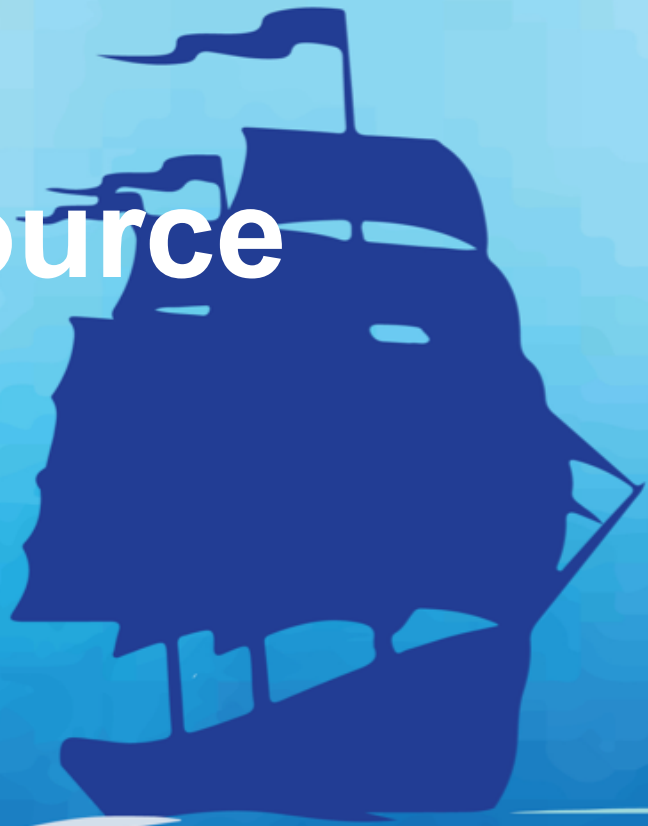
KubeCon | CloudNativeCon

Europe 2018

Inside Kubernetes Resource Management (QoS)

Mechanics and Lessons from the Field



Michael Gasch
Application Platform Architect (VMware)
@embano1

Everything You Ever Wanted To Know About Resource Scheduling... Almost

Tim Hockin <thockin@google.com>
Senior Staff Software Engineer, Google
@thockin

Only a Part of a Bigger Picture

Agenda

A Small Mistake and its Consequences

Operating System Basics

Kubernetes Resource QoS Deep Dive

Best Practices from the Field

QnA

Appendix/Resources

A Small Mistake and its Consequences

A Small Mistake...

kubernetes/ingress-nginx

```
apiVersion: extensions/v1beta1
kind: Deployment
[...]
  template:
    [...]
    spec:
      serviceAccountName: nginx-ingress-serviceaccount
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.14.0
          args:
            [...]
          env:
            [...]
          ports:
          - name: http
            containerPort: 80
          - name: https
            containerPort: 443
          resources: {}
          livenessProbe:
            [...]
          readinessProbe:
            [...]
          securityContext:
            runAsNonRoot: false
```

WHOOPS 💩

…and its Consequences

During Admission, this Pod might be

- Rejected (ResourceQuota)
- Modified (LimitRanger)

After Creation, this Pod might

- Not get enough Resources (“Starvation”)
- Negatively affect other Pods or Host Services (“Noisy Neighbor”)
- Be evicted first by the Kubelet
- Be OOM_killed first (OutOfMemory)

In General, this Pod does not have predictable Runtime Behavior.
Depending on your Workload, that might be OK though.


SO WHAT?
WHO CARES?

Some User Stories

Shahidh K Muhammed Follow
Design Engineer by training, Polyglot by passion, @HasuraHQ by choice, Kubernetes by chance!
Apr 17 · 5 min read

Debugging TCP socket leak in a Kubernetes cluster

runtime: long GC STW pauses (≥80ms) #19378

Closed obeattie opened this issue on 3 Mar 2017 · 9 comments

obeattie commented on 3 Mar 2017 • edited

1 year, lessons learned from a 0 to Kubernetes

Container resource consumption—too important to ignore

My First Ingress Outage

szuecs commented 5 days ago

We also run into this issue in production, start investigating latency critical applications moving to kubernetes.
This is kind of a blocker for production clusters, because you can not set CPU limits for all latency critical applications. When we dropped the CPU limits from our ingress controller we got 12-20x less latency for the p99. The ingress controller is consuming about 30m CPU on average (kubectl top pods), but even setting to 1500m did not dropped the p99 very much. We got only improvements of factor 2-

Application Pauses When Running JVM Inside Linux Control Groups

Causes and Solutions

Don't Let Linux Control Groups Run Uncontrolled

Addressing Memory-Related Performance Pitfalls of Cgroups

Zhenyun Zhuang November 28, 2016

iner_cpu_cfs_throttled_seconds_total is not appearing anymore
d from 60ms and 100ms to ~5ms.
kube-apiserver deployment.

How to Handle Java OOM Errors

Share on

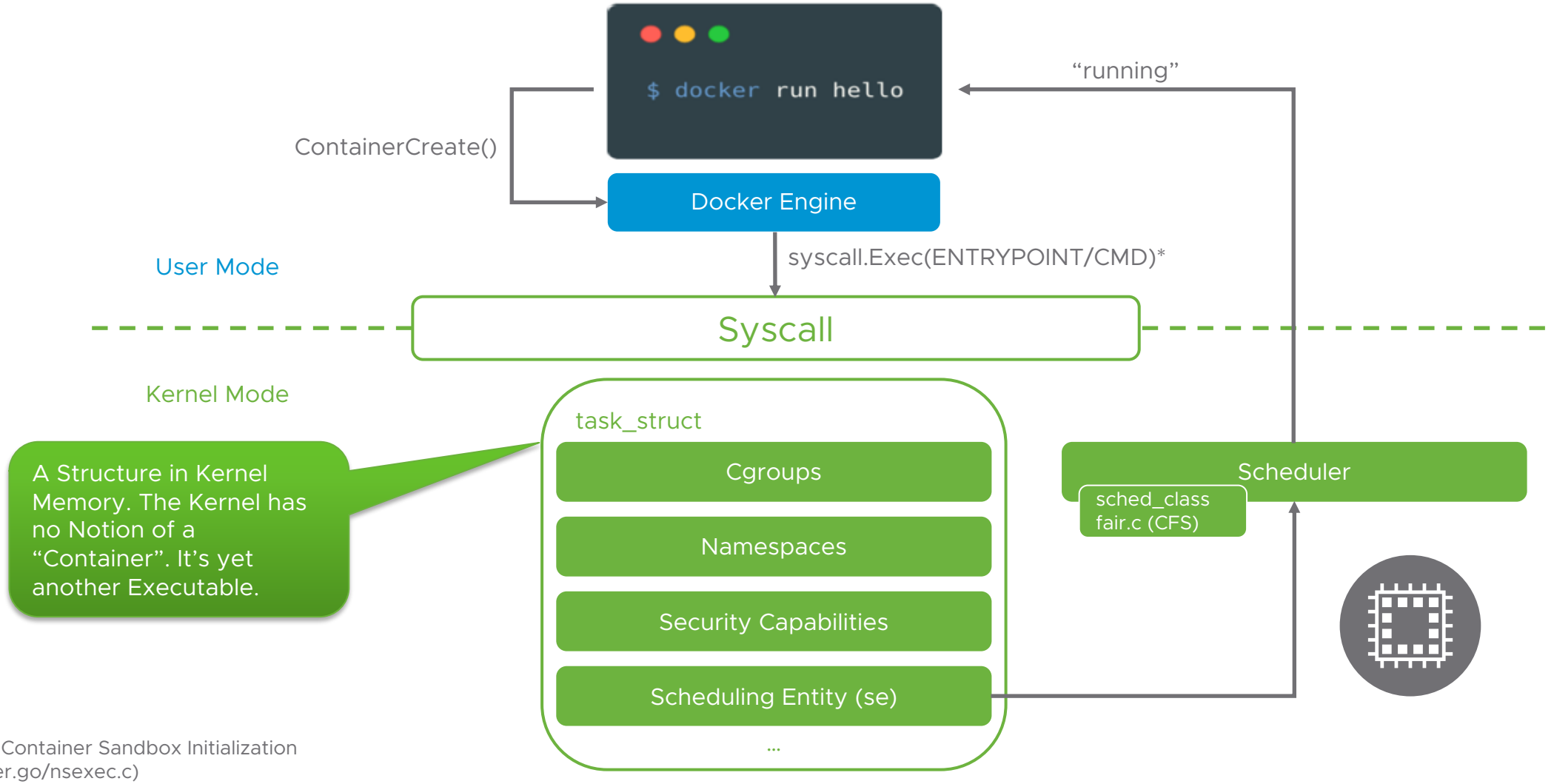
Like 9

usage with ASP.NET Core on Kubernetes

August 17, 2017 · KUBERNETES .NET-CORE ASP.NET-CORE

Operating System Basics

What happens when you “docker run”?



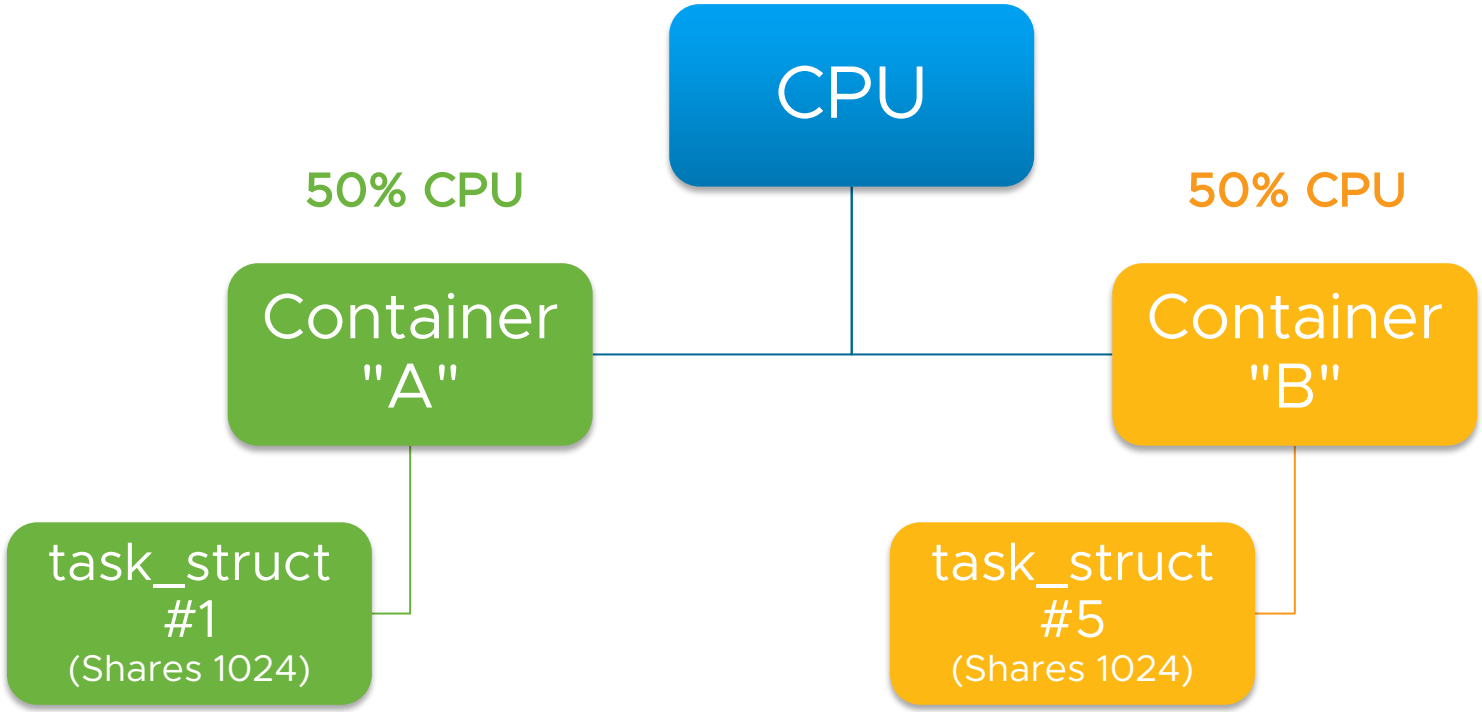
* After Container Sandbox Initialization (nsenter.go/nsexec.c)

vmware


SOUNDS TOO EASY

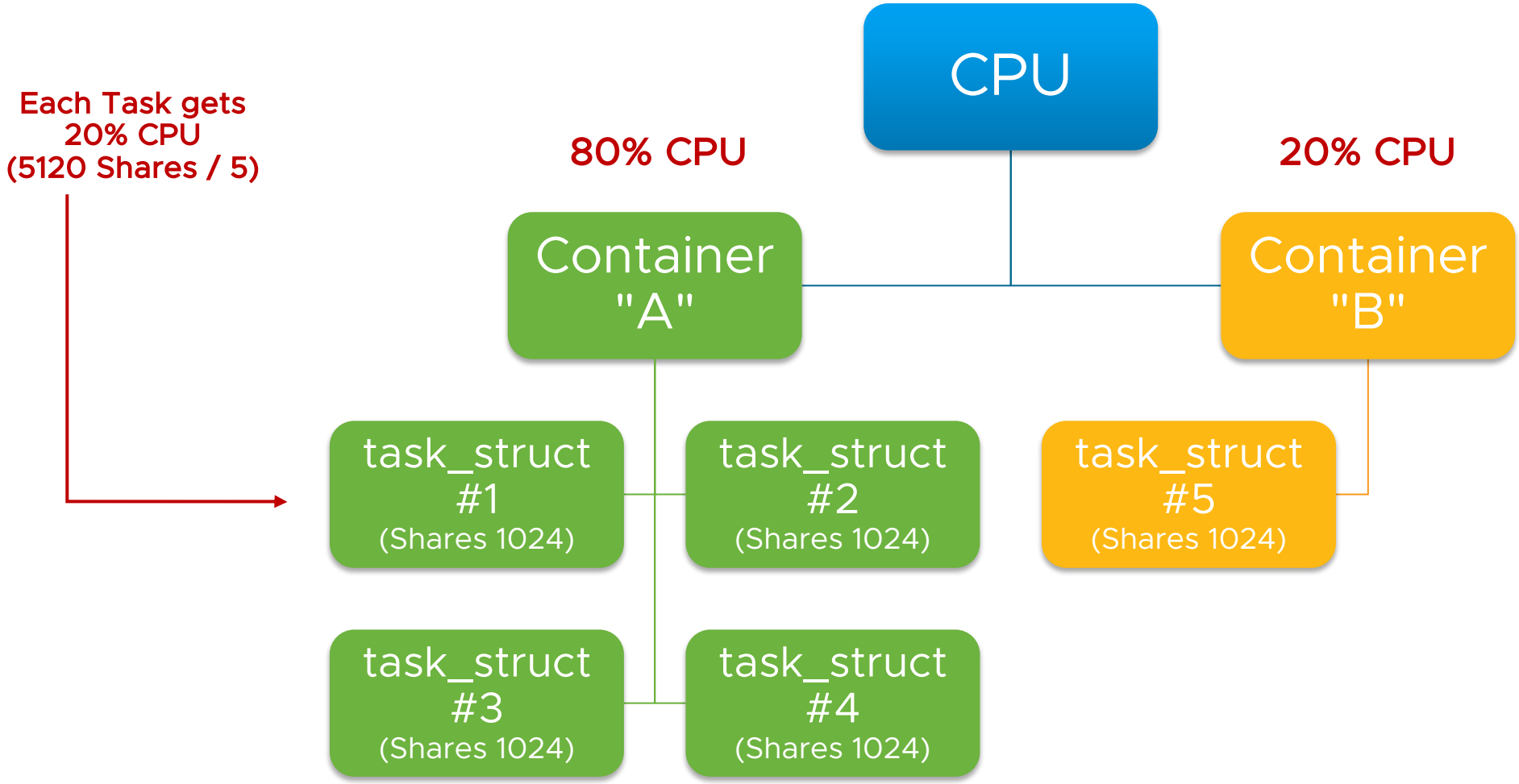
“Fairness” != “Fairness”

The Linux Completely Fair Scheduler (CFS)



“Fairness” != “Fairness”

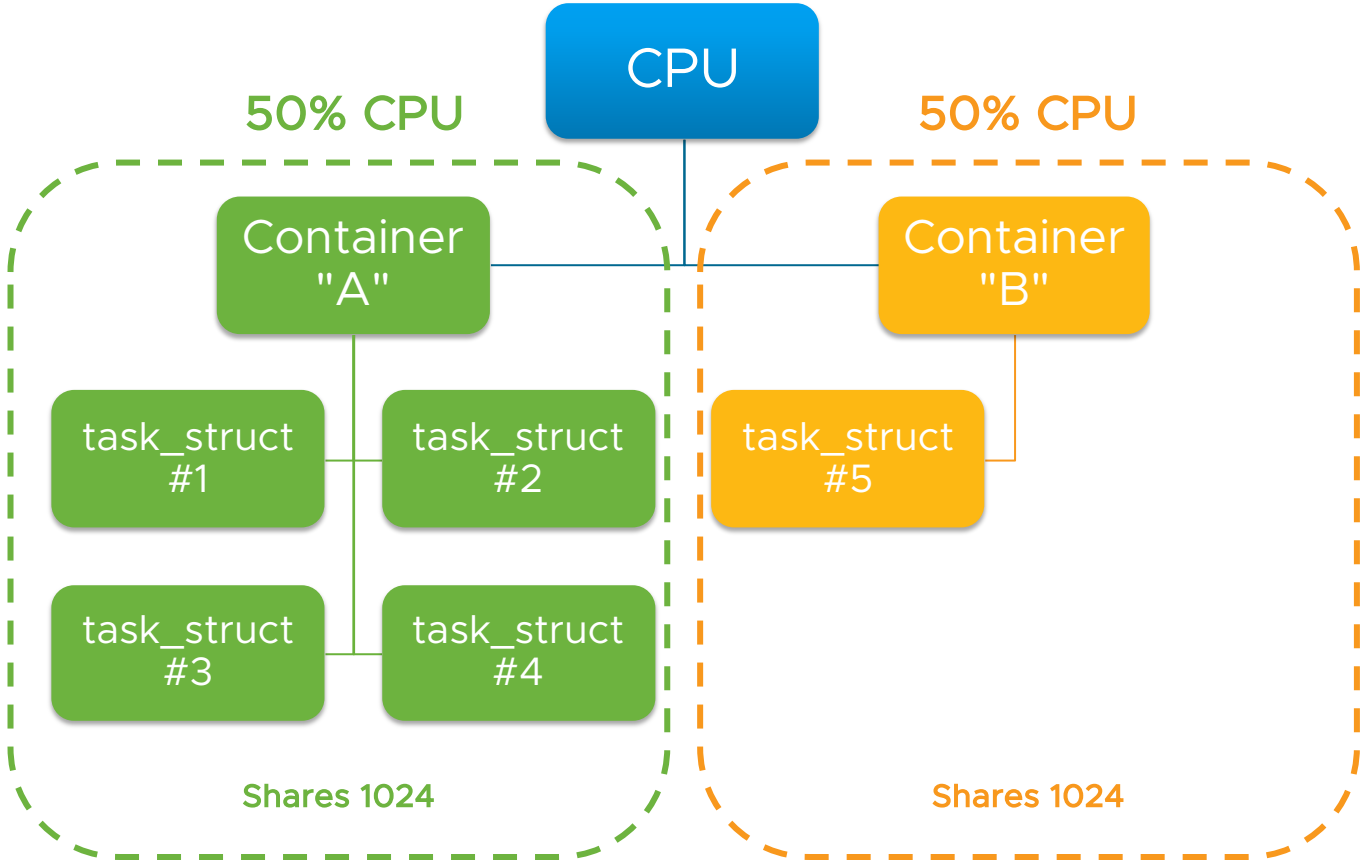
The Linux Completely Fair Scheduler (CFS)



Enter Linux Control Groups (Cgroups)

The Foundation of Resource QoS

Mechanism for Task Grouping, Accounting and Resource Management (Controllers)



Enter Linux Control Groups (Cgroups)

The Foundation of Resource QoS

Example Controllers

- CPU
- Memory
- IO

Use Cases

- Prioritize Workloads (e.g. CPU Usage)
- Limit Resources (e.g. Memory, PIDs)
- Accounting

Interface and Hierarchy typically mounted to /sys/fs/cgroup

Two Versions exist in the Linux Kernel (v1 and v2 Interface)

- v1 is still the Default used by all (?) Container Runtimes

Docker made it easy to use Cgroups

But DO understand the --cpus and -m (Memory) Flags!

```
$ docker run --cpus 1 -m 200m --rm -it busybox top
```

Container View

```
Mem: 1734492K used, 2311704K free, 9028K shrd, 17772K buff, 874376K cached
CPU0:  50.0% usr   0.0% sys   0.0% nic  50.0% idle   0.0% io   0.0% irq   0.0% sirq
CPU1:  75.0% usr   0.0% sys   0.0% nic  25.0% idle   0.0% io   0.0% irq   0.0% sirq
CPU2:   0.0% usr   0.0% sys   0.0% nic   100% idle   0.0% io   0.0% irq   0.0% sirq
CPU3:   0.0% usr   0.0% sys   0.0% nic   100% idle   0.0% io   0.0% irq   0.0% sirq
Load average: 0.25 0.83 1.46 3/791 6

  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     R     1240   0.0   0   0.0 top
```

Docker made it easy to use Cgroups

Under the Hood

Container Host View

```
$ cat /sys/fs/cgroup/cpu/docker/3ecf6640f9acec5866e0b3053912416dde3de7776c89de14ba6b8ece15e950ef/cpu.{shares,cfs_*}
1024        # cpu.shares        (*relative* weight)
100000      # cpu.cfs_period_us (*absolute* enforcement interval in µs)
100000      # cpu.cfs_quota_us  (*absolute* limit in µs)

$ cat /sys/fs/cgroup/memory/docker/3ecf6640f9acec5866e0b3053912416dde3de7776c89de14ba6b8ece15e950ef/memory.limit_in_bytes
209715200   # *absolute* memory limit in bytes
```

--cpus 1

-m 200m

Not a “Guarantee”, just a Weight Value! (Default = 1024)

Hard Limit, i.e. no Reservation!

Not Clock Speed (GHz)! Can vary, e.g. CPU Throttling, different Host Hardware, etc.

Recap

Operating System Basics

From the Linux Kernel View, Containers are “normal” Processes (task_struct)

The default Linux Kernel Scheduler Algorithm is “Completely Fair”

Containers, using Cgroups and Namespaces, provide a certain Level of Prioritization and Isolation on the Host

But how to do it at Cluster Scale?

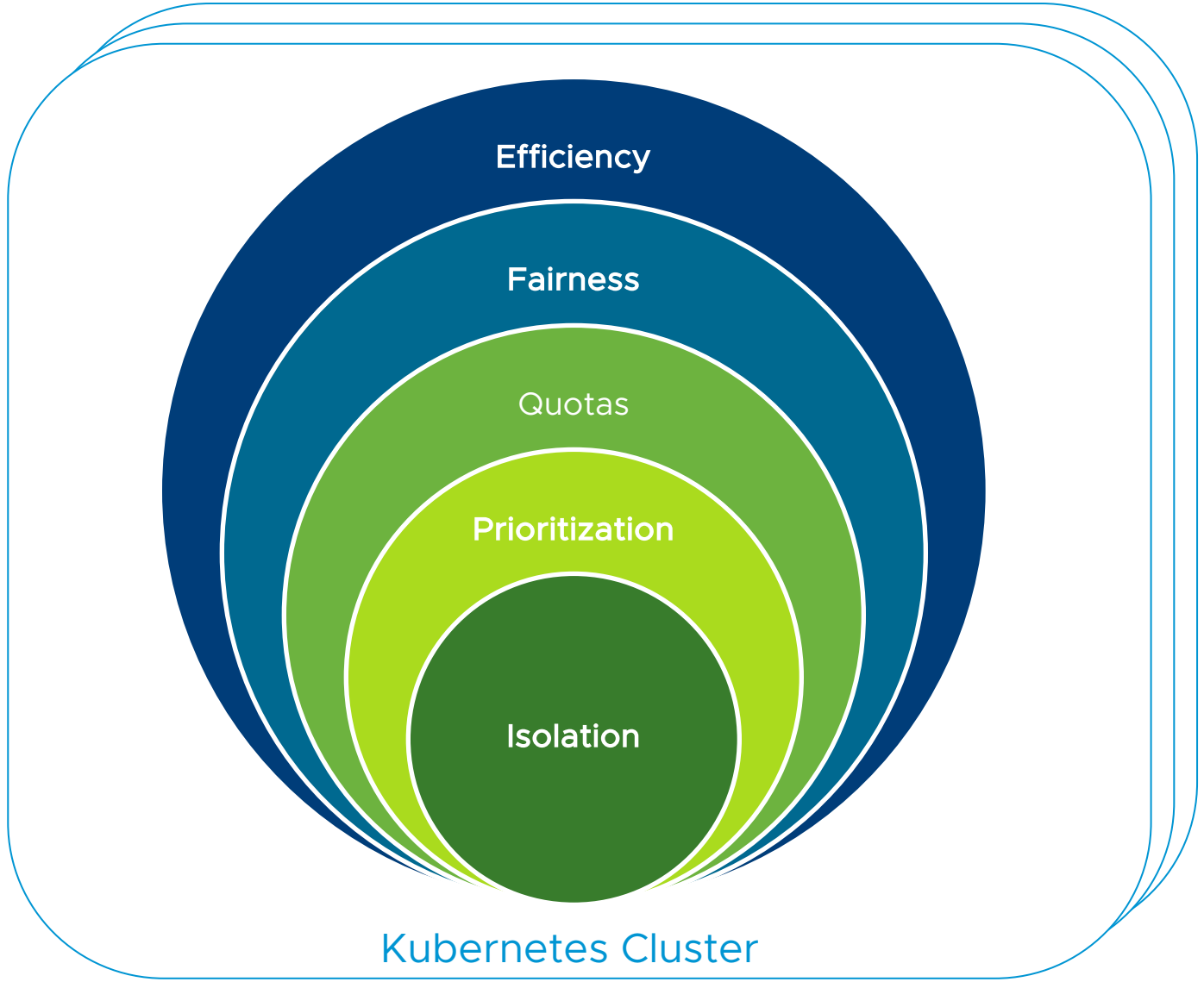
Kubernetes Resource QoS

Deep Dive
(Kubernetes v1.10)

vmware

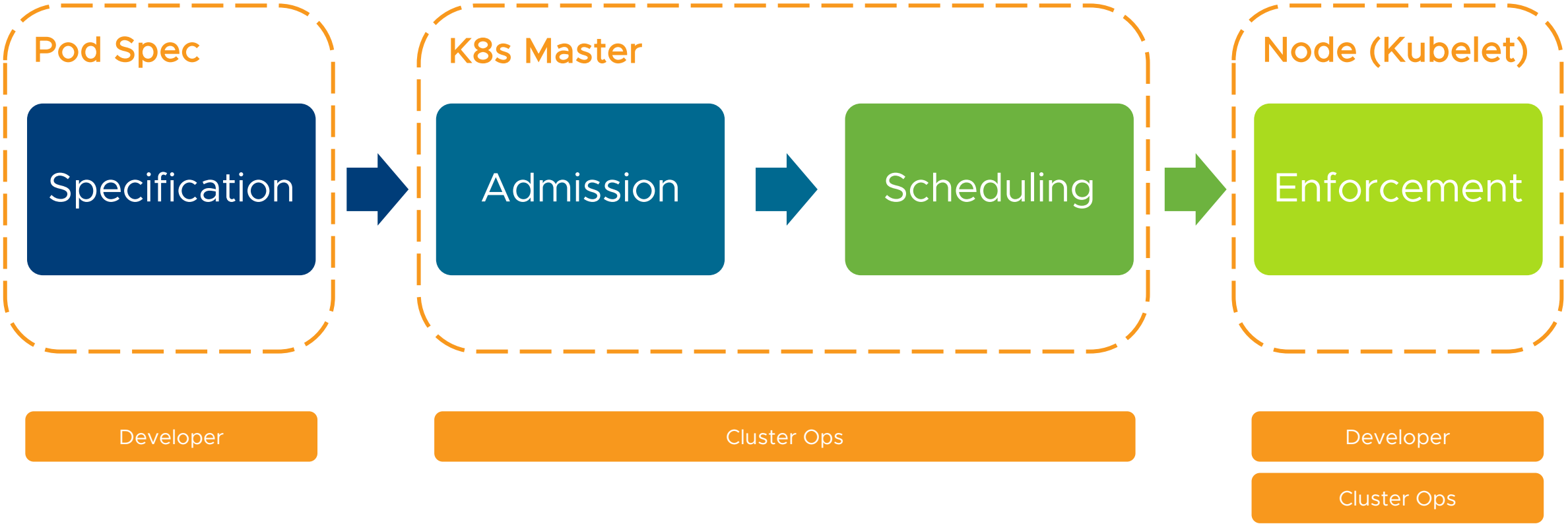
Kubernetes Resource QoS Use Cases

Kubernetes Cluster is the QoS Boundary



QoS Lifecycle, Admission and Enforcement

30k Feet View



Supported QoS Resources in the Pod Spec

Specification

The following Resources can be specified

- Stable
  - cpu (in absolute MilliCPUs or CPU Fractions, e.g. 0.5)
  - memory (in Bytes, or with Suffixes, e.g. M, Mi, etc.)
- Beta
  - hugepages-<size>
  - ephemeral-storage
  - DevicePlugins (e.g. nvidia.com/gpu)
- Custom
  - Extended Resources (replaced Opaque Integer Resources, OIRs), e.g. Licences, Dongles, etc.

Supported QoS Resources in the Pod Spec

Specification

For each Resource, Requests (R) and Limits (L) can be specified

- Those are specified **at the Container** Level (http://bit.ly/2ExayUD)
- Only cpu, memory and ephemeral-storage allow for Overcommitment
- Read-only Fields (after Creation)

What about Sysctls? https://bit.ly/2HRbqAK

And IO/NET Bandwidth? http://bit.ly/2F9gC2

Supported QoS Resources in the Pod Spec

Example

Specification

Pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources:
      limits:
        cpu: "2"
        memory: 200Mi
status: {}
```

When Requests are omitted:
Request == Limits

Namespace QoS Quotas and Defaults

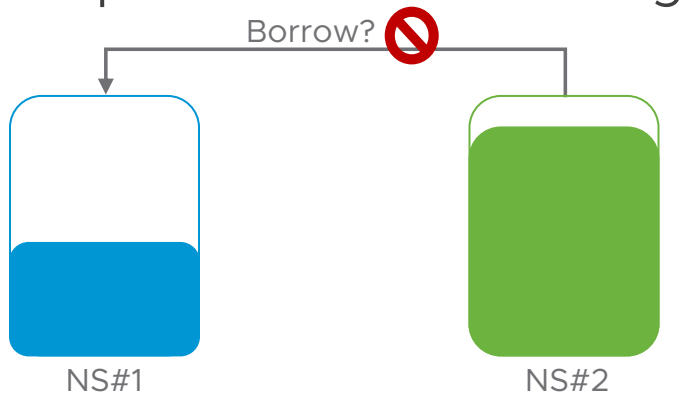
Admission

To control/limit Resource Usage, Namespaces can have Resource Quotas specified

- Enforced during Admission via “ResourceQuota” Admission Controller
- Note:
  - Logical Constraint, i.e. not aware of Cluster Capacity/Usage
  - Pods must adhere to Quota Specification, otherwise will be rejected
  - Running Pods are unaffected by Quota Changes
- Details: http://bit.ly/2GBcloc

Defaults Requests/Limits can be enforced via “LimitRanger” Admission Controller

There is currently no Inter-Namespace Resource Sharing (static Partitioning)



How QoS Affects Scheduling

Scheduler's View of a Node

The Scheduler tracks "Node Allocatable" Resources ("NodeInfo" Cache)

- Note: this is **not** actual Usage
- "Allocatable" typically is < "Node Capacity" (http://bit.ly/2opSBw0)
- Internals: http://bit.ly/2yRHTGo

```
$ kubectl describe no minikube
[...]
Capacity:
 cpu:                2
 ephemeral-storage:  16888216Ki
 memory:             2048052Ki
 pods:               110
Allocatable:
 cpu:                2
 ephemeral-storage:  15564179840
 memory:             1945652Ki
 pods:               110
```

Calculation of "Node Allocatable" Resources on each Node (Kubelet)

How QoS Affects Scheduling

Scheduling Algorithm

Scheduling Algorithm (Predicates “must” & Priority “Ranking”) and Node Condition influence Pod Placement (Node Binding)

- Note:
  - No Overcommit for “Requests” (Sum_Req <= Node Allocatable)
  - Priority Queue not active by default (alpha) -> critical Pods could be blocked after Host/Rack Failure
  - Limits not taken into Account currently (Placement might not be optimal)
    - Alpha Feature Gate (https://bit.ly/2qp3dgq)
  - Prior v1.10: DaemonSets are not scheduled by kube-scheduler (alpha in v1.10)

Scheduling of Pods with Requests and “Node Allocatable”

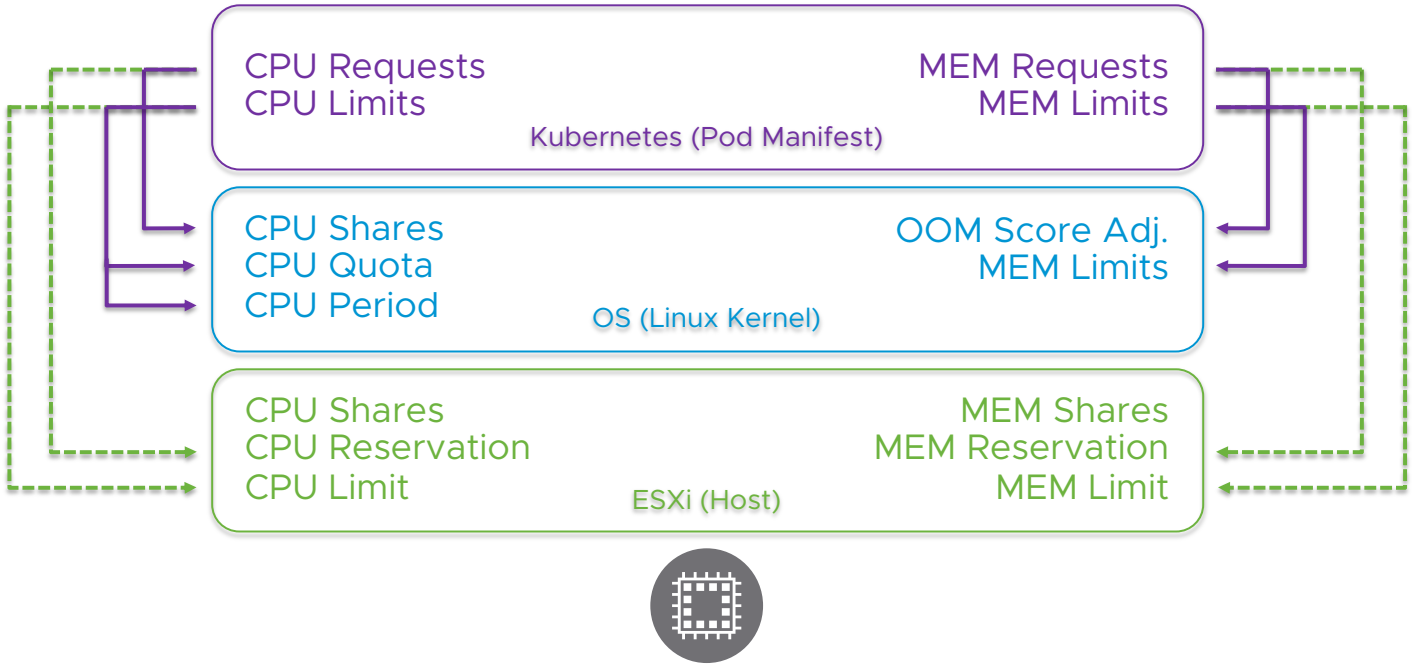
Enforcement

How QoS is enforced at the Node

Pod Creation

Cgroups are used to map Pod CPU and Memory Resources

- Note: Two Cgroups Drivers exist (cgroupfs [default], systemd)



Important
To be precise, Kubelet Heuristics for Cgroups Hierarchy Calculations are much more complex than depicted here (https://bit.ly/2Hwblgp).

Enforcement

How QoS is enforced at the Node

Pod Creation (Example)

```
$ kubectl run nginx --image=nginx --limits cpu=1,memory=200Mi --restart=Never
```

Kubelet View

```
$ cat /sys/fs/cgroup/cpu/kubepods/pod8d69938d-3e49-11e8-8690-000c29f521a3/cpu.{shares,cfs_*}
1024        # cpu.shares        (*relative* weight)
100000      # cpu.cfs_period_us (*absolute* enforcement interval in µs)
100000      # cpu.cfs_quota_us  (*absolute* limit in µs)

$ cat /sys/fs/cgroup/memory/kubepods/pod8d69938d-3e49-11e8-8690-000c29f521a3/memory.limit_in_bytes
209715200   # *absolute* memory limit in bytes
```

cpu=1

memory=200Mi

vmware

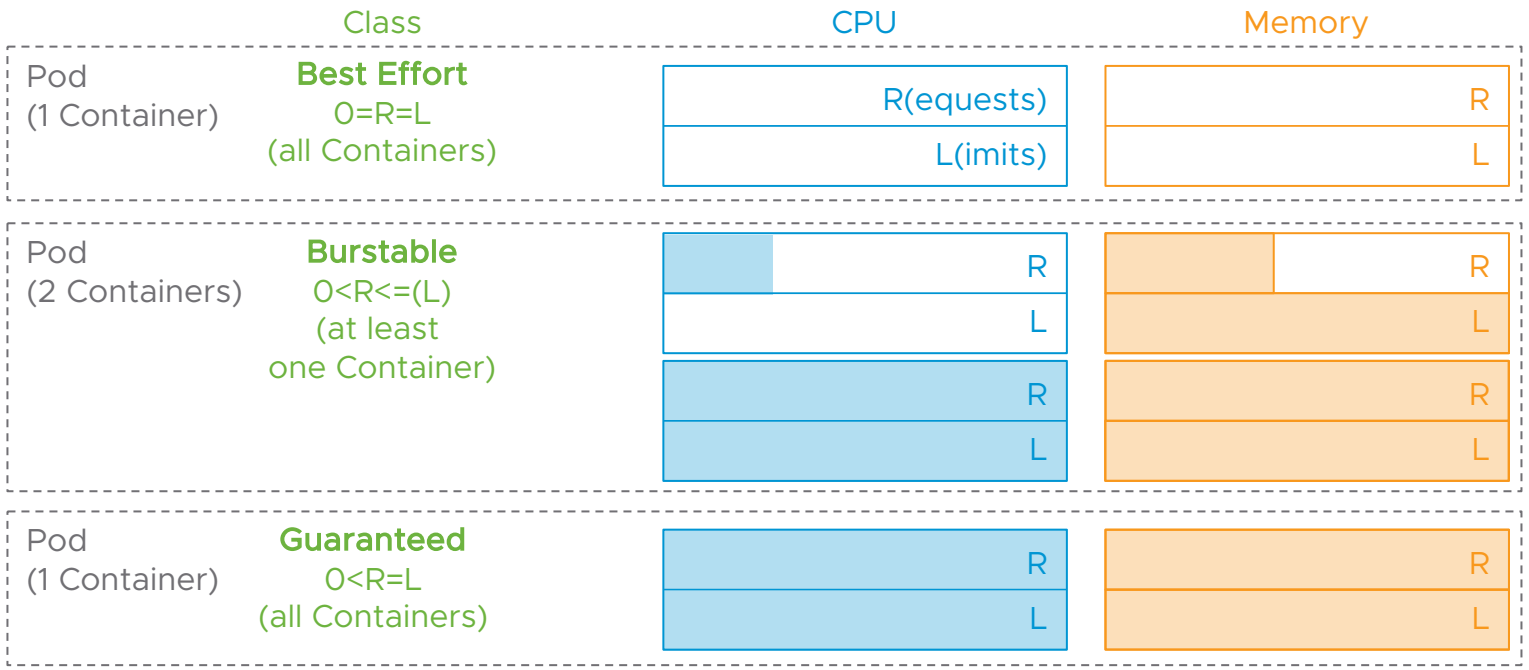
Enforcement

QoS Classes

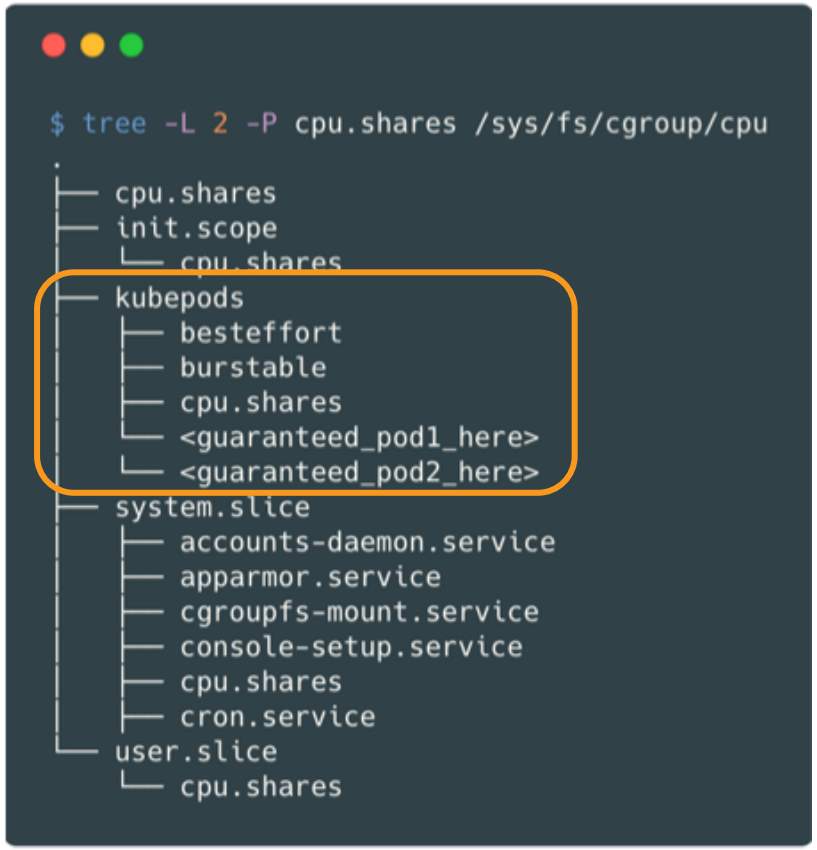
Implicit Definition from Pod Specification

Classes calculated based on CPU and Memory Resource Specifications (Requests/Limits)

- Details: http://bit.ly/2sO2KYX



QoS Examples



Kubelet View

vmware


Oversubscription anyone?

Enforcement

QoS Classes and Node Behavior

Response to “Kubelet Out of Resource” Conditions

Resources are either compressible (CPU) or uncompressible (Memory, Storage)

- Compressible = **Throttling** (Weight: cpu.shares)
- Uncompressible = **Evict** (Kubelet) or **OOM_kill** (“OutOfMemory Killer” by Kernel)

Kubelet Eviction Thresholds can be “hard” (instantly) and “soft” (allow Pod Termination Grace Period)

- Note:
  - If Kubelet cannot react fast enough, e.g. Memory Spike, Kernel OOM kills Container
    - There’s no Coordination between Eviction and OOM Killer (Race Condition possible)
  - Kubelet related File System Thresholds also trigger Eviction (after unsuccessful Reclamation)
  - Kubelet signals Pressure to API Server (honored by Scheduler)

Eviction Thresholds

Enforcement

QoS Classes and Node Behavior

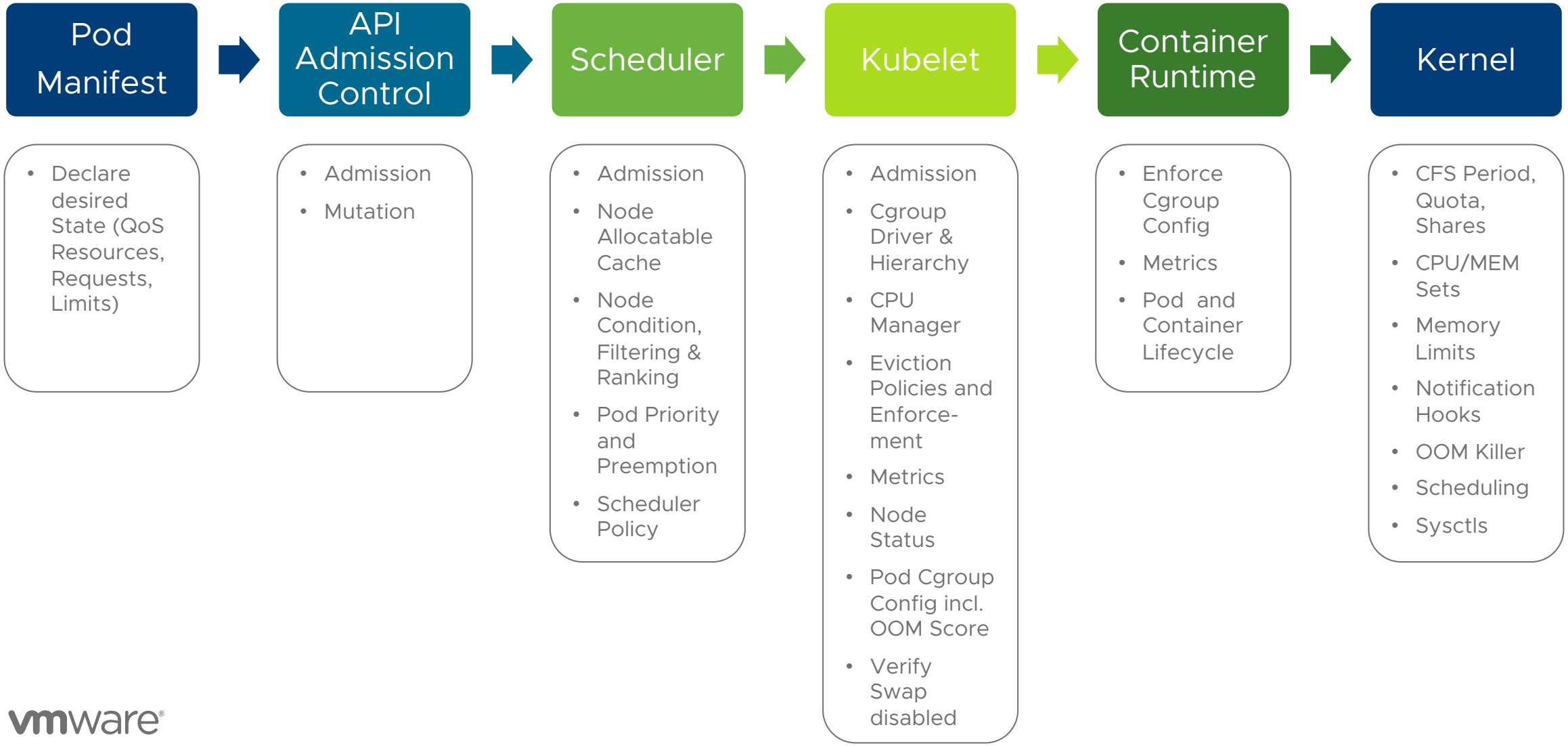
Eviction Order

Out of Resource Eviction Order (descending)

- Kubernetes before v1.9:
  - Largest Consumer relative to Request starting from QoS Best Effort -> Burstable -> Guaranteed
- Kubernetes v1.9 and above:
  - “Usage > Requests?” -> Pod Priority -> Usage - Requests
- Note:
  - Even “Guaranteed” Pods can be evicted
  - DaemonSets and other critical Pods are just Pods from the View of the Kubelet/OS (Pod Priority helps)
  - Details: https://bit.ly/2HuiG6k

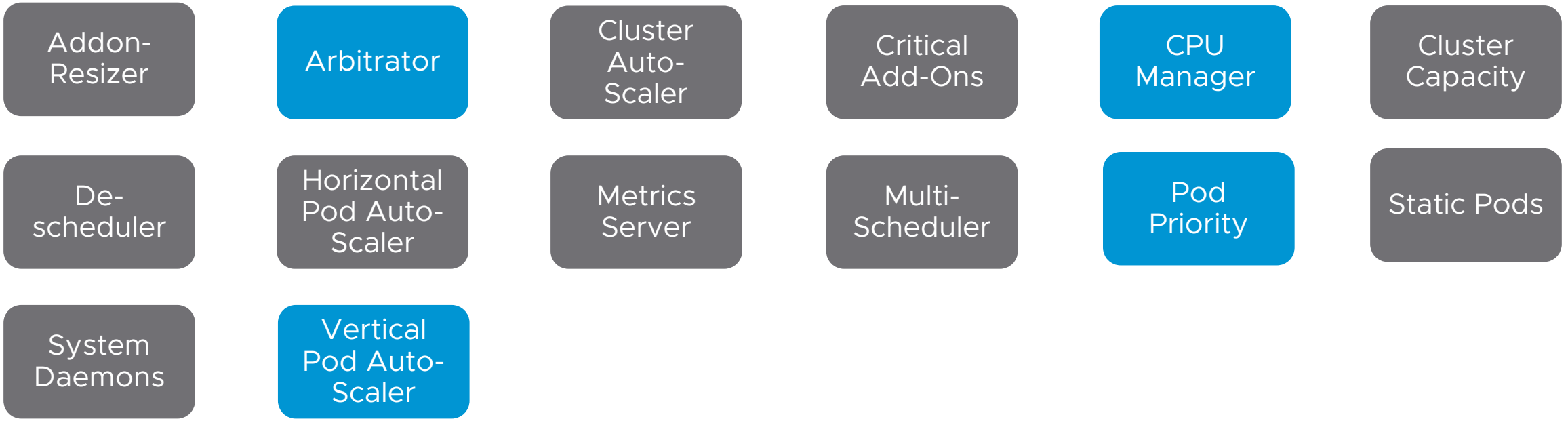
QoS Lifecycle, Admission and Enforcement

Details



But the Community wants more!

Kubernetes Resource QoS gets better (and more complex) with every Release



Best Practices from the Field

#1 Start using it!

If in doubt start with “guaranteed” QoS Class for your Workloads (i.e. no Overcommitment)

Enable Quotas and enforce sane Defaults (ResourceQuota, LimitRanger)

Protect critical (System) Pods (DaemonSets, Controllers, Master Components)

- Apply QoS Class “burstable” or “guaranteed” with sufficient Memory Requests (OOM Score Adj.)
- Reserve Node Resources with Labels/Selectors (if Scheduler Priority not active)
- PriorityClasses, hopefully Beta in v1.11, will significantly help

Embed QoS into your CI/CD Process

- Should be Part of all Stages
- Benchmarking/ Stress Tests for correct Values and/or consider VPA (Borg Autopilot) ☺

#2 It's only Part of the QoS Equation

Align Kubernetes QoS to underlying Infrastructure QoS (e.g. VM Reservations/Limits, Burstable Cloud Instances)

Monitor your Cluster and Resource Usage

- CPU and Memory, but also
  - Kernel Resources (Pids, Ports, Open File Handles, Sockets, etc.)
  - File Systems (Utilization, iNodes)
  - I/O (NET, Disk)
  - VM Instances (Cloud Provider Metrics)
  - Cgroups Statistics
  - Quotas
  - Apply RED/USE Method
- Typical Issues:
  - Long running Pods filling up File Systems (Log, temp)
  - Unbounded Pods (no Memory Limits) killing Critical Pods (OOM Score Adj.)

#3 Code and Language Runtime

Language Runtimes and Cgroups – The Issue

Many Language Runtimes, e.g. JRE, Go, .NET, etc. have no/limited Cgroups Awareness, i.e.

- Might see all Host CPUs
- Might see full Host Memory
- Behavior might be different between Language Runtime Versions

Leads to incorrect Tuning of:

- Heap Size, e.g. -Xmx in Java (http://bit.ly/2HG2C0V)
- Thread Pool Size, e.g. GOMAXPROCS in Go (http://bit.ly/2sKZcH6)
- GC Tuning, e.g. Size of Generation Spaces/ Number GC Threads in .NET (http://bit.ly/2GGzOEb)

Leads to

- Inefficient CPU/Memory Consumption
- Lower Performance
- Crashes

#3 Code and Language Runtime (continued)

Language Runtimes and Cgroups – Remediation

Check Language Spec and use latest Version (if possible)

- Java 10 made big Improvements for Container Support

Align Heap (+Overhead)/GC/Thread Parameters to Pod Resources

Could be done via

- Templating Engine
- Environment Variables
- ConfigMap
- Downward API (https://bit.ly/2qAn3pe)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downward-test
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox:1.24
      command: [ "/bin/sh", "-c", "env" ]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      env:
        - name: MY_CPU_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.cpu
        - name: MY_CPU_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.cpu
        - name: MY_MEM_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.memory
        - name: MY_MEM_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.memory
  restartPolicy: Never
```

Downward API Example

#4 Advanced Tuning

Fine-tune Kubelet (Node)

- --eviction-hard/soft (and related Values like Grace Periods)
- --fail-swap-on (default in recent Versions)
- --kube-reserved/--system-reserved for critical System and Kubernetes Services
- Notes:
  - NOT (!) intended for Pods/Workloads
  - Profile Service and OS Behavior before “enforcing” (optional), Risk of unintended OOMs

Use CPU Manager for Latency-critical Workloads (https://bit.ly/2vewWO6)

Use ”burstable” QoS w/out CPU Limit for Performance-critical Workloads

- Github Discussion: https://bit.ly/2qBc6Ui
- PriorityClasses, hopefully Beta in v1.11, will significantly help

#5 OS and Kernel

Make sure your Kernel has full Cgroups support compiled in and enabled

- Debian/Ubuntu https://dockr.ly/2H3ZEma

Windows is not Linux 😀 (http://bit.ly/2FpNDaR)

Disable Swap (required by Kubelet for proper QoS Calculation)

Always remember that you're running on a shared Kernel

- https://sysdig.com/blog/container-isolation-gone-wrong/
- https://hackernoon.com/another-reason-why-your-docker-containers-may-be-slow-d37207dec27f
- https://blog.hasura.io/debugging-tcp-socket-leak-in-a-kubernetes-cluster-99171d3e654b
- Mixing VM and Container-Level Isolation is powerful

Stay current with Releases (Kubernetes, Container Runtime, OS/Kernel)

- Changelogs and Design Docs (http://bit.ly/2CFY9HX) are your Friend

Thank You!

@embano1

vmware

Appendix

Session Abstract

Kubernetes Quality of Service (QoS) offers powerful primitives for resource management, that is workload prioritization, fairness and efficiency. But it's also a complex topic to understand and get right in production, e.g. if you are new to this topic or running highly dynamic and distributed systems.

Getting the most value out of a Kubernetes cluster requires utilizing the features provided to categorize and prioritize your workloads. We'll look at how Kubernetes provides this functionality through its QoS implementation. Both, from an end-user's perspective but also digging into the mechanics.

The work doesn't stop there though. This talk will also explore techniques on how to tune your application using best practices and lessons learned in the field. Finally, we'll provide community resources and an outlook about taking QoS to the next level in your cluster.

> Quality of service (QoS) is the description or measurement of the overall performance of a service, such as [...] a cloud computing service, particularly the performance seen by the users [...].
>
> Wikipedia

> In computing, scheduling is the method by which work specified by some means is assigned to (finite) resources that complete the work.
>
> Wikipedia

Resources

Tutorials

Configure QoS for Pods

- https://groups.google.com/forum/#!msg/kubernetes-sig-scheduling/kMG7yfONwY4/Nx3abXuNAAAJ

Configure Resource Quotas for a Namespace

- https://kubernetes.io/docs/tasks/administer-cluster/quota-memory-cpu-namespace/

Pod Priority and Preemption

- https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/

Reserve Compute Resources

- https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/#node-allocatable

Resources

Tutorials

Configure Out Of Resource Handling

- https://kubernetes.io/docs/tasks/administer-cluster/out-of-resource/#eviction-policy

Using Admission Controllers

- https://kubernetes.io/docs/admin/admission-controllers/

Kubelet Flags

- https://kubernetes.io/docs/reference/generated/kubelet/

LinuxCon 2016 Cgroups

- http://man7.org/conf/lceu2016/cgroups-LinuxCon.eu_2016-Kerrisk.pdf

Kernel Documentation Cgroups

- https://www.kernel.org/doc/Documentation/cgroup-v2.txt

Resources

Cgroups

LinuxCon 2016 Cgroups

- http://man7.org/conf/lceu2016/cgroups-LinuxCon.eu_2016-Kerrisk.pdf

Kernel Documentation Cgroups

- https://www.kernel.org/doc/Documentation/cgroup-v2.txt

LWN Cgroups Tutorial

- https://lwn.net/Articles/604609/

Red Hat CFS Guide

- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/sec-cpu

Resources

Design Docs

Kubernetes Resource Model

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/resources.md

Resource QoS in Kubernetes

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/resource-qos.md

Downward API

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/downward_api_resources_limits_requests.md

Implementing Resource Controls for Windows Containers and Docker Mappings

- https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/resource-controls

Resources

Design Docs

Kubelet Eviction Policy

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/kubelet-eviction.md

Priority in Resource Quota

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/pod-priority-resourcequota.md

Kubelet Pod Resource Management

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/pod-resource-management.md

Node Allocatable Resources

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/node-allocatable.md

Resources

Design Docs

Kubelet Disk Accounting

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/disk-accounting.md

Pod Preemption in Kubernetes

- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/pod-preemption.md

Resources

Talks

Tim Hockin – Everything you ever wanted to know about Resource Scheduling…almost

- https://www.youtube.com/watch?v=nWGkvrlPqJ4
- https://speakerdeck.com/thockin/everything-you-ever-wanted-to-know-about-resource-scheduling-dot-dot-dot-almost

Cluster Management at Google with Borg

- https://www.youtube.com/watch?v=0W49z8hVn0k&t=0s&index=38&list=WL

Local Ephemeral Storage Resource Management

- https://www.youtube.com/watch?v=cvK1t1h15XM
- https://schd.ws/hosted_files/kccncna17/3e/Kubecon_localstorage.pdf

Resources

Talks

Load Testing Kubernetes: How to Optimize Your Cluster Resource Allocation in Production

- https://www.youtube.com/watch?v=-lsJyni7EQA

Container Performance Analysis (Brendan Gregg)

- https://www.youtube.com/watch?v=bK9A5ODIgac

Resources

User Stories

https://engineering.linkedin.com/blog/2016/08/don_t-let-linux-control-groups-uncontrolled

https://engineering.linkedin.com/blog/2016/11/application-pauses-when-running-jvm-inside-linux-control-groups

https://circleci.com/blog/how-to-handle-java-oom-errors/

https://mesosphere.com/blog/java-container/

https://blog.markvincze.com/troubleshooting-high-memory-usage-with-asp-net-core-on-kubernetes/

- https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals

https://very-serio.us/2017/12/05/running-jvms-in-kubernetes/

QoS Classes

A.k.a. Priorities in Case a Node runs out of Resources (Oversubscription)

Guaranteed

+ Predictable SLA and highest Priority (Eviction)
- Lower Efficiency (Resources capped, no Overcommit)

Burstable

+ Increase Overcommit Level, use idle Resources*
- Medium Priority (Eviction), unbounded Resources*

Best Effort

+ High Resource Efficiency & Utilization
- Resource Starvation and Eviction very likely

*

When no Limits set

vmware

Enforcement

How QoS is enforced at the Node

Details

Resource	Compressible*	Node Overcommit allowed	Out of Resource Handling
cpu	Yes	Yes	Throttle
memory	No	Yes	Evict or OOM_kill
hugepages-<size>	No	No	n/a
ephemeral-storage	No	Yes	Evict
Extended Resources	No	No	n/a
Device Plugins	No	No	n/a

*

Compressible “yes” = Throttling
Compressible “no” = OOM Killer and/or Eviction

Resource Management at the OS Level