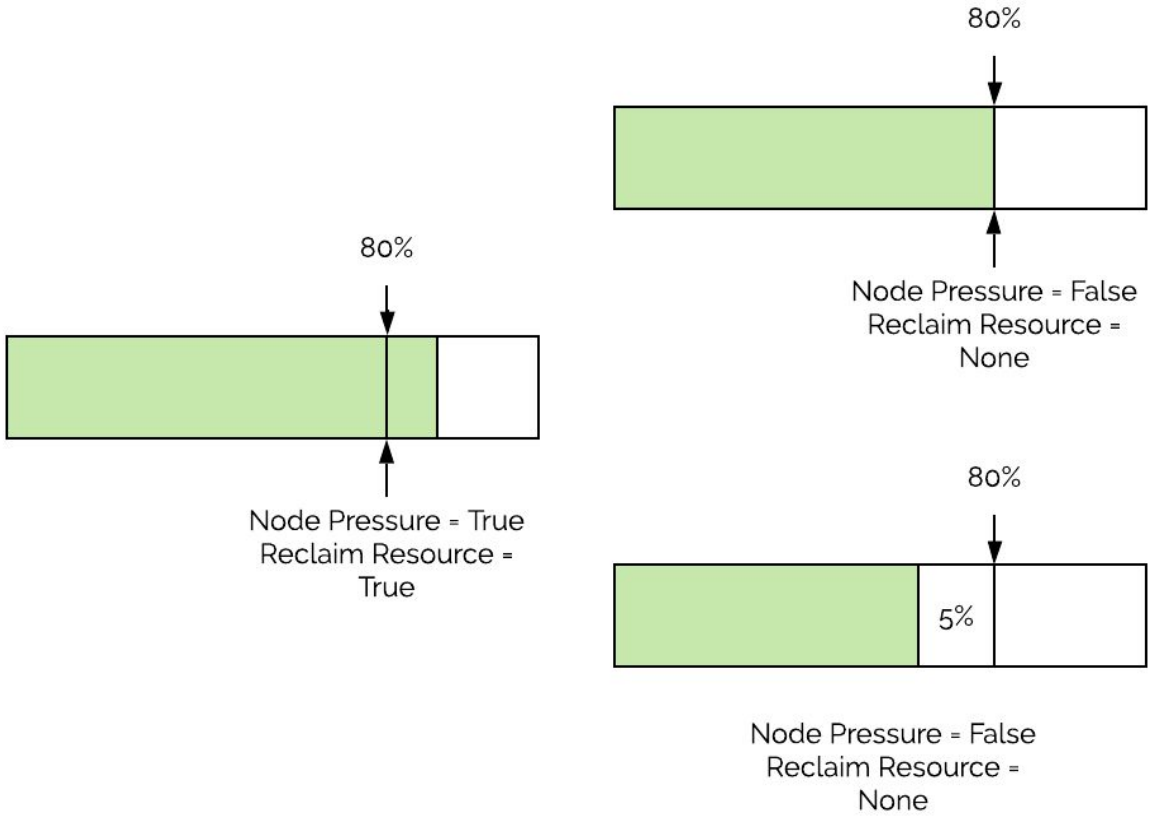


Eviction Manager - Oscillation of Node Conditions

- If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, it would cause the corresponding node condition to constantly oscillate between true and false, and could cause poor scheduling decisions as a consequence.

- Flag
  - ***eviction-pressure-transition-period*** - duration for which the kubelet has to wait before transitioning out of an eviction pressure condition.

Eviction Manager - Minimum Eviction Reclaim



Eviction Manager - Minimum Eviction Reclaim

- In certain scenarios, eviction of Pods could result in reclamation of small amount of resources. Such scenarios, can result in kubelet hitting eviction thresholds in repeated successions. As eviction of resources like disk, is time consuming, this will keep the node in constant pressure.
- Flags
  - ***eviction-minimum-reclaim*** - describes minimum reclaim amount for a set of eviction thresholds
- Example
  - --eviction-hard=”imagefs.available<10Gi,memory.available<600Mi” --eviction-minimum-reclaim=”imagefs.available=2Gi,memory.available=0”

- The default *eviction-minimum-reclaim* is 0 for all resources

Node Conditions

- kubelet reports node status updates to the Master at regular interval.
- Configurable ***--node-status-update-frequency*** (defaults to 10s)
- kubectl get nodes <NODE_NAME> -o json ***OR***
  kubectl describe node <NODE_NAME>
- Conditions
  - Disk Pressure
  - Memory Pressure
  - Pid Pressure

Disk Pressure

- kubelet supports only two filesystem partitions.
  - **nodefs** filesystem
  - **imagefs** filesystem

.

- Eviction Signals
  - ***nodefs.available, nodefs.inodesFree, imagefs.available, or imagefs.inodesFree***
  - Default:
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%

- Examples
  - --eviction-hard=”imagefs.available<10Gi”
  - --eviction-hard=”imagefs.available<20%”

Disk Pressure Condition


```
{
  "lastHeartbeatTime": "2019-05-04T05:14:59Z",
  "lastTransitionTime": "2019-05-04T05:01:38Z",
  "message": "kubelet has disk pressure",
  "reason": "KubeletHasDiskPressure",
  "status": "True",
  "type": "DiskPressure"
},
```

Disk Pressure - kubelet log

```
eviction_manager.go:333] eviction manager: attempting to reclaim ephemeral-storage
eviction_manager.go:344] eviction manager: must evict pod(s) to reclaim ephemeral-storage
eviction_manager.go:362] eviction manager: pods ranked for eviction:
```

```
eviction_manager.go:333] eviction manager: attempting to reclaim ephemeral-storage
eviction_manager.go:344] eviction manager: must evict pod(s) to reclaim ephemeral-storage
eviction_manager.go:355] eviction manager: eviction thresholds have been met, but no pods are
active to evict
```

Memory Pressure

- Available memory on the node has satisfied an eviction threshold.

- Eviction signals
  - ***memory.available*** - Uses /proc/meminfo and cgroup info to compute available memory
  - Default:
    - memory.available<100Mi

- Example
  - --eviction-hard=”memory.available<600Mi”

Memory Pressure Condition


```
{
  "lastHeartbeatTime": "2019-05-04T05:14:59Z",
  "lastTransitionTime": "2019-05-04T05:01:38Z",
  "message": "kubelet has insufficient memory available",
  "reason": "KubeletHasInsufficientMemory",
  "status": "True",
  "type": "MemoryPressure"
},
```

Memory Pressure - kubelet log

```
eviction_manager.go:333] eviction manager: attempting to reclaim memory
eviction_manager.go:344] eviction manager: must evict pod(s) to reclaim memory
eviction_manager.go:362] eviction manager: pods ranked for eviction:
```

```
eviction_manager.go:333] eviction manager: attempting to reclaim memory
eviction_manager.go:344] eviction manager: must evict pod(s) to reclaim memory
eviction_manager.go:355] eviction manager: eviction thresholds have been met, but no pods are
active to evict
```

Node OOM

- System oom_killer

- ***oom_score_adj values***
  - *Guaranteed* -998
  - *BestEffort* 1000
  - *Burstable* `min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 999)`

- Unlike Pod eviction, if a Pod container is OOM killed, it may be restarted by the kubelet based on its RestartPolicy

Reclaim

- Delete dead Pods and their containers
- Delete all unused (unreferenced) images
- Evict end-user Pods

Reclaim - Node level disk space

- With imagefs
  - If nodefs filesystem has met eviction thresholds, kubelet frees up disk space by deleting the dead Pods and their containers.
  - If imagefs filesystem has met eviction thresholds, kubelet frees up disk space by deleting all unused images.

- Without imagefs
  - If nodefs filesystem has met eviction thresholds, kubelet frees up disk space in the following order:
    - Delete dead Pods and their containers
    - Delete all unused images

Reclaim - Pod Ranking for Eviction

- Criteria for ranking of Pods for eviction
  - Whether or not a Pod's usage of the starved compute resource exceeds requests.
  - Pod's Priority
  - Consumption of the starved compute resource relative to scheduling requests.

- Ranking
  - BestEffort or Burstable Pods
    - If a Pod's usage of a starved resource exceeds its request. They are ranked by Priority, and then by usage above request.
  - Guaranteed and Burstable Pods
    - If a Pod's usage is beneath requests they are evicted last.
    - If at all, evict Pods of Lowest Priority first.

(Cont.) Reclaim - Pod Ranking for Eviction

- ***Inode starvation*** - evict Pods with the lowest quality of service first.
- ***Disk space starvation*** - evict Pods with largest disk consumption amongst the lowest quality of service first.
  - With imagefs
    - If nodefs is triggering evictions, kubelet sorts Pods based on the usage on nodefs - local volumes + logs of all its containers.
    - If imagefs is triggering evictions, kubelet sorts Pods based on the writable layer usage of all its containers.
  - Without imagefs
    - If nodefs is triggering evictions, kubelet sorts Pods based on their total disk usage - local volumes + logs & writable layer of all its containers

Scheduler

- Memory Pressure
  - No new BestEffort Pods are scheduled to the node
- Disk Pressure
  - No new Pods are scheduled to the node

Best Practices

- Use config file and manage via configuration management system or use dynamic Configuration.
- Watchout and avoid parameters to be deprecated.
- Reserve resources for system (system-reserved & kube-reserved) and compute thresholds accordingly.
- Enforce judicious use of priority and QoS flags while deploying apps/pods as the ranking for eviction is computed based on them.

(Cont.) Best Practices

- If application requires time to drain while stopping, Use soft-eviction and thresholds to gracefully terminate the pods instead of being killed instantly.

- Use minimum Eviction and oscillation parameters to prevent aggressive and time consuming evictions/reclaim.

- Evictions may lead to an unbalanced cluster. Keep a watch on hotspots and underutilized nodes.

- Plan early!

Thank You!