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

Adobe

Carlos Sanchez / csanchez.org / @csanchez

Cloud Engineer

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

25+ 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

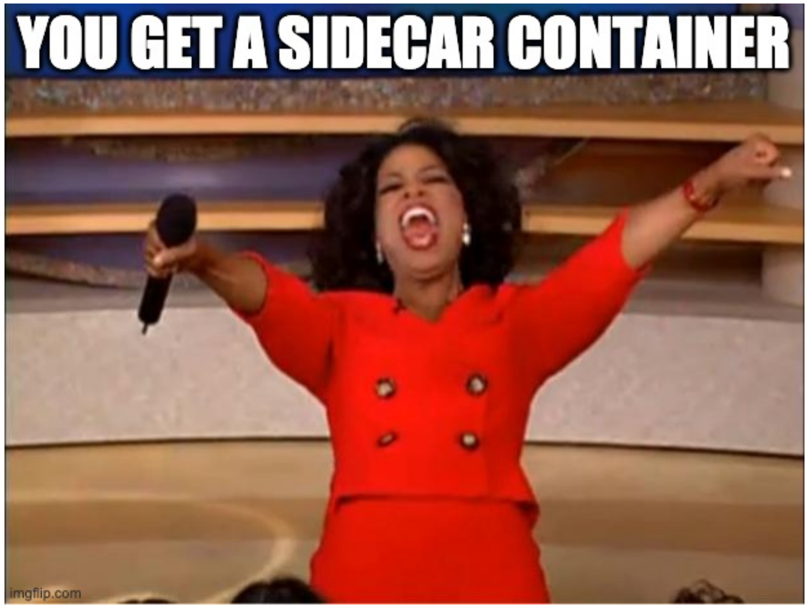
Different requirements, different languages

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

100s of customers

1000s of sandboxes

Each customer environment is a micro-monolith ™

Multiple teams building services

Need ways to scale that are orthogonal to the dev teams

Kubernetes workloads must set 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

JVM takes all the memory on startup and manages it

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

JDKs >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

Configured through

- `-XX:InitialRAMPercentage`
- `-XX:MaxRAMPercentage`
- `-XX:MinRAMPercentage`

JDK recently changed the way cores are computed

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

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

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.

SCALING

- ⚠️ API rate limits can be hit on upgrades, so we limit each cluster in the hundreds of nodes
- You could have bigger nodes too

Using Kubernetes Vertical and Horizontal 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

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

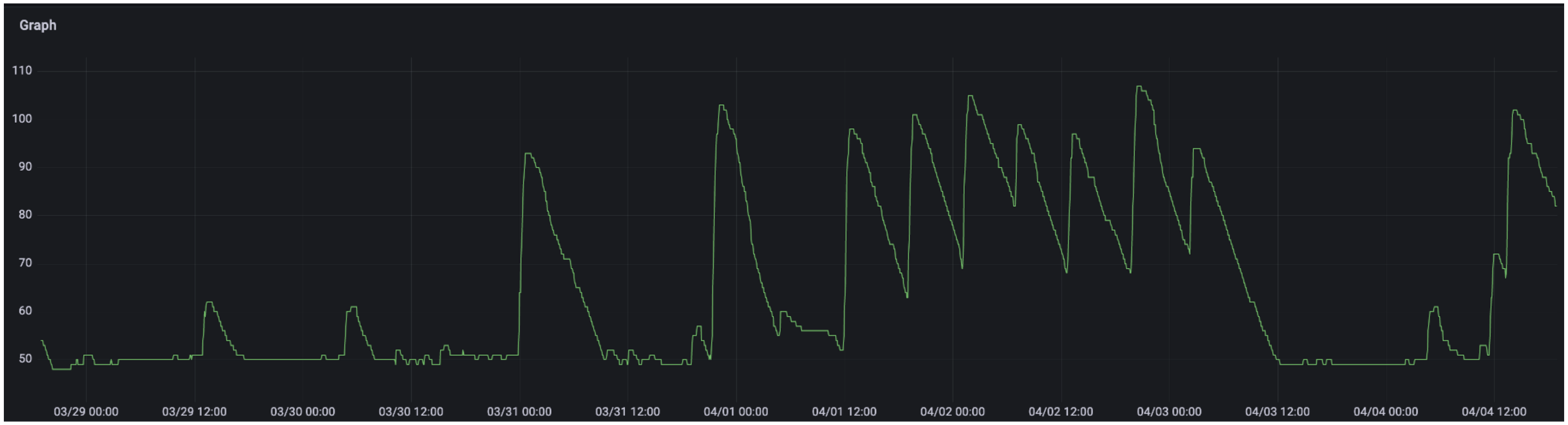
KUBERNETES CLUSTER AUTOSCALER

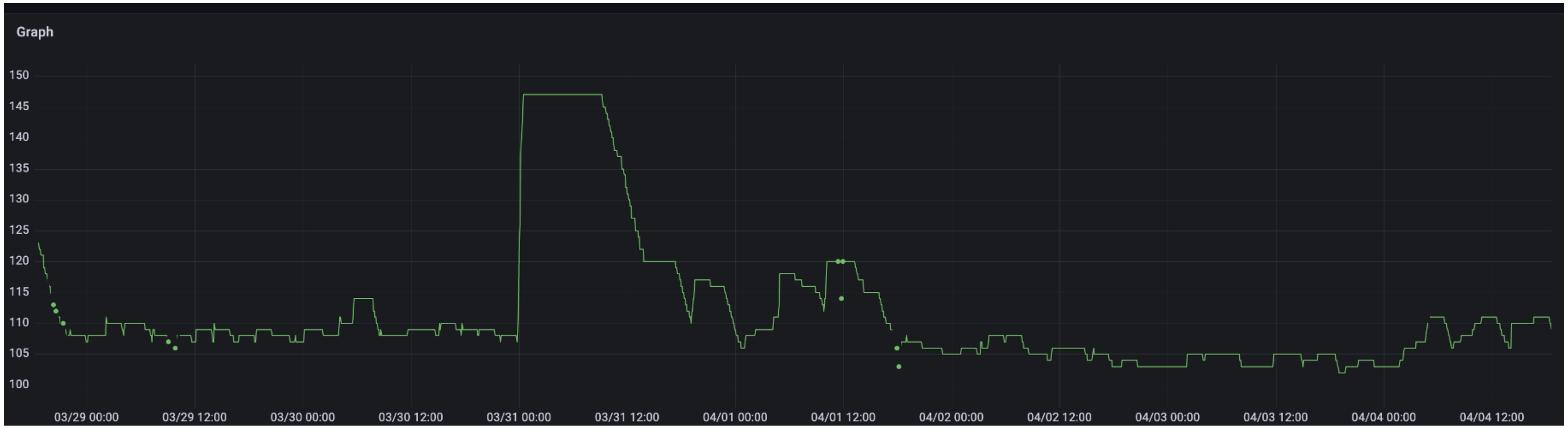
Max nodes managed at the cluster level

Least waste Scaling Strategy

Selects the node group with the least idle CPU after scaleup

Savings: 30-50%


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

VPA

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)

(there is a PR in Kubernetes to 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%

HPA

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

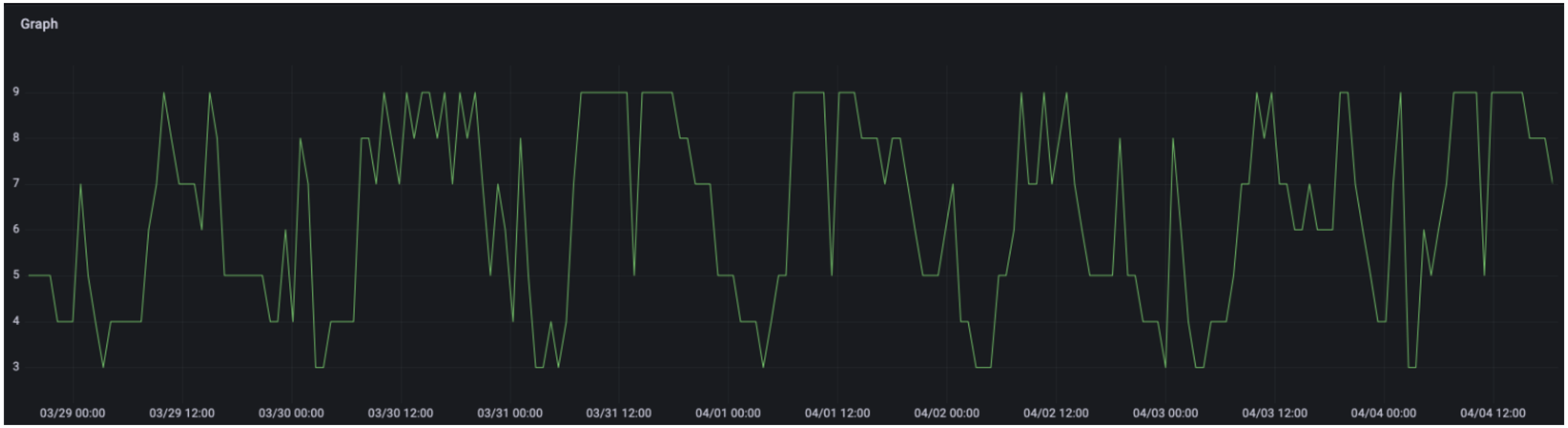
AEM needs to be warmed up on startup

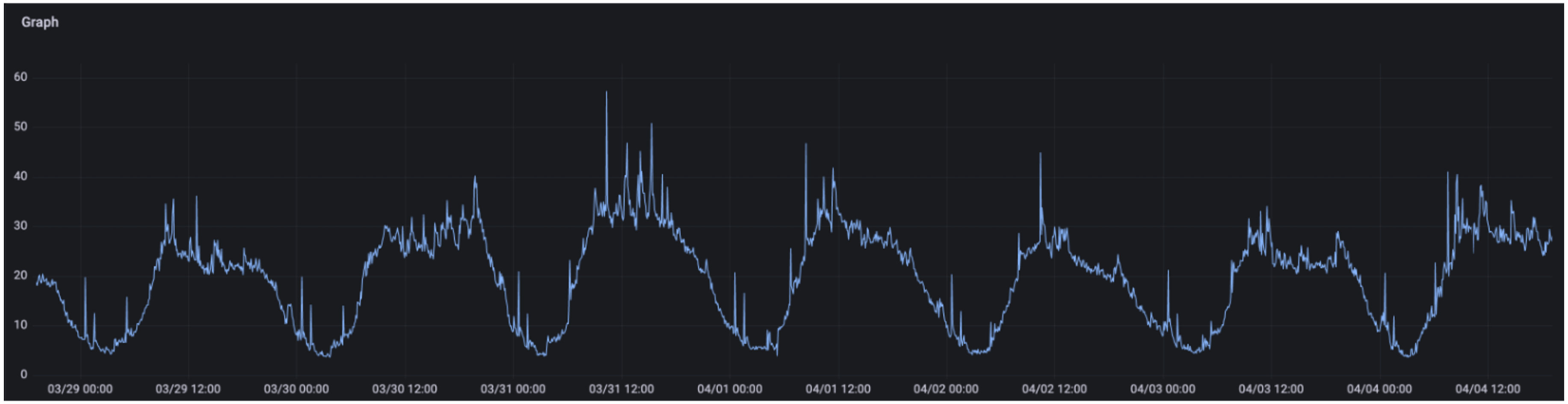
rpm autoscaling is better suited

As long as customers don’t have expensive requests

Savings: 50-75%

Pods vs RPM


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

Secrets of Performance Tuning Java on Kubernetes

Bruno Borges

ABSTRACT

Java on Kubernetes may seem complicated, but after a bit of YAML and Dockerfiles, you will wonder what all that fuss was. But then the performance of your app in 1 CPU/1 GB of RAM makes you wonder. Learn how JVM ergonomics, CPU throttling, and GCs can help increase performance while reducing costs.

DAY & TIME

Thursday 10:20 - 60 min

ROOM

Room 5

ETCD LIMITS

Stored objects capacity (secrets/configmaps) is not infinite

Every mounted object will create a watch

Number of watches cause increased memory usage

Slower response time can lead to API slowness and timeouts

That will prevent reaction to changes in the cluster, ie. pod movements

ETCD LIMITS

Better to create less objects with more data than too many objects (50k+)

Use immutable secrets/configmaps to avoid watches if not needed

Don't use etcd as a database!

INGRESS CONTROLLER LIMITS

Ingress Controller uses Envoy proxy

Envoy is an edge and service proxy, designed for cloud-native applications.

Hundreds of ingresses can make Envoy reprogramming slower

Can cause traffic go to old pods no longer running

HIBERNATION

Scaling to zero environments not used

Scaling down multiple deployments associated to one “AEM environment”

Deleting ingress routes and other objects that may limit cluster scale

HIBERNATION

- Kubernetes job checks Prometheus metrics
- If no activity in *n* hours, scale deployment to 0
- Customer is shown a message to de-hibernate by clicking a button

HIBERNATION

Ideally it would de-hibernate automatically on new request, more like Function as a Service

but JVM takes ~5 min to start

Savings: 60-80%

AUTOMATIC RESOURCE CONFIGURATION

Service built internally at Adobe to help with resource consumption

AUTOMATIC RESOURCE CONFIGURATION

In most clusters services request more cpu/memory than used

ARC can transparently reduce cpu/memory requirements

Limits are not affected, so side effects are limited, would not trigger OOM Killer (likely)

ARC RECOMMENDER

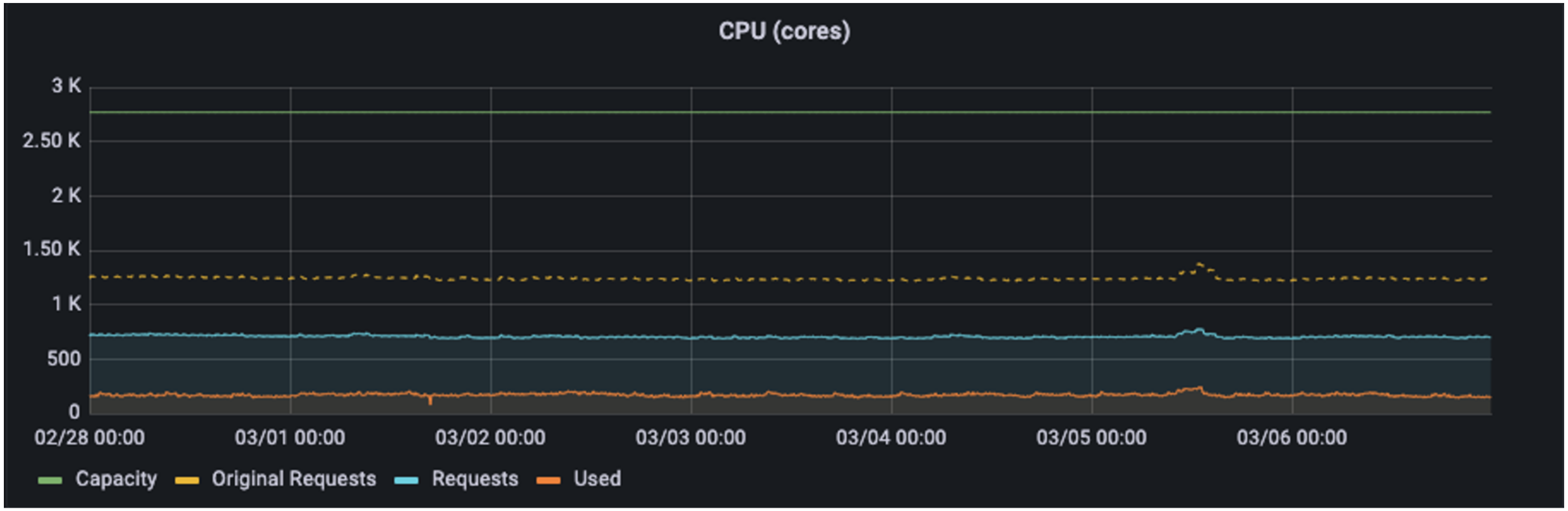
ARC recommender leverages historical metrics at the deployment level

Can provide recommendations about optimization at deployment level based on actual usage

ARC CLUSTER RATIOS

ARC can dial down resource requests at cluster/namespace level

Savings: 10-15%



CPU (cores)
3 K
2.50 K
2 K
1.50 K
1 K
500
0
02/28 00:00
03/01 00:00
03/02 00:00
03/03 00:00
03/04 00:00
03/05 00:00
03/06 00:00
Capacity
Original Requests
Requests
Used

Why ARC and not VPA recommender?

Full control over recommendation algorithm

Implementation at more global cluster level with deployment level recommendations

NETWORKING

NETWORKING

Kubernetes networking is complex

Multitenancy is even more

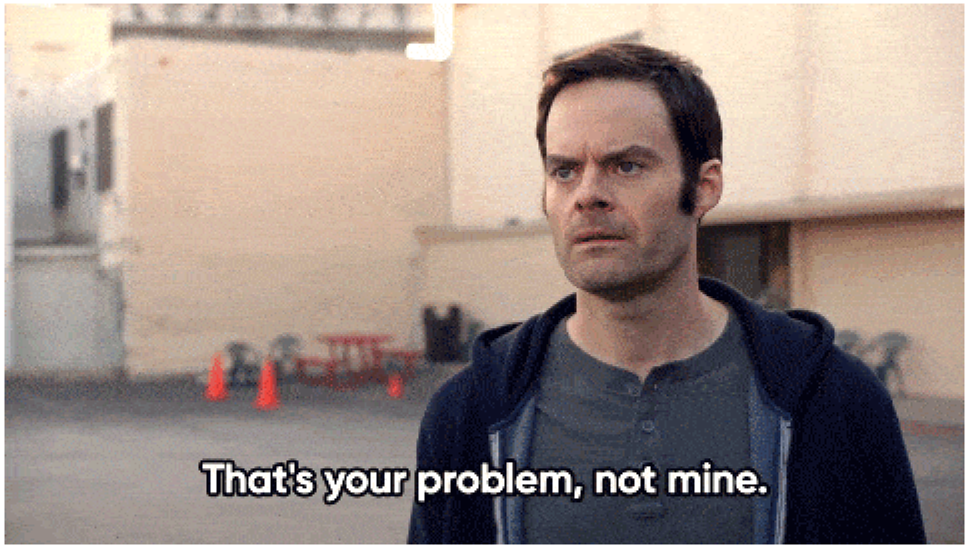
Services cannot connect to other namespaces

Everything blocked by default, open on each service case by case

NETWORKING

Everything is virtual

- Allows flexibility
- Introduces complexity



NETWORKING: CILIUM

- eBPF instead of iptables
- More efficient and performant
- Custom network policies at level 7 (path, header, method,...)

NETWORKING: CILIUM

`NetworkPolicy` to block/allow traffic

- Block access to other namespaces
- Allow outgoing https and other common ports

Customers may also want to allow specific ingress ips only, ie. for dev/stage

NETWORKING: INGRESS

Custom ingress controller

- With more features than standard Kubernetes `Ingress` object
- blocklist/allowlist, path based routing,...
- Uses Envoy proxy behind the scenes

NETWORKING: ENVOY

- ⚠️ Missconfigurations can cause cluster wide issues
- ⚠️ Restarting it when config is wrong will clear all routes
- ⚠️ Locks when the rate of changes is too high

NETWORKING: ENVOY

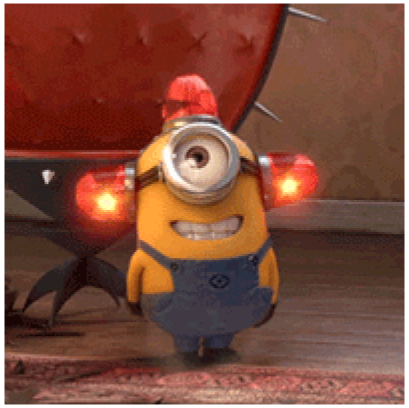
We had to do work to fix issues and use it correctly

ie. Validation of all configs both at build and runtime

LOGGING

- Using `fluent-bit` sidecars to send logs to centralized store
- Grafana `loki` for log aggregation

MONITORING AND ALERTING



- Multiple Prometheus and Grafana
- Aggregating all clusters data
- Alerts coming from Prometheus AlertManager

CUSTOMER LOGGING

Customers also need access to some logs

- `fluent-bit` sends logs to `loki` service
- Customer can view them in Cloud Manager
- ⚠️ `logstash` is heavy, 2GB+ memory needed, `loki` a better option

RESILIENCY AND SELF HEALING

- Readiness and liveness probes so services are marked unavailable and restarted automatically
- `PodDisruptionBudget` to ensure a number of replicas on rollouts and cluster upgrades
- `topologySpreadConstraints` to distribute service across nodes and availability zones

MULTITENANCY

Limit blast radius

Customers are namespace isolated

⚠️ All deployments **must** have CPU/memory requests and limits

RUNNING CUSTOMER CODE

Pods that run customer code are a higher risk

Started testing Kata Containers, pod runs in a VM transparently

Contributing improvements upstream

SECURITY PROFILES OPERATOR

Using SELinux, seccomp and AppArmor in Kubernetes

Allows recording, auditing and blocking system calls

EXTERNAL SERVICES

PERSISTENCE

- External MongoDB
- Azure Blob Storage

PERSISTENCE

Kubernetes Persistent Volumes are not very scalable

- ⚠️ In the past hit API rate limitting at 100s of Persistent Volumes
- It is supposed to have improved

DATA PROCESSING

Using Kafka based service to sync data between author and publish

Works worldwide, when publishers are in multiple regions

CDN: FASTLY

- Fastly in front of the Load Balancer
- Binary content stored in Azure blobs

EGRESS CONNECTIVITY EXTENSIONS

Dedicated ip requested by some customers for firewall configuration

Or to avoid being throttled/blocked by other tenants

VPN connections

EGRESS DEDICATED IP

Scale set of Envoy proxies dedicated per customer

Dedicated egress load balancer that sets the outgoing ip

Using Envoy to tunnel from sidecar to outgoing load balancer

Short lived certificates for mTLS tunnels

PRIVATE CONNECTIONS

Customers can connect through VPN to their private networks

Using transparently Azure VPN Gateway at the end of the Envoy mTLS tunnels

CONTINUOUS DELIVERY

AEM: From yearly to daily release

Using Jenkins for CI/CD

Tekton and other services to orchestrate pipelines

GITOPS

Most configuration is stored in git and reconciled on each commit

Pull vs Push model to scale

KUBERNETES DEPLOYMENTS

Combination of

- Helm: AEM application
- Plain Kubernetes files: ops services
- Kustomize: some new microservices

HELM

⚠️ Don't mix application and infra in the same package

Helm operator to pull in each namespace

PROGRESSIVE DELIVERY

Rollout to different customer groups in separate waves

Rollout to a percentage of customers

Global Helm overrides using Helm values or Kustomize patches through the Helm operator

SHIFT LEFT

Detect problems as soon as possible, not in production

Running checks on PRs so developers can act on them asap

Generating the gitops and helm templates and running multiple tests

KUBECTL APPLY DRY RUN

The most basic check

```
kubectl apply --dry-run=server
```

KUBECONFORM

`Kubeconform` to validate Kubernetes schemas

Succesor of `Kubeval`

We added schema validation of some custom CRDs

CONFTEST

`Conftest` to validate policies while allowing developer autonomy

- security recommendations
- labelling standards
- image sources

Using Rego language 🥺

PLUTO

`Pluto` to detect API versions deprecated or removed csanchez.org

csanchez

carlossg

Adobe