KubeCon

North America 2017

Local Ephemeral Storage Resource Management

Jing Xu, Google

Agenda

- Motivation
- Resource management and model
- Storage Overview
- Local Ephemeral Storage Management
- Future Work

jinxu@google.com


jinxu@slack.kubernetes.com

jingxu97@github.com

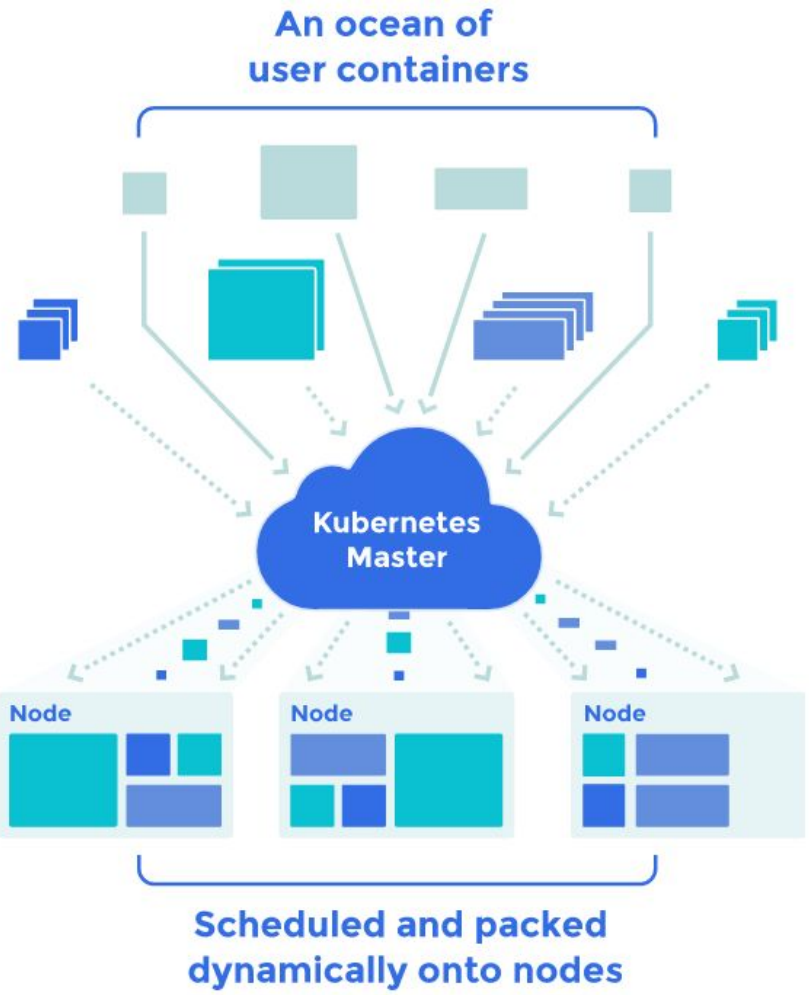
Motivation

You might wonder

- why my container got killed?
- why my service is running slow?
- why machine keeps crashing?
- ...

Resources are shared

- one container used up all cpu/memory
- one container produced lots of data

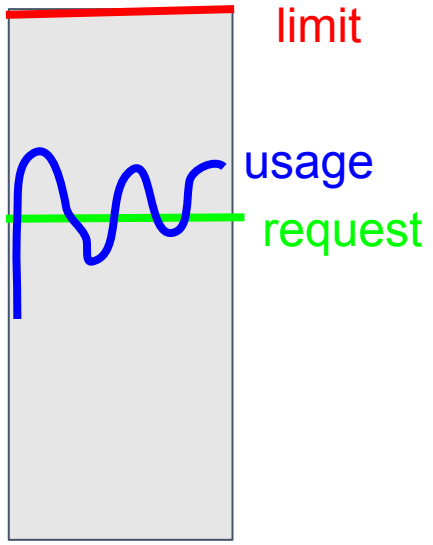


Resources Management Goals

- Efficient allocation of infrastructure resources.
  - underutilized: cost-inefficiency.
  - over-subscribed: failures, downtime, or missed SLAs.
- Resource and performance Isolation
  - a workload should not use up all resources
- Guarantee system stability
  - make sure critical system processes have enough resources

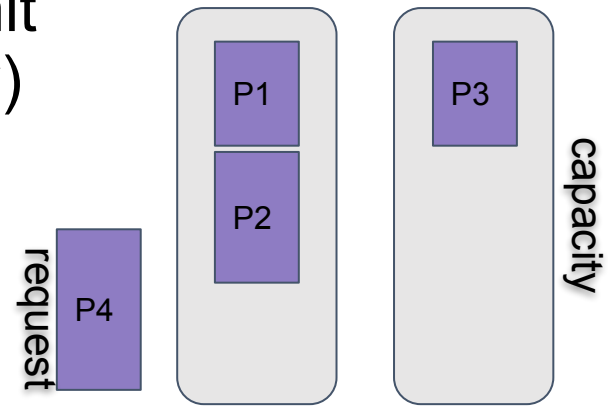
Resource in Kubernetes

- Resources
  - requested by, allocated to, and consumed by a pod/container
  - compressible (CPU) or incompressible (memory)
- Resource Model
  - Desired State (specification)
    - **request**: the amount of resources requested by a container/pod
    - **limit**: an upper cap on the resources used by a container/pod
  - Actual State
    - actual resource usage



Resource Management

- Efficient allocation
  - scheduler finds the “best” host that satisfies the resource request
  - reduce the chance of resource overcommit
- Resource isolation
  - makes sure the actual usage is under the resource limit
  - actions could be throttle (CPU), kill container (memory)

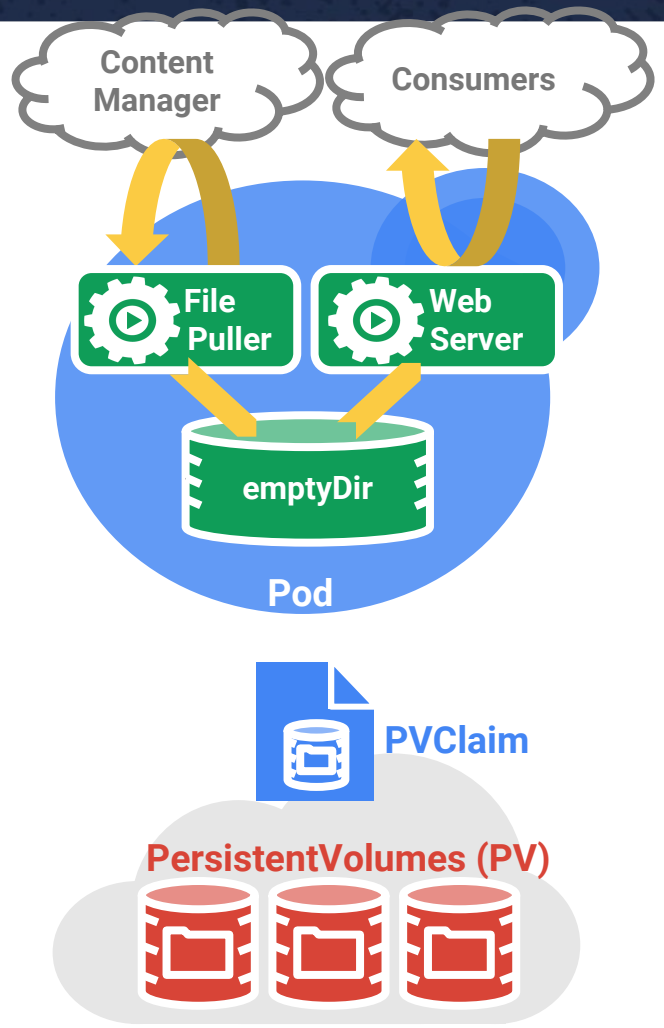


Agenda

- Motivation
- Resource management and model
- Storage Overview
- Local Ephemeral Storage Management
- Future Work

Kubernetes Storage

- Ephemeral
  - container: writable layer and logs
  - pod: volumes (emptyDir, secrets, configMap,...)
- Persistent
  - dedicated disks
    - remote (network attached storage) or dedicated local disk
  - explicit lifetime outlives containers/pods
    - represented by volumes (PVC/PV)



Local Ephemeral Storage Management

- Efficient allocation
  - support local ephemeral storage as a resource
- Resource isolation
  - avoid single pod or container uses up all disks
- System stability
  - reserve certain amount of local storage for system use to make system more stable

Container-level

User sets `ephemeral-storage` resource requirements

Container

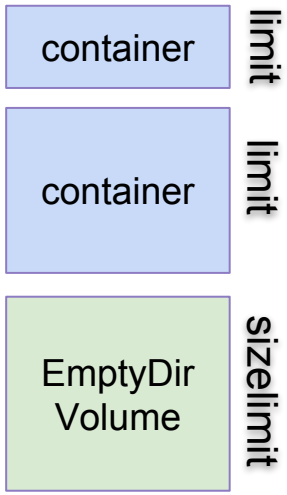
- request: guaranteed resource to the container
- limit: maximum resource allowed to use
  - if container's usage exceeds its limit, pod will be evicted

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
  - name: wp
    image: wordpress
    resources:
      requests:
        ephemeral-storage: "10Gi"
      limits:
        ephemeral-storage: "10Gi"
```

Pod-level

User can set ephemeral volume sizeLimit

- emptyDir volume ***sizeLimit***
  - If emptyDir volume usage exceeds its limit, pod will be evicted



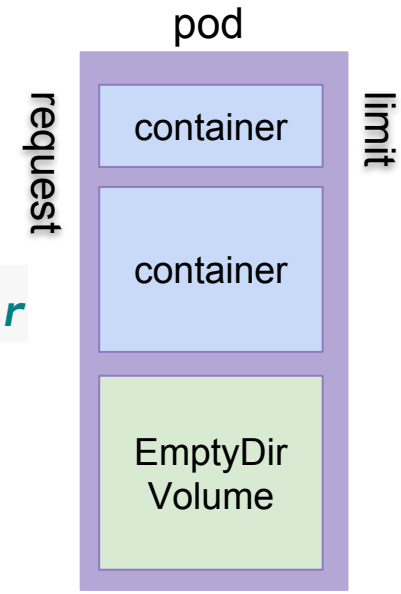

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
  - name: wp
    image: wordpress
    resources:
      requests:
        ephemeral-storage: "10Gi"
      limits:
        ephemeral-storage: "10Gi"
  volumeMounts:
     - mountPath: /cache
       name: cache-volume
  volumes:
   - name: cache-volume
     emptyDir:
       sizeLimit: 20Gi
```

Pod-level

User cannot set explicit pod-level resources, implicitly, sum of the containers' resources

Pod

- Request: `(2+10)Gi`
- Limit: `(4+10)Gi`
- Usage: `Usage_containers + Usage_emptyDir`




```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
  - name: wp
    image: wordpress
    resources:
      requests:
        ephemeral-storage: "10Gi"
      limits:
        ephemeral-storage: "10Gi"
  volumeMounts:
     - mountPath: /cache
       name: cache-volume
  volumes:
   - name: cache-volume
     emptyDir:
       sizeLimit: 20Gi
```

QoS Classes

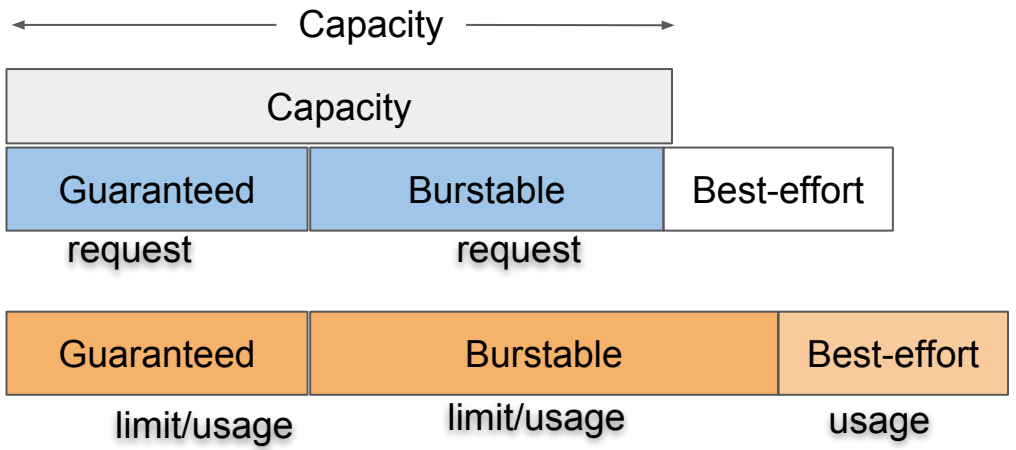
Based on request/limit set, pods have different QoS

- Guaranteed
  - 0 < request == limit
  - pods are guaranteed to not be killed until exceeding the limit
- Burstable
  - 0 < request < limit
  - pod might use more resources than request, more likely to be killed
- Best effort
  - no request/limit specified, lowest priority

Node-level

- Allocation
  - ∑Pod request < Capacity, but
  - ∑Pod limit > Capacity
  - ∑Pod usage > Capacity
- Disk Pressure
  - reclaim resources: delete dead pods and unused images
  - evict pods: choose victim pods in the order of their QoS

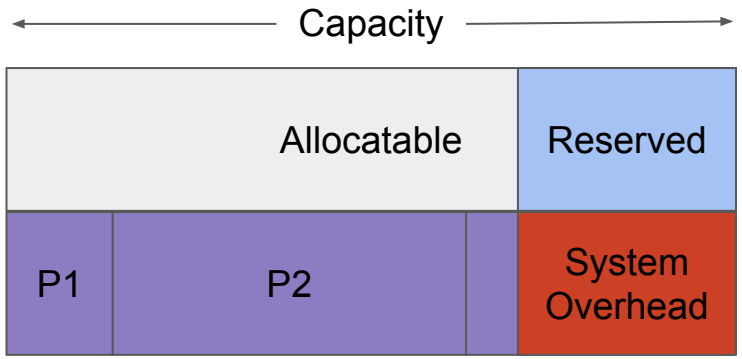
```
--eviction-hard="nodefs.available<1Gi"
--eviction-soft="nodefs.available<2Gi"
--eviction-soft-grace-period="nodefs.available=1m"
```



Node-level

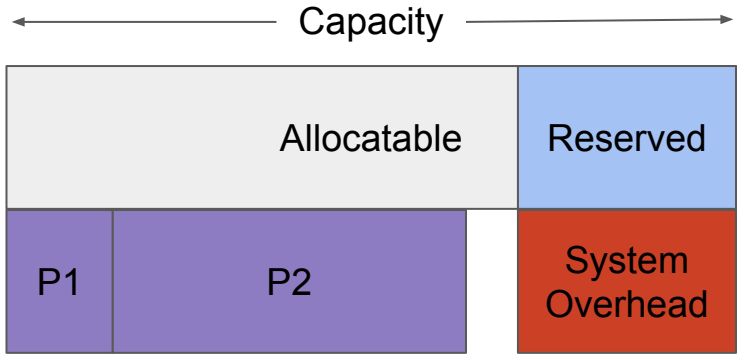
System processes also compete resources with user pods

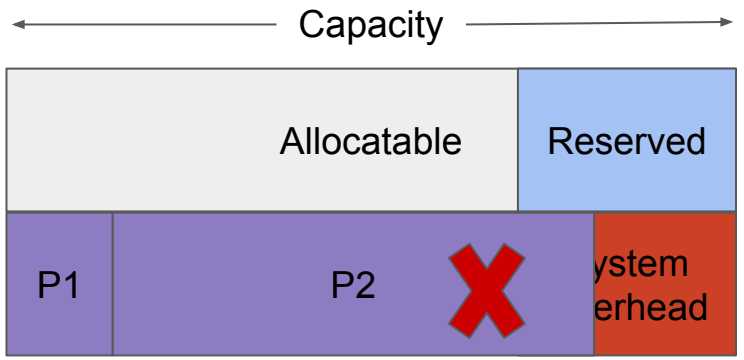
- Allocatable resource
  - how much resources can be allocated to users' pods
  - allocatable = capacity - reserved (system overhead)
- How much to reserve?
  - overhead = Node Usage - ∑Pod Usage
  - roughly proportional to capacity
    - kubelet usage: O(#Pods)



Node-level Allocatable

- Scheduling
  - allocatable is sufficient for the request
  - constraints: ∑Pod request < Allocatable
- Eviction
  - make sure guaranteed reserved resources
  - evict if ∑Pod usage > Allocatable





Eviction Policy

- Pod priority
  - alpha feature in release 1.8
  - the importance of a Pod relative to other Pods.
- Eviction policy
  - evict pods where usage > requests
  - rank pods by priority
  - rank pods by usage-requests

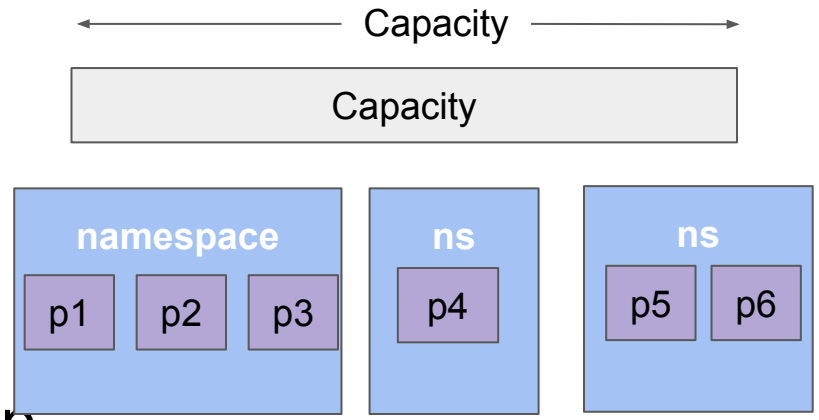
```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: db
    image: mysql
 priorityClassName: high-priority
```

```yaml
apiVersion:
scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class
should be used for XYZ service pods
only."
```

Namespace-level

How do teams/groups share resources?

- Namespace
  - partition resources into a logically named groups
  - ability to specify resource constraints for each group



Namespace-level Resource

- Quota
  - resource isolation among namespaces
  - quota object specifies total requests/limits in namespace
  - ∑Pod request <= request quota
  - ∑Pod limit <= limit quota

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: demo
spec:
  hard:
    requests.ephemeral-storage: 10Gi
    limits.ephemeral-storage: 15Gi
```

During pod creation, quota is checked against the resource requests/limits

Namespace-level

- LimitRange
  - Configure default requests and limits for a namespace

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
spec:
  limits:
  - default:
      ephemeral-storage: 1Gi
    defaultRequest:
      ephemeral-storage: 256Mi
    type: Container
```

Summary

- Support local storage as first-class resource
  - able to set resource limit/request
- Support local storage resource management at
  - container/pod-level: resource allocation and limitation
  - node-level: allocatable to ensure system stability
  - namespace-level: support isolation among teams

Future Work

- Disk IO isolation
- Extend Metrics API to include local ephemeral storage
- Pod-level resource request/limits
- Dynamic resource management
  - static resource setting might not be appropriate
  - resource requirements might change dynamically
  - system process resource consumption might change

Acknowledgement

Kubernetes team work

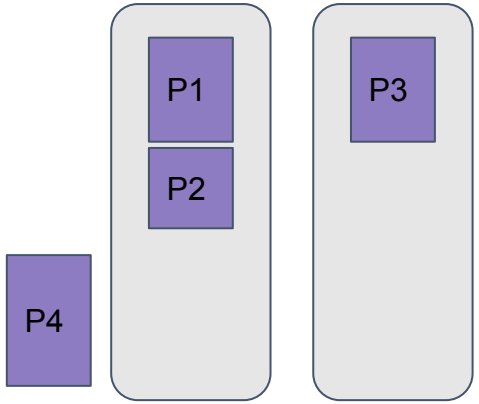
- Saad Ali, Michelle Au, Cheng Xing, David Zhu (Storage Team)
- Vishnu Kannan, David Ashpole, Tim Allclair, …

Community Contributor

- Nick Ren

Resource Management

- Efficient allocation
  - scheduler finds the “best” host that satisfies the resource request
- Resource isolation
  - kubelet monitors the resource usage, and makes sure the actual usage is under the resource limit
  - actions could be throttle, kill container, evict pod
- System Stability
  - reserve resources for system processes
  - evict pods if allocatable is not enough for all running pods



Eviction

- Order
  - best effort: consume the most of the starved resource are failed first
  - burstable: pods that consume the greatest amount of the starved resource relative to their request for that resource are killed first
  - guaranteed: pods that consume the greatest amount of the starved resource relative to their request are killed first