kube-arbitrator: Policy based resource sharing in Kubernetes (2)

Da Ma (@k82cn, madaxa@cn.ibm.com)

User Cases: Run multiple type of workloads in Kubernetes

- Long running service (app area) & bigdata (bigdata area) can share resources:
  - Support define resource usage of each area, e.g. 50% resources to app area, 50% to bigdata area.
  - Support **borrow/lending** protocol: if the resources is idle in one area, it can be lend out and be preempted back when launch more tasks
- Run multiple cluster in bigdata area, e.g. Hadoop & Spark:
  - Support define resources usage of each cluster within bigdata area
  - Support sharing resources between those clusters

Motivation & Goals

“Batch” workloads

- Complex resource requirements: minimum number of resources, application notification requirements, topology requirements

Resource Sharing/Management (Queue)

- Dynamic resource management between tenants
- Fine-grain scheduling policy

Motivation & Goals

Idea: Introduce a QueueJob and Queue controller in Kubernetes

- Queue = represents tenants for resource sharing
- QueueJob = represents a collection of objects (pods, config maps, volumes, etc) managed atomically
- QueueJobs can be queued when not enough resources are available, preempted and resized

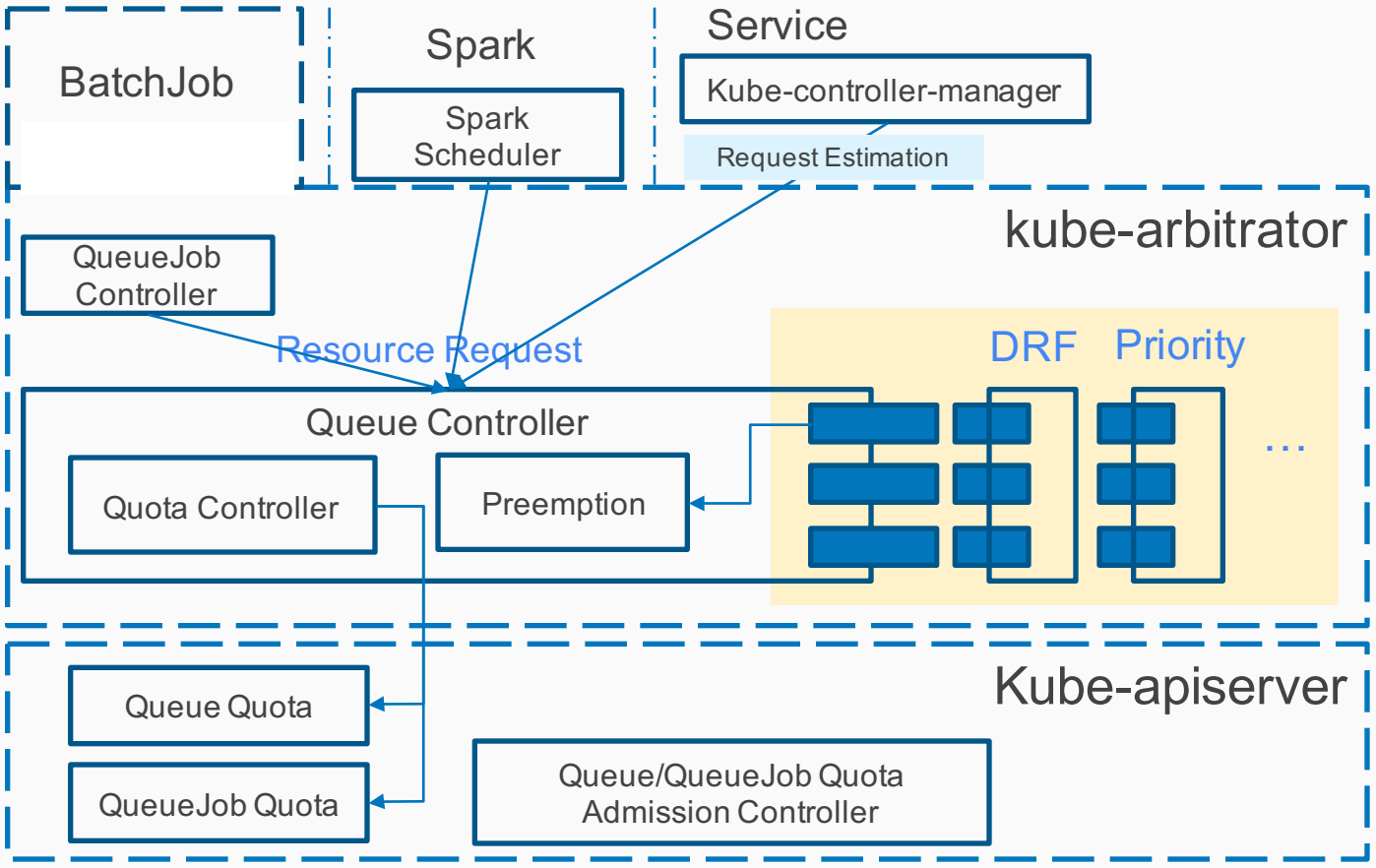
Proposals in the community:

- https://github.com/kubernetes-incubator/kube-arbitrator
- https://docs.google.com/document/d/1-H2hnZap7gQivcSU-9j4ZrJ8wE_WwcfOkTeAGjzUyLA/edit#heading=h.a1k69dgabg0w

Kubernetes features & gaps

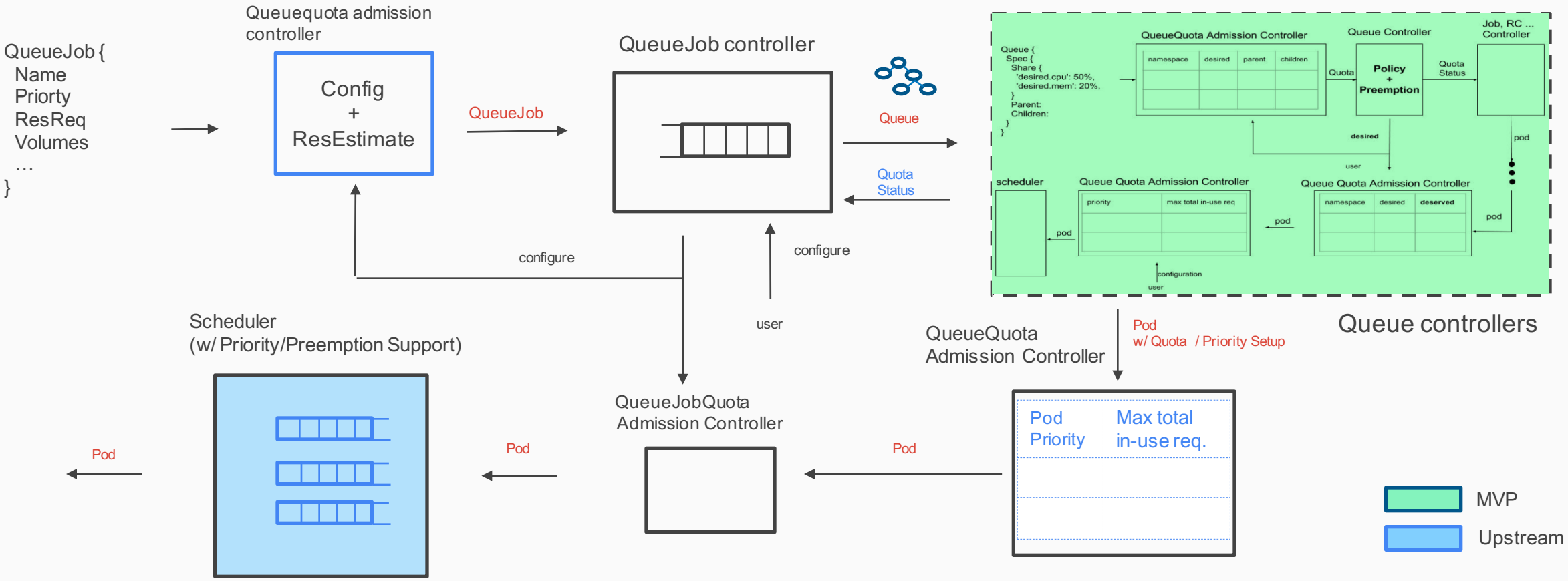
- Admission Controller + Quota: static plan / allocation
- Multiple Scheduler: No QoS
- Auto-Scaling & Node-level QoS: no cluster-level QoS
- Re-scheduling & Preemption/Eviction
- Workload-specific controller & ThridPartyResources

Architect Overview

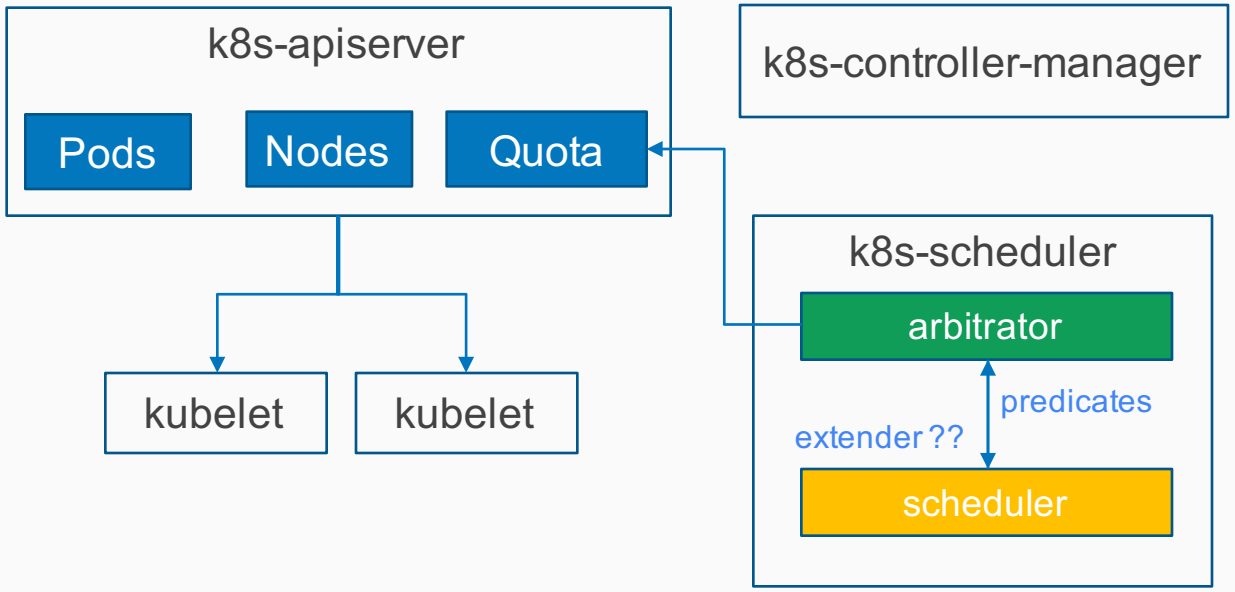


1. kube-arbitrator's target is to support multiple workloads in one cluster
2. QueueJobController support batchjob as default; other workload are supported by different framework, e.g. Spark
3. Queue/QueueJob quota represent resource allocation
4. ResourceRequest is also a CRD to represent request, e.g. CPU, memory, min desired resource, volume.
5. AdmissionController provide metrics for Request Estimatior for Service, e.g. RC (discussing)
6. Priority is one of policies, the default policy will be DRF

Architect Overview



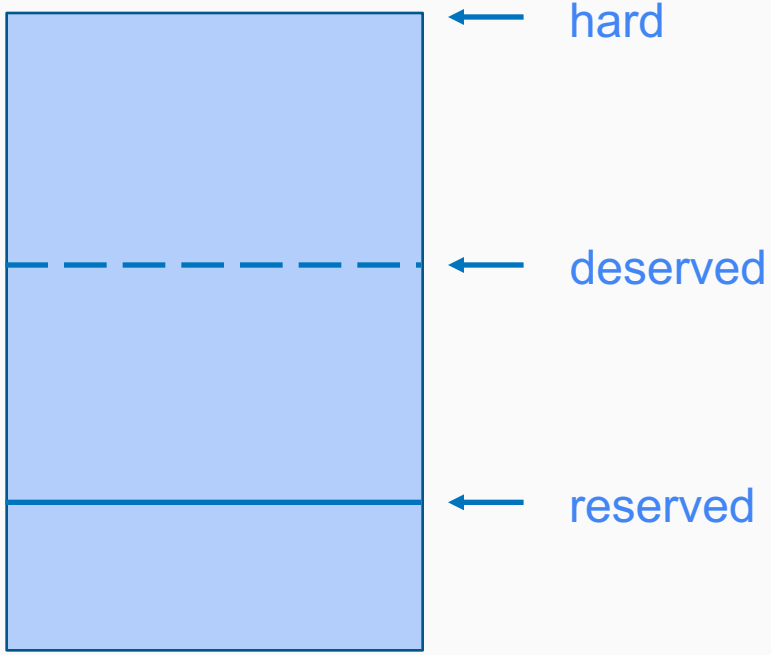
Architect Overview



1. Arbitrator will calculate **deserved** resource (Quota.Deserved) based on Scheduler's configuration, arbitrator's policy, e.g. DRF, and namespace's request (pending pods)
2. Arbitrator will evict Pods of overused namespace
3. Scheduler dispatch tasks based on **Quota** (# of deserved), **Pods** and **Nodes attributes**
4. Scheduler **predicates** "namespace will not be overused"

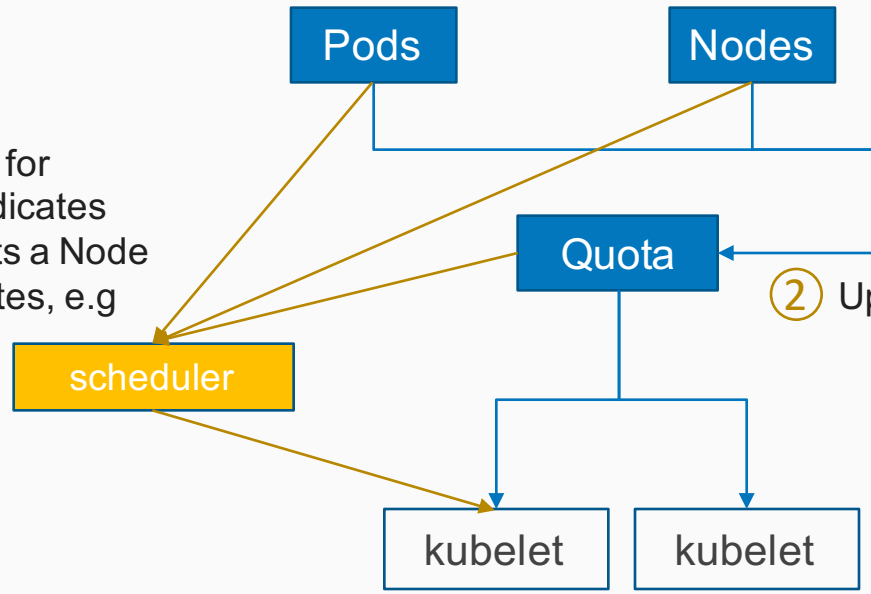
Architect Overview

Quota



1. Only ***Compute Resource Quota*** and ***Storage Resource Quota*** are available for reserved & deserved.
2. The **reserved** section defines the resources that reserved for the namespace. The total reserved resources can not exceed cluster resources
3. The **deserved** is updated by arbitrator, defines the total resources that allocated to a namespace; the deserved resources can not exceed **Quota.hard** and can not less than **Quota.Reserved** (exception excluded, e.g. Node failed)

Architect Overview



① Calculate **deserved** resources of each namespace based on:

1. Total resources from Nodes
2. Resource requirement from **pending** Pods
3. Resource requirement from **running** Pods
4. Reserved resource of Quota

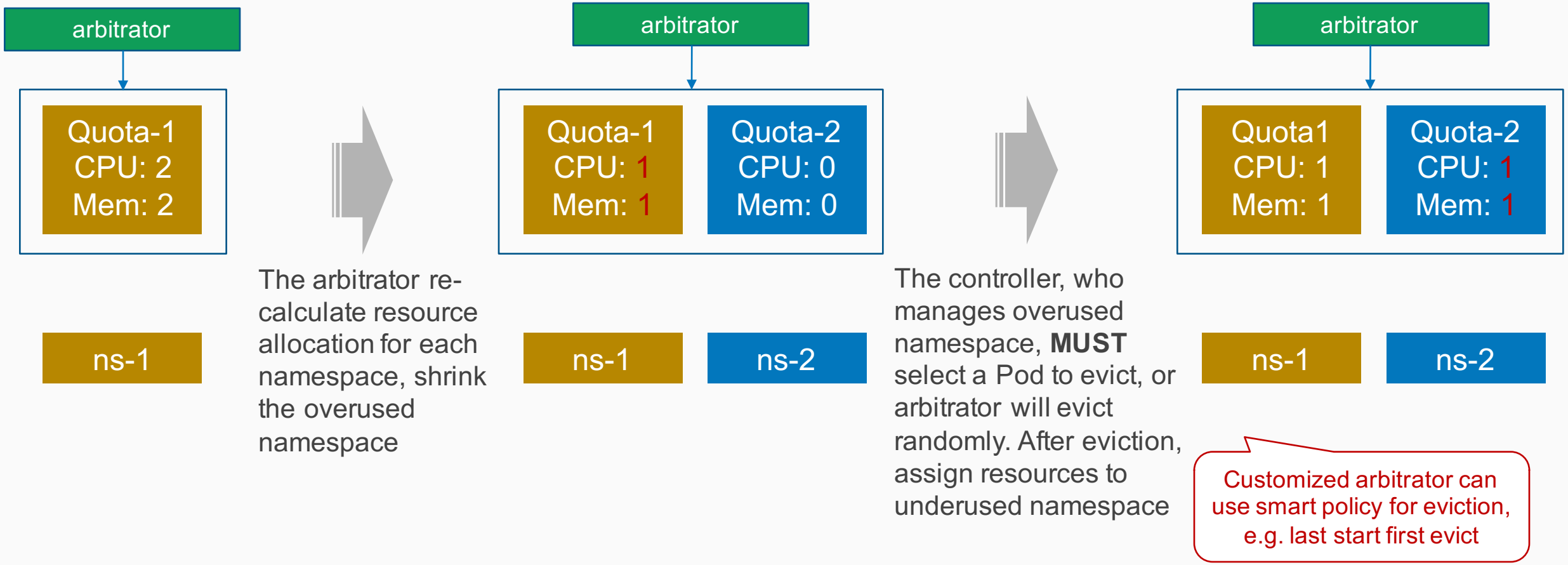
② Update **Quota.deserved**

③ Add scheduler predicate for **Quota.Deserved;** if predicates passed, scheduler selects a Node for **Pod** based on attributes, e.g PodAffinity.

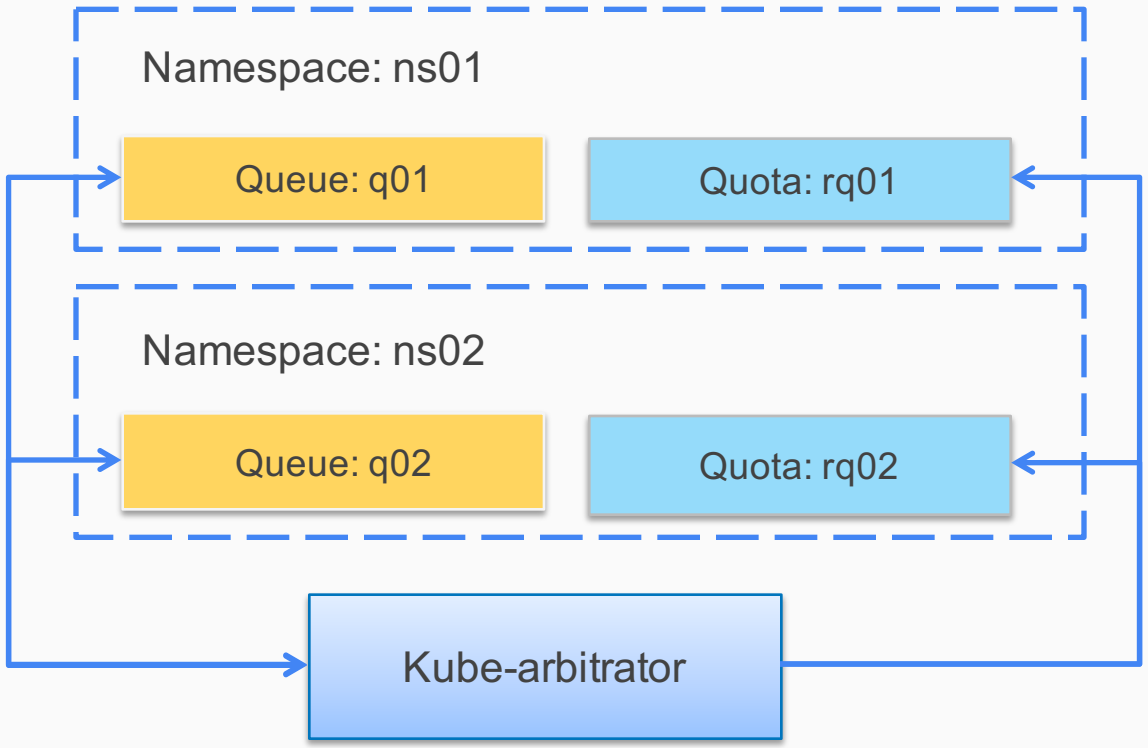
1.5 Update **Eviction** for overused namespace after grace period

④ Before launching, kubelet check scheduler's request: reject overused scheduler's request. if node is exhaust, evict most overused scheduler's request

Pre-emption & Reclaim



Current work



1. The resource allocation is based on namespaces
2. User need to create a queue and a resource quota under a queue
3. A queue contains weight and resource request
4. Kube-arbitrator collect cluster information, include resources, pods, etc.
5. Kube-arbitrator collect all queue information and allocate cluster resources to each queue by weight and resource request.
6. Kube-arbitrator update resource limitation to resource quota
7. User can submit jobs to related namespaces now.

Feature Interaction

- Workload-specific controller

  The arbitrator will also evict overused namespace in workload-specific controller. The workload-specific controller can not use more resource than **Quota.Deserved**. If **Quota.Deserved** updated, it selects one Pods to evict; otherwise, arbitrator will evict pods (e.g. FCFS) after grace period

- Multiple-scheduler

  If enable multiple-scheduler, enable only one arbitrator to avoid race condition

- Admission Controller

  Arbitrator only handles ***Compute Resource Quota*** and ***Storage Resource Quota*** of **ResourceQuotaAdmission**, the other metrics of ResourceQuotaAdmission will follow current behaviors. No impact to other admission plugins

Roadmap of kube-arbitrator

- Fine-grained scheduling
- Resource request
- New quota for Queue/QueueJob

Live demo

Reference

- k8s#36716 : Manage multiple applications in Kubernetes
- Github: kubernetes-incubator/kube-arbitrator
- Design Doc: https://docs.google.com/document/d/1-H2hnZap7gQivcSU-9j4ZrJ8wE_WwcfOkTeAGjzUyLA/edit#heading=h.a1k69dgabg0w

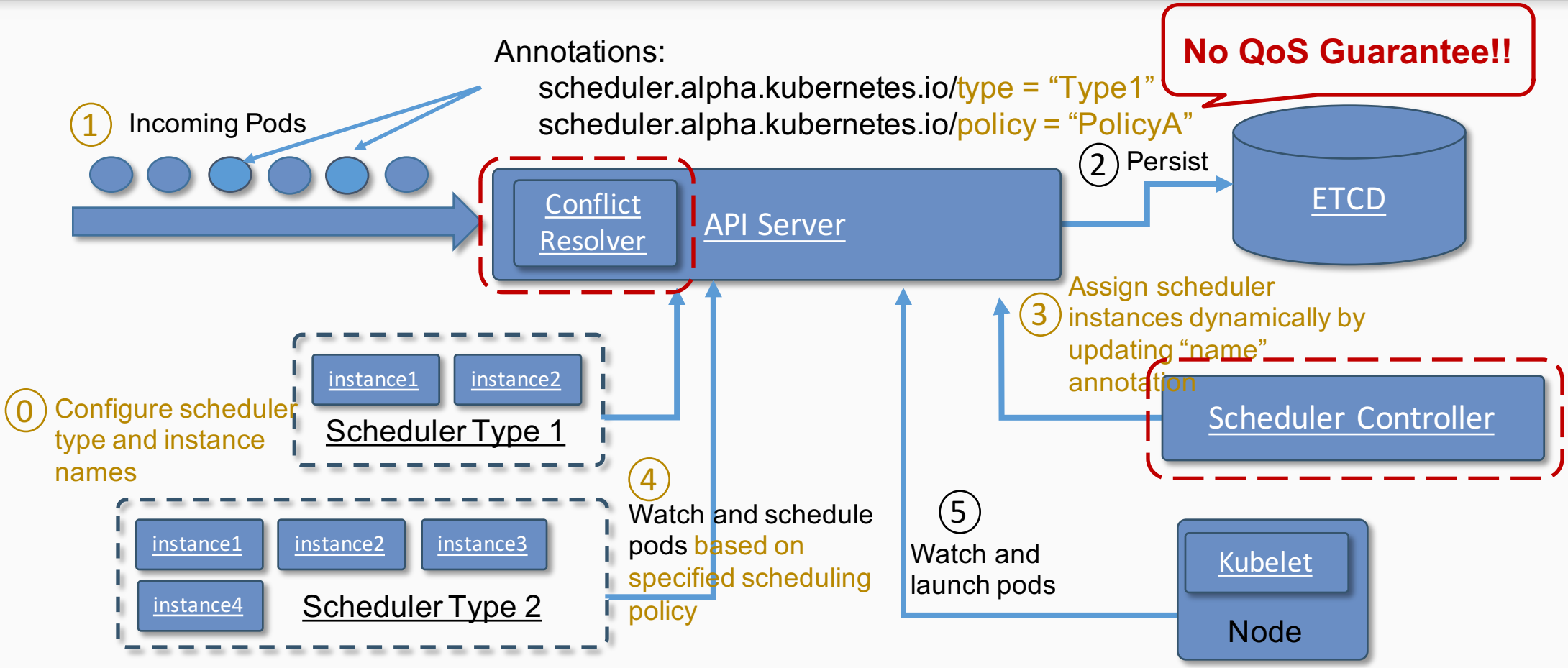
Thank You !

Back Up

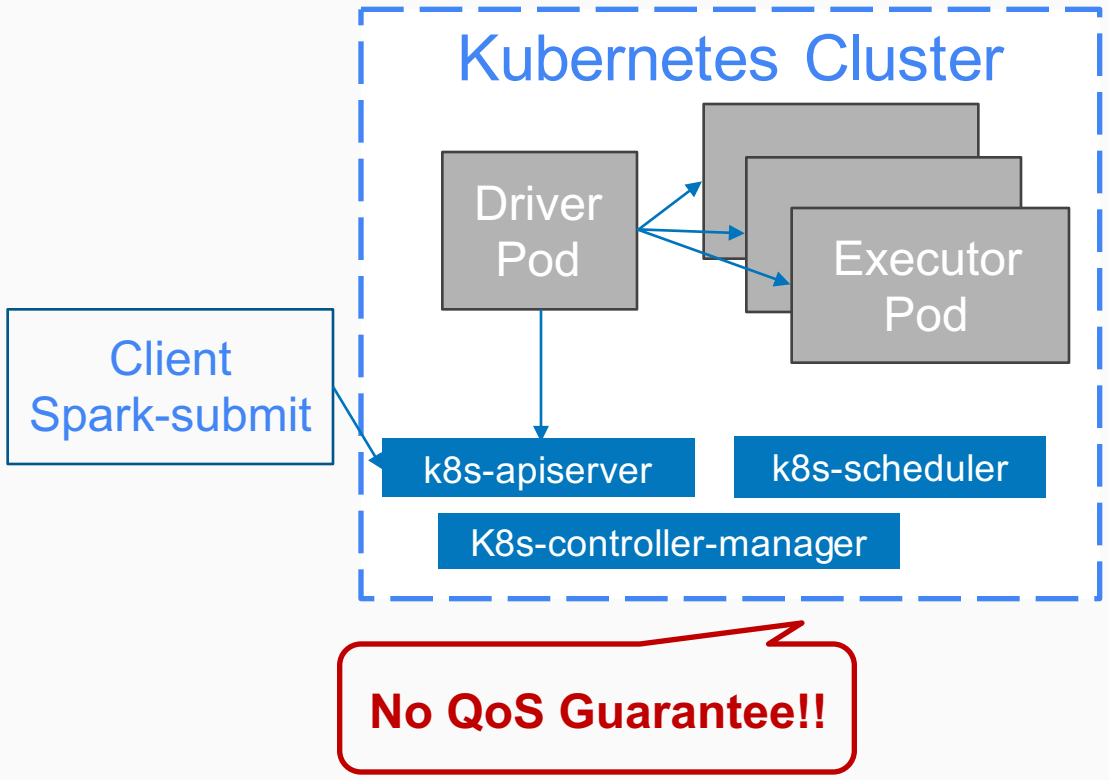
Components

1. Arbitrator
   - Calculate **deserved** resource number (Quota.Deserved) based on Scheduler's configuration & arbitrator's policy, e.g. DRF
   - Arbitrator will calculate deserved resources based on namespace
   - Arbitrator will also consider namespace's request, e.g. pending pods , when calculating deserved resources
   - Arbitrator will calculate the deserved resource by scheduling interval, e.g. 10s
   - If namespace is overused, arbitrator trigger eviction for that namespace with grace period
   - Arbitrator will re-use Quota by adding new section, e.g. deserved
2. Scheduler dispatch tasks based on **Quota** (#), **Pods** and **Nodes attributes**
   - Scheduler decide which host is used based on Pods and Nodes' attributes
   - Scheduler can **not** use more resource than Quota.Deserved

Multiple Scheduler in Kubernetes

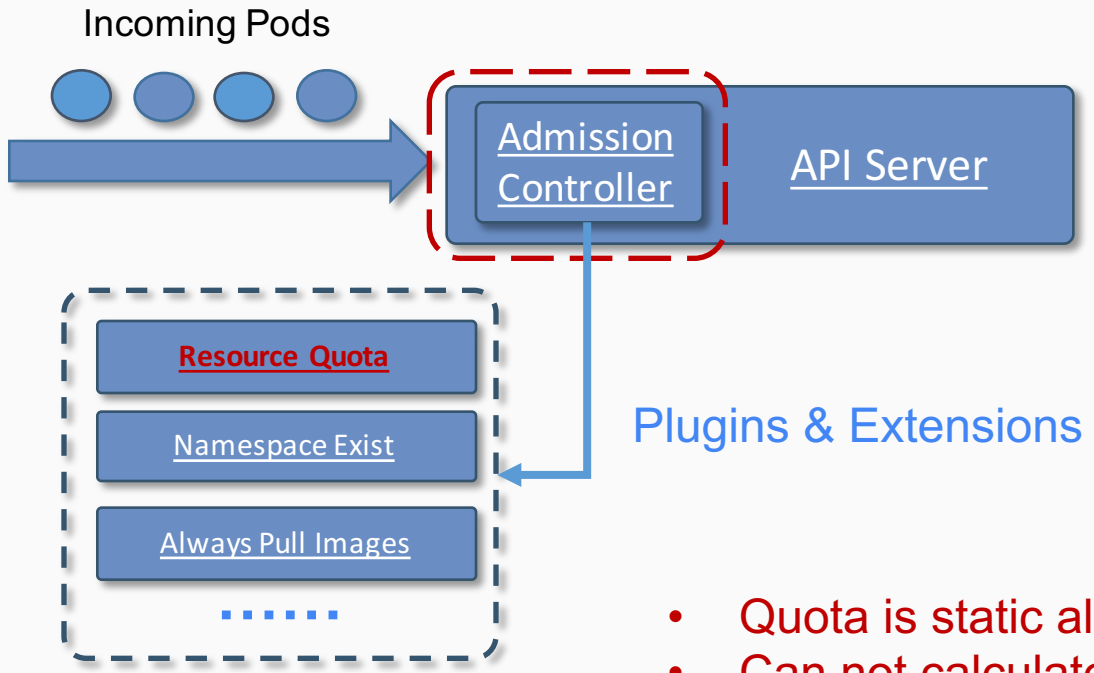


Support Spark natively in Kubernetes



1. Using fabric8io/kubernetes-client library (Java client for Kubernetes & OpenShift 3)
2. New scheduling sub-classes:
   - **KubernetesClusterScheduler**:
     - ✓ Create Driver Pods (from outside of cluster)
   - **KubernetesClusaterSchedulerBackend**:
     - ✓ Create Executor Pods (from driver pod)

Admission & Resource Quota



1. **Admission controller** control and limit the **CRUD** operations clients perform synchronously
2. **ResourceQuotaAdmission** is one of Admission plugins which reject pods creation if exceed Quota

- Quota is static allocation for each namespace
- Can not calculate **deserved** based on namespace's **request** (pending Pods) because of creation rejection

Re-schedule & Preemption

- Arbitrator is the decision maker for **preemption** who defined “priority”, e.g. under-used namespace’s priority is higher than overused namespace’s
- Re-scheduler
  - ❖ Eviction for critical pods, e.g. DNS in ***kube-system***

    Arbitrator trigger the eviction for Pods in ***kube-system***; it always “reserved” enough resources for ***kube-system***
  - ❖ Eviction for better placement

    Default scheduler trigger eviction for better placement; the pods, assigned by workload-specific controller e.g. “computing” tasks, or marked by non-reschedulable are not re-scheduled by default scheduler

Workload-specific Controllers & ThirdPartyResource

- Prototype based on **ThirdPartyResource** & **workload-specific controller**
- General requirement on resource multi-tenant for Kubernetes user
- Can not leverage ***kubelet*** to resolve conflict

Horizontal and Vertical scaling / Node-level QoS

- Horizontal Vertical scaling

  HPA will change replicas of deployment based on metric; not “quota” restriction to each namespace

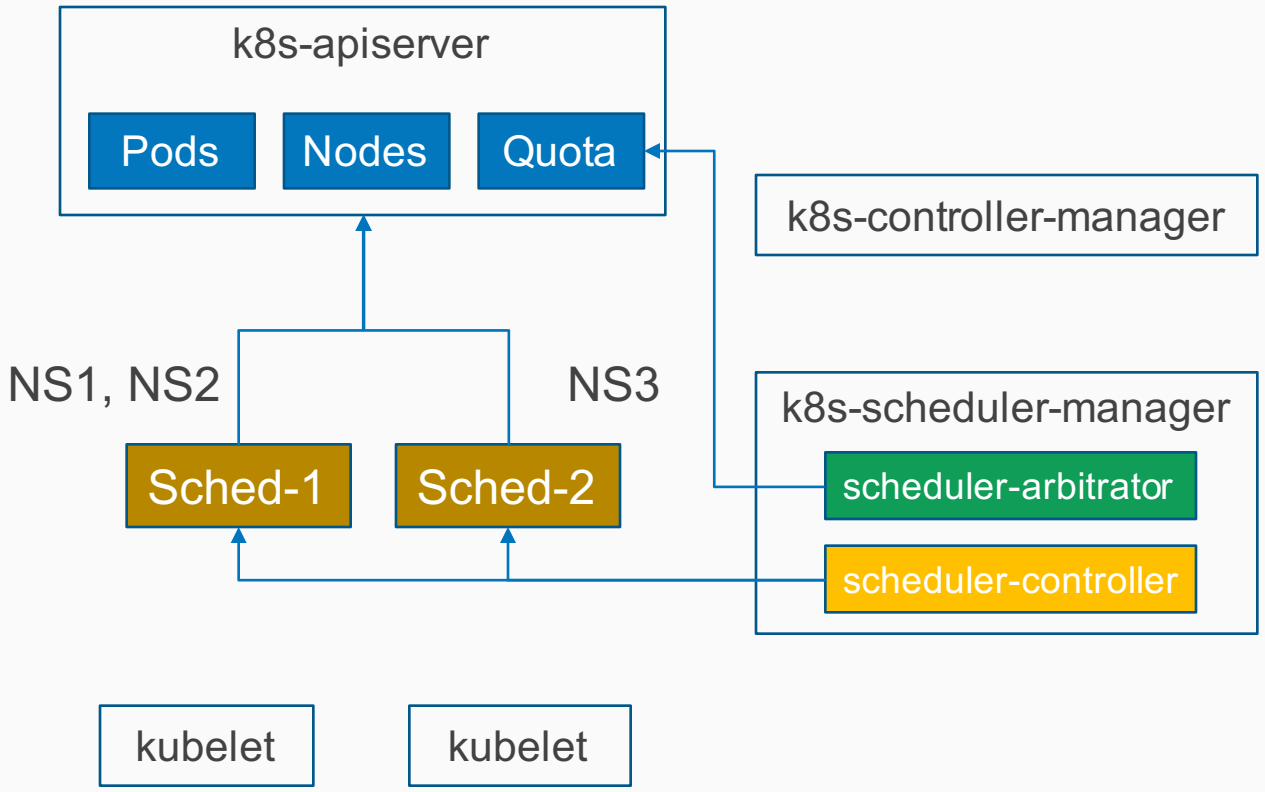
- Vertical Scaling

  On-going, node-level

- Node-level QoS

  Need a cluster-level QoS; but arbitrator’s policy need to include **request** & **limit** of QoS.

Arbitrator with Multiple-Schedulers



1. Scheduler is TPR
2. Scheduler-controller will start schedulers based on RC/RS configuration
3. Scheduler-arbitrator will allocate **deserved** resource (Offer) based on Scheduler's configuration & arbitrator's policy, e.g. DRF
4. Scheduler dispatch tasks based on **Quota** (#), **Pods** and **Nodes attributes**