# WE MOVED ONE JAVA PRODUCT TO KUBERNETES AND THIS IS WHAT WE LEARNED

Adobe

Carlos Sanchez / csanchez.org / @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

Running on Azure

35+ 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

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 (dev, stage, prod environments)
- Each environment is a micro-monolith ™
- Sandboxes, evaluation-like environments

Customers interact through Cloud Manager, a separate service with web UI and API

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

## SIDECARS

- Custom developed (threaddump collector, storage initialization)
- OSS (fluent-bit)
- Extended from OSS (httpd)

## 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

# SCALING AND RESOURCE OPTIMIZATION

Each customer environment (1000s) 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

## WHAT IS THE DEFAULT JVM HEAP SIZE?

It depends

## 25% of detected memory

unless old Java 8 patch versions

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?

It depends

Java 11 or later: SerialGC or G1GC

Java 8: SerialGC or ParallelGC

Serial if <2 processors and < 1792MB memory available

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.<br><br>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?

It depends

## Java 11, 17, 18 older than October 2022 release

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

In Java 19 and backported to 11, 17, 18 in the October 2022 release

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

In versions 18.0.2+, 17.0.5+ and 11.0.17+, OpenJDK will no longer take CPU shares settings into account for its calculation of available CPU cores.

Do not trust JVM ergonomics

Configure cpus with

- `-XX:ActiveProcessorCount`

## 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

## CPU LIMITS IN KUBERNETES

Do NOT set CPU limits for your production workloads

You would be wasting unused CPU cycles

Limits are still interesting for dev/stage to guarantee you are not using more CPU than you should csanchez.org

csanchez

carlossg

Adobe