

OPTIMIZING RESOURCE USAGE IN KUBERNETES



Carlos Sanchez / csanchez.org / [@csanchez](https://twitter.com/csanchez)

Principal Scientist

Adobe Experience Manager Cloud Service

Author of Jenkins Kubernetes plugin

Long time OSS contributor at Jenkins, Apache Maven,
Puppet,...

ADOBE EXPERIENCE MANAGER

Content Management System

Digital Asset Management

Digital Enrollment and Forms

Used by many Fortune 100 companies

An existing distributed Java OSGi application
Using OSS components from Apache Software
Foundation

A huge market of extension developers

AEM ON KUBERNETES

AEM ON KUBERNETES

Running on Azure

35+ clusters and growing

Multiple regions: US, Europe, Australia, Singapore,
Japan, India, more coming

AEM ON KUBERNETES

Adobe has a dedicated team managing clusters for multiple products

Customers can run their own code

Cluster permissions are limited for security

AEM ON KUBERNETES

Using namespaces to provide a scope

- network isolation
- quotas
- permissions

AEM ENVIRONMENTS

- Customers can have multiple AEM environments that they can self-serve
- Each customer: 3+ Kubernetes namespaces (dev, stage, prod environments)
- Each environment is a micro-monolith TM

ENVIRONMENTS

Using init containers and (many) sidecars to apply
division of concerns



SCALING AND RESOURCE OPTIMIZATION

Each customer environment (17k+) is a micro-monolith TM

Multiple teams building services

Need ways to scale that are orthogonal to the dev teams

Kubernetes workloads can define resource requests
and limits:

Requests:

how many resources are guaranteed

Limits:

how many resources can be consumed

And are applied to

CPU

Memory

Ephemeral storage

CPU: may result in CPU throttling

Memory: limit enforced, results in Kernel OOM killed

Ephemeral storage: limit enforced, results in pod
eviction

JAVA AND KUBERNETES

QUIZZ

Assume:

- Java 11+ and latest releases
- 4GB memory available
- 2+ CPUs available

WHAT IS THE DEFAULT JVM HEAP SIZE?

1. 75% of container memory
2. 75% of host memory
3. 25% of container memory
4. 25% of host memory
5. 127MB

It depends

WHAT IS THE DEFAULT JVM HEAP SIZE?

1. **75% of container memory** (< 256 MB)
2. 75% of host memory
3. **25% of container memory** (> 512 MB)
4. 25% of host memory
5. **127MB** (256 MB to 512 MB)

JDKs >8u191 and >11 will detect the available memory
in the container, not the host

Using the container memory limits, so there is no
guarantee that physical memory is available

Do not trust JVM ergonomics

Configure memory with

- `-XX:InitialRAMPercentage`
- `-XX:MaxRAMPercentage`
- ~~`-XX:MinRAMPercentage`~~ (allows setting the maximum heap size for a JVM running with less than 200MB)

Typically can use up to 75% of container memory

Unless there is a lot of off-heap memory used
(ElasticSearch, Spark,...)

JVM takes all the memory on startup and manages it

JVM memory use is hidden from Kubernetes, which
sees all of it as used

Set request and limits to the same value

WHAT IS THE DEFAULT JVM GARBAGE COLLECTOR?

1. SerialGC
2. ParallelGC
3. G1GC
4. ZGC
5. ShenandoahGC

It depends

WHAT IS THE DEFAULT JVM GARBAGE COLLECTOR?

1. **SerialGC** *<2 processors & < 1792MB available*
2. **ParallelGC** *Java 8*
3. **G1GC** *Java >=11*
4. **ZGC**
5. **ShenandoahGC**

Poorly tuned GC will cause pauses and other issues

Do not trust JVM ergonomics

Configure GC with

- `-XX:+UseSerialGC`
- `-XX:+UseParallelGC`
- `-XX:+UseG1GC`
- `-XX:+UseZGC`
- `-XX:+UseShenandoahGC`

Garbage Collectors

Recommendations

	Serial	Parallel	G1	Z	Shenandoah
Number of cores	1	2+	2+	2+	2+
Multi-threaded	No	Yes	Yes	Yes	Yes
Java Heap size	<4GBytes	<4Gbytes	>4GBytes	>4GBytes	>4GBytes
Pause	Yes	Yes	Yes	Yes (<1ms)	Yes (<10ms)
Overhead	Minimal	Minimal	Moderate	Moderate	Moderate
Tail-latency Effect	High	High	High	Low	Moderate
JDK version	All	All	JDK 8+	JDK 17+	JDK 11+
Best for	Single core, small heaps	Multi-core small heaps. Batch jobs, with any heap size.	Responsive in medium to large heaps (request-response/DB interactions)	responsive in medium to large heaps (request-response/DB interactions)	responsive in medium to large heaps (request-response/DB interactions)

HOW MANY CPUS DOES THE JVM THINK ARE AVAILABLE?

1. Same as the host
2. Same as the k8s container cpu requests
3. Same as the k8s container cpu limits

It depends

HOW MANY CPUS DOES THE JVM THINK ARE AVAILABLE?

1. Same as the host Java 19+ / 17.0.5+ / 11.0.17+ / 8u351+
2. Same as the k8s container cpu requests
3. Same as the k8s container cpu limits (otherwise, sort of)

Before Java 19/17.0.5/11.0.17/8u351

- 0 ... 1023 = 1 CPU
- 1024 = (no limit)
- 2048 = 2 CPUs
- 4096 = 4 CPUs

JDK-8281181 Do not use CPU Shares to compute active processor count

the JDK interprets `cpu.shares` as an absolute number that limits how many CPUs the current process can use

Kubernetes sets `cpu.shares` from the CPU requests

Do not trust JVM ergonomics

Configure cpus with

- `-XX:ActiveProcessorCount`

**IN A HOST WITH 32 CPUS, AND 2 JVMS
WITH EACH 8 CPU REQUESTS/16 CPU
LIMIT, HOW MUCH CPU CAN EACH USE IF
BOTH ARE AS BUSY AS POSSIBLE?**

1. 100%
2. 75%
3. 50%
4. 25%

**IN A HOST WITH 32 CPUS, AND 2 JVMS
WITH EACH 8 CPU REQUESTS/16 CPU
LIMIT, HOW MUCH CPU CAN EACH USE IF
BOTH ARE AS BUSY AS POSSIBLE?**

1. 100%
2. 75%
3. 50%
4. 25%

**IN A HOST WITH 32 CPUS, AND 2 JVMS
WITH EACH 8 CPU REQUESTS AND NO
LIMITS, HOW MUCH CPU CAN ONE USE IF
THE OTHER ONE IS NOT USING CPU?**

1. 100%
2. 75%
3. 50%
4. 25%

**IN A HOST WITH 32 CPUS, AND 2 JVMS
WITH EACH 8 CPU REQUESTS AND NO
LIMITS, HOW MUCH CPU CAN ONE USE IF
THE OTHER ONE IS NOT USING CPU?**

1. 100%
2. 75%
3. 50%
4. 25%

CPU REQUESTS IN KUBERNETES

It is used for scheduling and then a relative weight

It is not the number of CPUs that can be used

CPU REQUESTS IN KUBERNETES

1 CPU means it can consume one CPU cycle per CPU period

Two containers with 0.1 cpu requests each can use 50% of the CPU time of the node

CPU LIMITS IN KUBERNETES

This translates to cgroups quota and period.

Period is by default 100ms

The limit is the number of CPU cycles that can be used
in that period

After they are used the container is throttled

CPU LIMITS IN KUBERNETES

Example

500m in Kubernetes -> 50ms of CPU usage in each
100ms period

1000m in Kubernetes -> 100ms of CPU usage in each
100ms period

CPU LIMITS IN KUBERNETES

This is challenging for Java and multiple threads

For 1000m in Kubernetes and 4 threads

you can consume all the CPU time in 25ms and be
throttled for 75 ms

+-----+
| Core 1 |
+-----+
+-----+
| Core 2 |
+-----+
+-----+
| Core 3 |
+-----+
+-----+
| Core 4 |
+-----+

+-----+
| Thread 1 |
+-----+

+-----+
| Thread 1 |
+-----+

<----->

<-----

Period 100 ms



KUBERNETES CLUSTER AUTOSCALER

Automatically increase and reduce the cluster size

KUBERNETES CLUSTER AUTOSCALER

Based on CPU/memory requests

Some head room for spikes

Multiple scale sets in different availability zones

At scale it is easy to cause a Denial of Service in our
services or cloud services

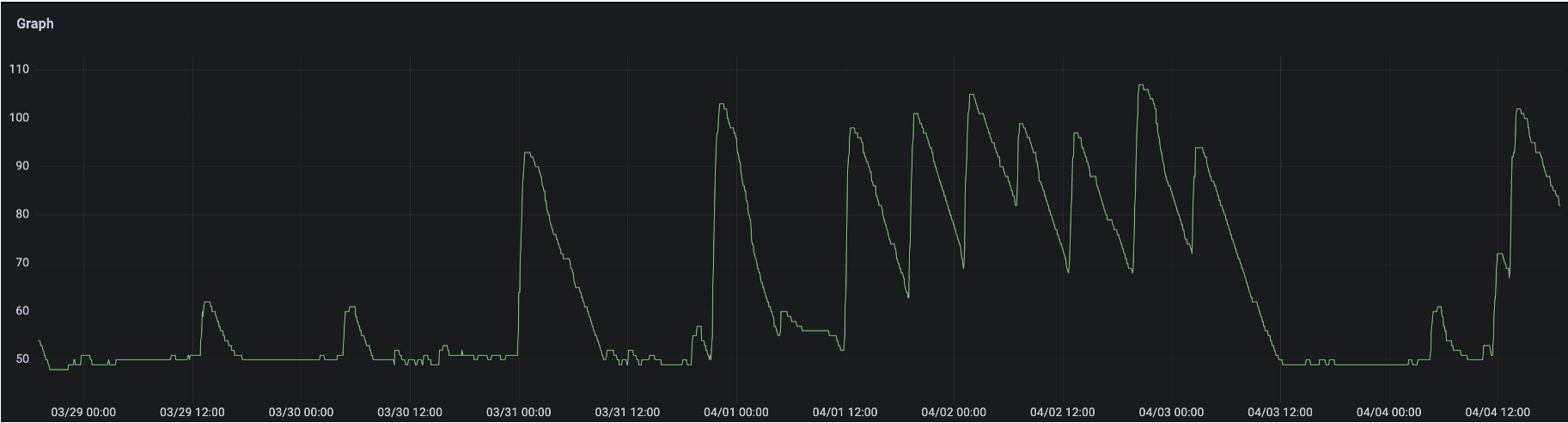
KUBERNETES CLUSTER AUTOSCALER

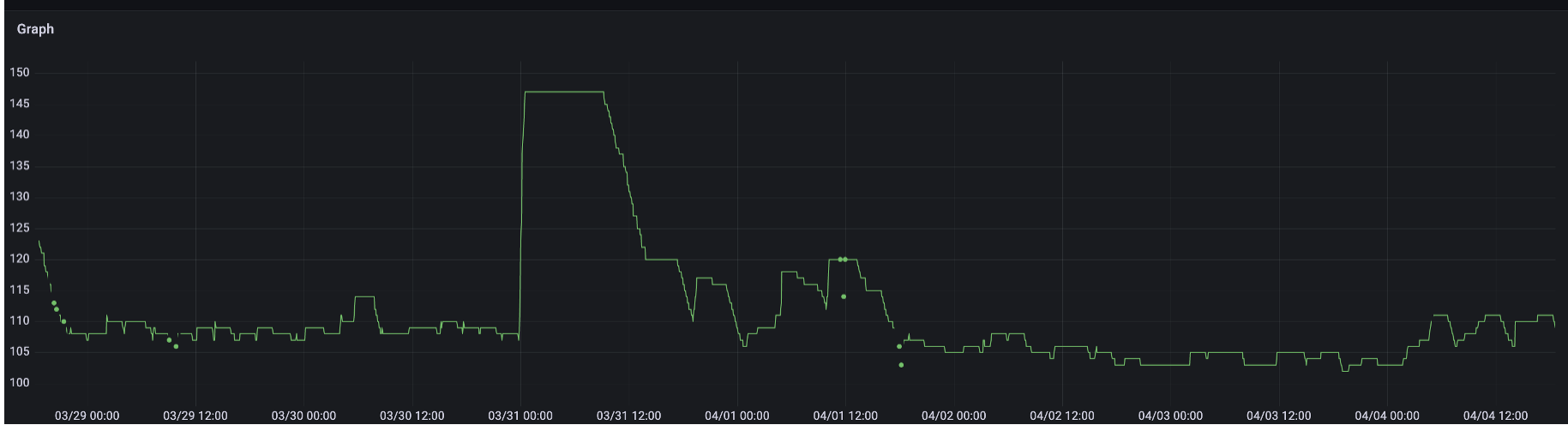
Max nodes managed at the cluster level

Least waste Scaling Strategy

Selects the node group with the least idle CPU after
scaleup

Savings: 30-50%





VERTICAL POD AUTOSCALER

Increasing/decreasing the resources for each pod


VPA

Allows scaling resources up and down for a deployment

Requires restart of pods (automatic or on next start)

(next versions of Kubernetes will avoid it)

Makes it slow to respond, can exhaust resources in busy nodes

 Do not set VPA to auto if you don't want random pod restart

VPA

Only used in AEM dev environments to scale down if unused

And only for some containers

JVM footprint is hard to reduce

Savings: 5-15%

HORIZONTAL POD AUTOSCALER

Creating more pods when needed

HPA

AEM scales on CPU and http requests per minute (rpm) metrics

 Do not use same metrics as VPA

CPU autoscaling is problematic

Periodic tasks can spike the CPU, more pods do not help

Spikes on startup can trigger a cascading effect

HPA

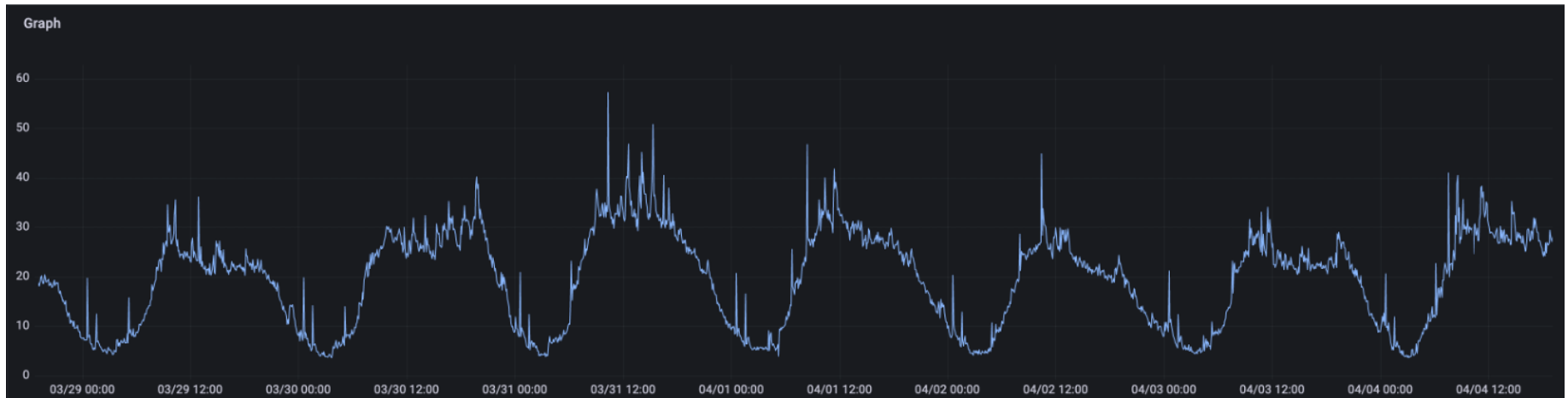
AEM needs to be warmed up on startup

rpm autoscaling is better suited

As long as customers don't have expensive requests

Savings: 50-75%

Pods vs RPM



VPA VS HPA

Increasing the memory and CPU is more effective than adding more replicas

But you are going to need multiple replicas for HA

RESOURCE OPTIMIZATION IN KUBERNETES

From the JVM: tuning CPU, memory, GC

From Kubernetes ecosystem: Cluster autoscaler, HPA, VPA

At application and infrastructure levels

A combination of them will help you optimize and reduce resources used

csanchez.org

