LESSONS LEARNED MIGRATING AN EXISTING PRODUCT TO A MULTI TENANT CLOUD NATIVE ENVIRONMENT

Adobe

Natalia Angulo / github.com/angulito / @nangulito

Carlos Sanchez / csanchez.org / @csanchez

Natalia / Software Developer Engineer

Maths, coding

Carlos / Principal Scientist

Author of Jenkins Kubernetes plugin
Long time OSS contributor at Jenkins, Apache Maven, Puppet,...

Adobe Experience Manager Cloud Service

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

Contributing and building on top of ASF projects

Sling

Felix

Maven

...

AEM ON KUBERNETES

Running on Azure

37+ clusters and growing

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

Adobe has a dedicated team managing clusters for multiple products

Customers can run their own code

Cluster permissions are limited for security

ie. Traffic leaving the clusters must be encrypted

AEM ENVIRONMENTS

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

Using namespaces to provide a scope

- network isolation
- quotas
- permissions

SERVICES

Multiple teams building services

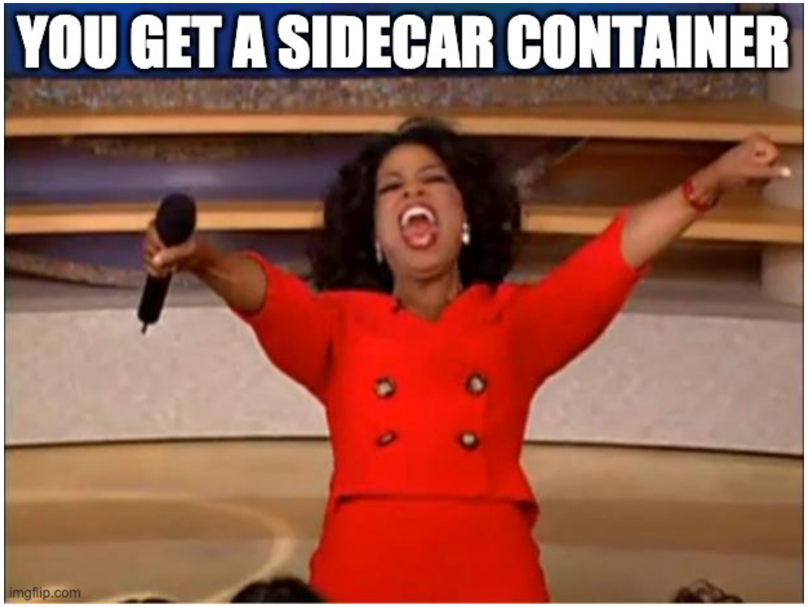
Different requirements, different languages

You build it you run it

Using APIs or Kubernetes operator patterns

ENVIRONMENTS

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



SIDECARS

- Service warmup
- Storage initialization
- httpd fronting the Java app
- Exporting metrics
- fluent-bit to send logs
- Java threaddump collection
- Envoy proxying
- Autoupdater

SERVICE WARMUP

Ensure that the service is ready to serve traffic

Probes the most requested paths for lazy caching

Without requiring expensive starts

EXPORTING METRICS

Export system level metrics such as

- disk size
- disk space
- network reachability

FLUENT-BIT TO SEND LOGS

Using a shared volume to send logs to a central location

Configured independently from the application

JAVA THREADDUMP COLLECTION

Gets the threaddumps generated by the JVM

and uploads them to a shared location

ENVOY PROXYING

Using Envoy for traffic tunneling and routing

Enables dedicated ips per tenant and VPN connectivity

Can be used as a load balancer and reverse proxy with many features: rate limiting, circuit breaking, retries, etc.

AUTOUPDATER

Runs on startup and updates any configuration needed

Allows patching the whole cluster fleet live

OPERATORS

AEM ENVIRONMENT OPERATOR

Overarching operator that manages lifecycle of environments

AEM ENVIRONMENT OPERATOR

An operator to rule them all

Launches jobs pre/post environment creation

Reconciles with other internal operators

Provides an API to other systems to manage environments

FLUXCD HELM OPERATOR

https://fluxcd.io/

FLUXCD HELM OPERATOR

Allows managing Helm charts using declarative state vs imperative commands

Integrated with our operators to manage the lifecycle of the Helm releases

and to gather state from the Helm operations

ARGO ROLLOUTS OPERATOR

https://argoproj.github.io/rollouts/

ARGO ROLLOUTS OPERATOR

Provides advanced deployment strategies

Canary, Blue/Green, A/B testing, etc.

Automated rollbacks

KUBERNETES SECURITY PROFILES OPERATOR

https://github.com/kubernetes-sigs/security-profiles-operator

KUBERNETES SECURITY PROFILES OPERATOR

Allows to create and manage security profiles for pods

Security profiles define capabilities that a pod can use

Can integrate with Seccomp, SELinux, AppArmor

Can be installed, recorded and distributed as OCI images

SCALING AND RESOURCE OPTIMIZATION

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

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

ARM ARCHITECTURE

15-25% cost savings for the same performance

Easy switch for containerized Java

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

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

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 WILL THE JVM BE ABLE TO USE?

1. Same as the k8s container cpu requests
2. Same as the k8s container cpu limits
3. As many as the OS allows

HOW MANY CPUS WILL THE JVM BE ABLE TO USE?

1. Same as the k8s container cpu requests
2. **Same as the k8s container cpu limits** <17.0.5 / <11.0.17 / <8u351
3. **As many as the OS allows** Java 19+ / 17.0.5+ / 11.0.17+ / 8u351+

Before Java 19/17.0.5/11.0.17/8u351

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

Number of CPUs / active processor count is used to compute the number of threads in the JVM

used when creating threads for various subsystems.

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 32 CPU HOST WITH 2 JVMS WHERE ONE IS IDLE, EACH WITH 8 CPU REQUESTS/16 CPU LIMIT, WHAT IS THE MAX CPU USED?

1. 8
2. 16
3. 32

IN A 32 CPU HOST WITH 2 JVMS WHERE ONE IS IDLE, EACH WITH 8 CPU REQUESTS/16 CPU LIMIT, WHAT IS THE MAX CPU USED?

1. 8
2. 16
3. 32

IN A 32 CPU HOST WITH 2 JVMS WHERE ONE IS IDLE, EACH WITH 8 CPU REQUESTS/NO LIMITS, WHAT IS THE MAX CPU USED?

1. 8
2. 16
3. 32

IN A 32 CPU HOST WITH 2 JVMS WHERE ONE IS IDLE, EACH WITH 8 CPU REQUESTS/NO LIMITS, WHAT IS THE MAX CPU USED?

1. 8
2. 16
3. **32**

CPU REQUESTS AND LIMITS IN KUBERNETES

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

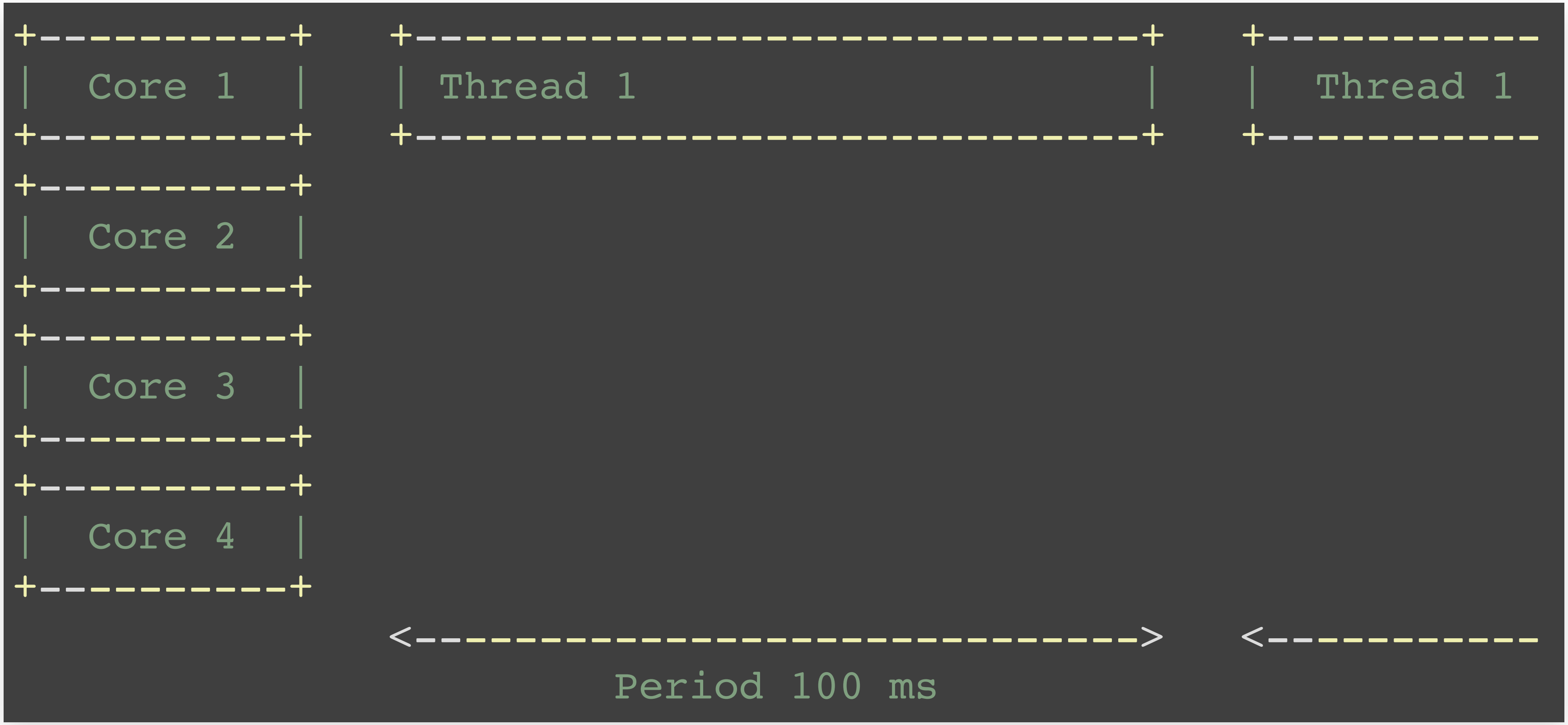
CPU LIMITS IN KUBERNETES

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



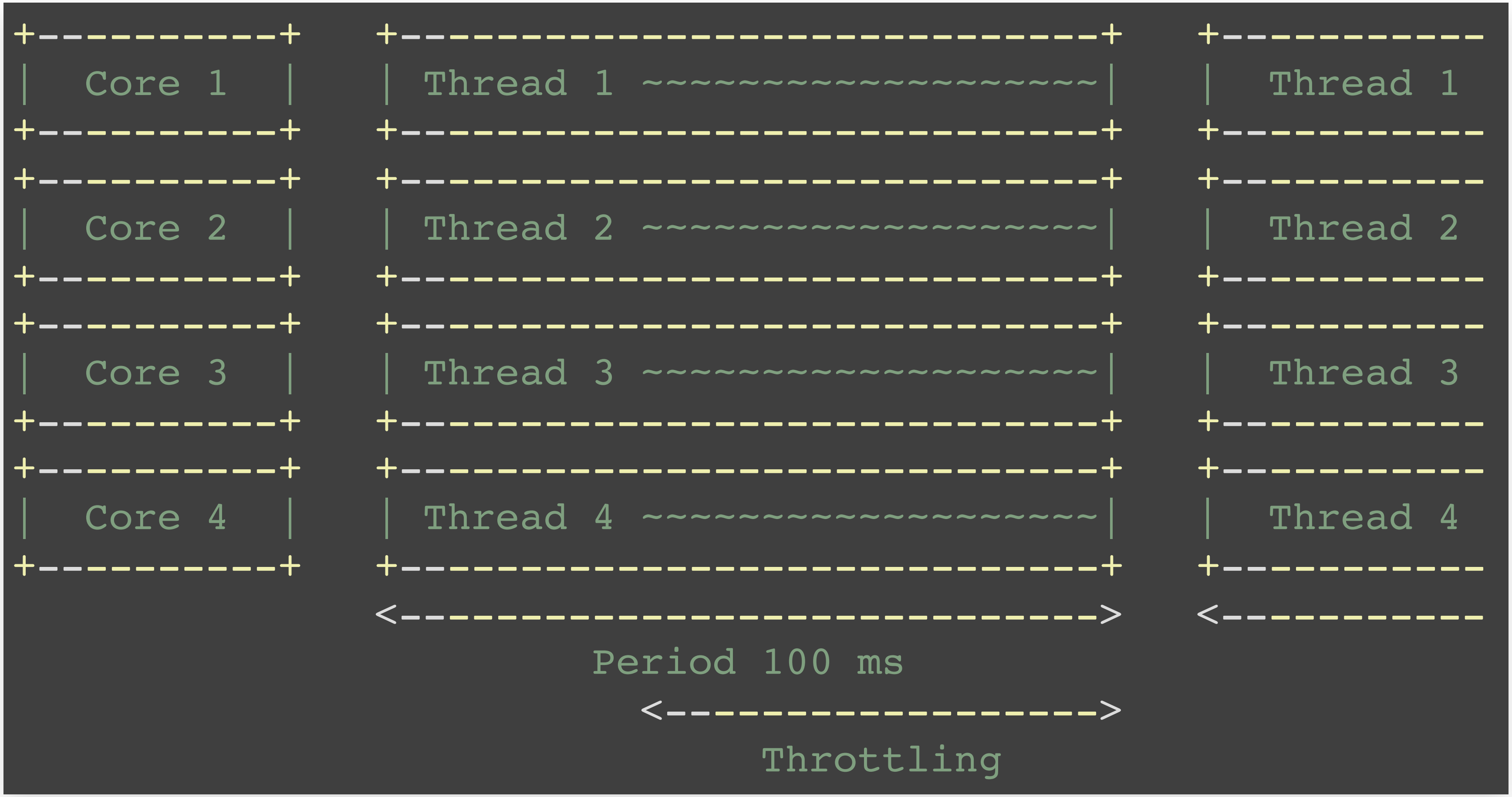
```
+----------+   +-----------------------------+   +-----------
|  Core 1  |   | Thread 1                    |   |  Thread 1
+----------+   +-----------------------------+   +-----------
+----------+
|  Core 2  |
+----------+
+----------+
|  Core 3  |
+----------+
+----------+
|  Core 4  |
+----------+
               <----------------------------->   <-----------
                        Period 100 ms
```

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  |   | Thread 1 ~~~~~~~~~~~~~~~~~~~|   |  Thread 1
+----------+   +-----------------------------+   +-----------
+----------+   +-----------------------------+   +-----------
|  Core 2  |   | Thread 2 ~~~~~~~~~~~~~~~~~~~|   |  Thread 2
+----------+   +-----------------------------+   +-----------
+----------+   +-----------------------------+   +-----------
|  Core 3  |   | Thread 3 ~~~~~~~~~~~~~~~~~~~|   |  Thread 3
+----------+   +-----------------------------+   +-----------
+----------+   +-----------------------------+   +-----------
|  Core 4  |   | Thread 4 ~~~~~~~~~~~~~~~~~~~|   |  Thread 4
+----------+   +-----------------------------+   +-----------
               <----------------------------->   <-----------
                        Period 100 ms
                          <------------------>
                               Throttling
```

KUBERNETES AUTOSCALING

KUBERNETES AUTOSCALING

- Cluster Autoscaler
- Horizontal Pod Autoscaler
- Vertical Pod Autoscaler

KUBERNETES CLUSTER AUTOSCALER

Automatically increase and reduce the cluster size

KUBERNETES CLUSTER AUTOSCALER

Based on CPU/memory requests

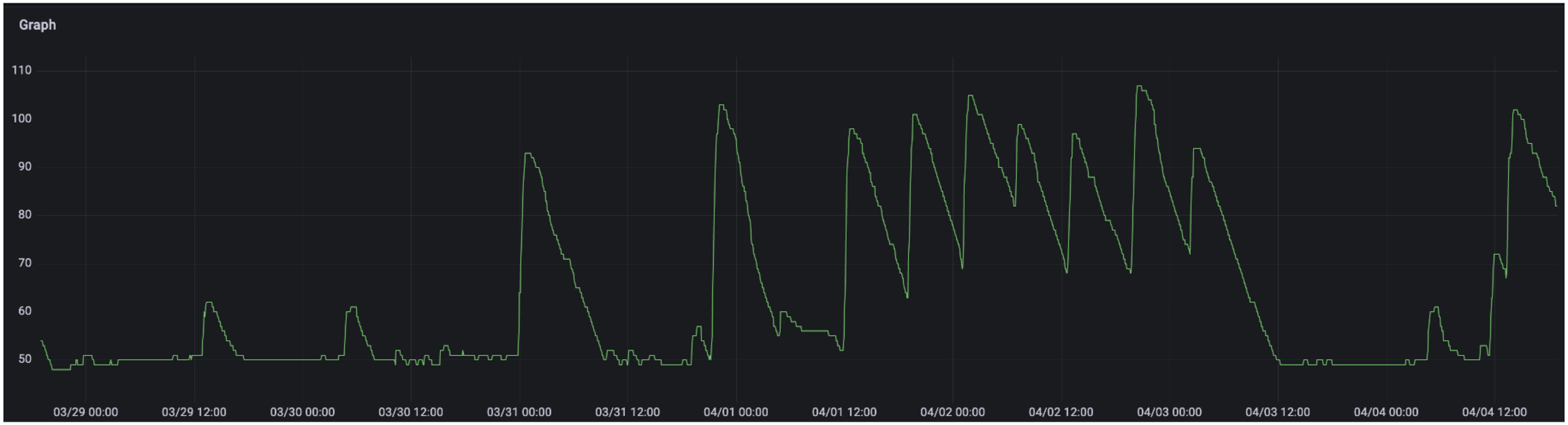
Some head room for spikes

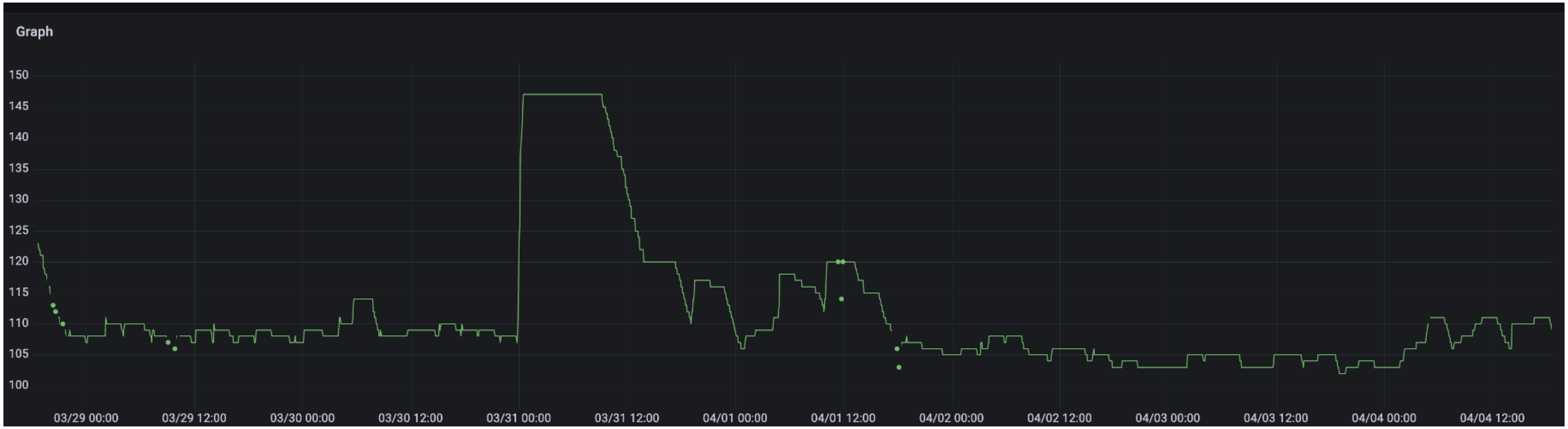
Multiple scale sets in different availability zones

KUBERNETES CLUSTER AUTOSCALER

Max nodes managed at the cluster level

Savings: 30-50%


Graph
110
100
90
80
70
60
50
03/29 00:00
03/29 12:00
03/30 00:00
03/30 12:00
03/31 00:00
03/31 12:00
04/01 00:00
04/01 12:00
04/02 00:00
04/02 12:00
04/03 00:00
04/03 12:00
04/04 00:00
04/04 12:00


Graph
150
145
140
135
130
125
120
115
110
105
100
03/29 00:00
03/29 12:00
03/30 00:00
03/30 12:00
03/31 00:00
03/31 12:00
04/01 00:00
04/01 12:00
04/02 00:00
04/02 12:00
04/03 00:00
04/03 12:00
04/04 00:00
04/04 12:00

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

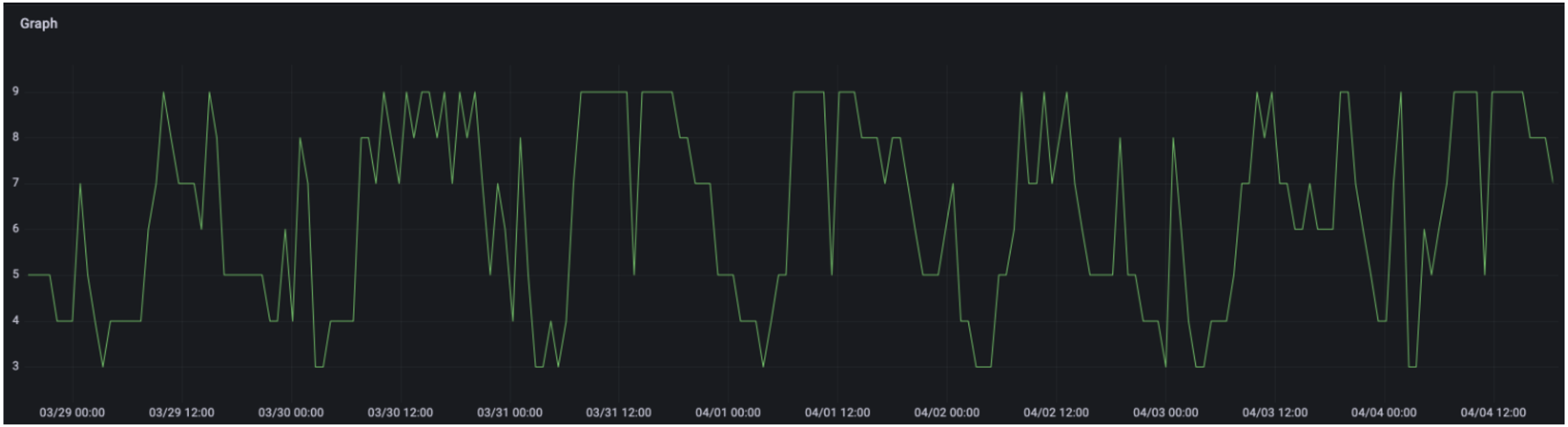
HPA

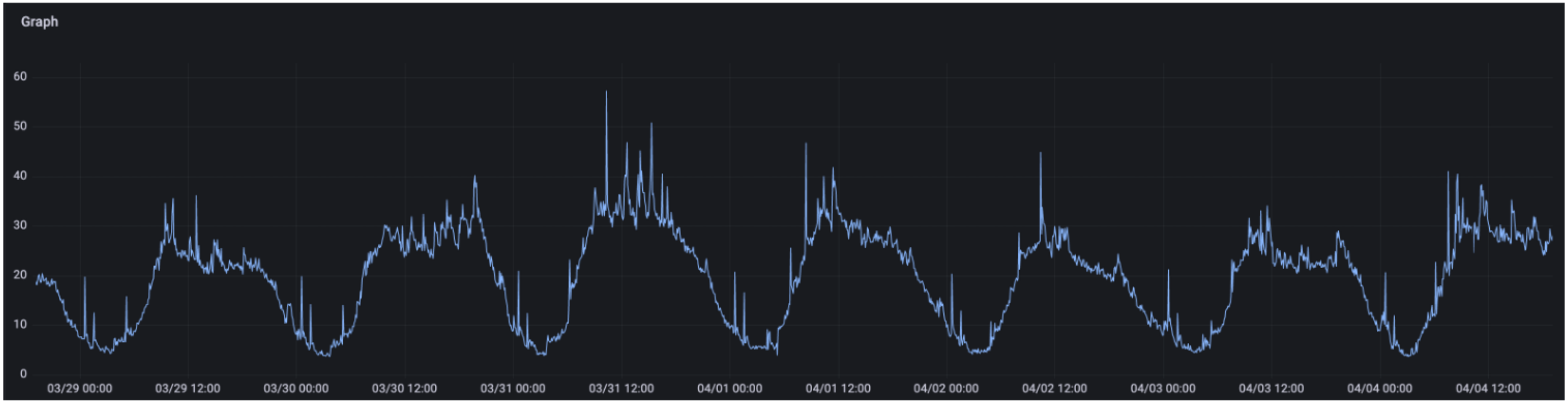
AEM needs to be warmed up on startup

rpm autoscaling is better suited

Savings: 50-75%

Pods vs RPM


Graph
9
8
7
6
5
4
3
03/29 00:00
03/29 12:00
03/30 00:00
03/30 12:00
03/31 00:00
03/31 12:00
04/01 00:00
04/01 12:00
04/02 00:00
04/02 12:00
04/03 00:00
04/03 12:00
04/04 00:00
04/04 12:00


Graph
60
50
40
30
20
10
0
03/29 00:00
03/29 12:00
03/30 00:00
03/30 12:00
03/31 00:00
03/31 12:00
04/01 00:00
04/01 12:00
04/02 00:00
04/02 12:00
04/03 00:00
04/03 12:00
04/04 00:00
04/04 12:00

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

Easy to start in k8s, then optimize

Use patterns to decompose application: sidecars, init containers, new services,...

Resource optimization: tuning JVM CPU, memory, GC nangulito

angulito

csanchez

carlossg

csanchez.org

Adobe