

THE CLOUD NATIVE JOURNEY AT



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

Cloud Engineer

Adobe Experience Manager Cloud Service

Author of Jenkins Kubernetes plugin

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

ADOBE EXPERIENCE MANAGER

DISCLAIMER

This applies to my team inside Adobe Experience Manager Cloud Service

There are many teams in AEM

There are maaany teams in Adobe

Content Management System

Digital Asset Management

Digital Enrollment and Forms

Used by many Fortune 100 companies

An existing distributed Java application

Author instances for authors

Publish instances for web visitors

Both can scale horizontally

STACK

Java using OSGi for modules

Using OSS components from Apache Software Foundation

A huge market of extension developers

Writing modules that run in-process on AEM

AEM ON KUBERNETES

Running on Azure

10+ clusters and growing

Multiple regions: US, Europe, Australia, more coming

Adobe has a dedicated team managing clusters for multiple products

AEM ENVIRONMENTS

- Customers can have multiple AEM environments that they can self-serve
- Each customer: 3+ Kubernetes namespaces (dev, stage, prod environments)
- Sandboxes, evaluation-like environments

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

ENVIRONMENTS

Namespaces provide a scope

- network isolation
- quotas
- permissions

ENVIRONMENTS

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



SIDECARS

- Storage initialization
- httpd fronting the Java app
- Exporting metrics
- fluent-bit to send logs
- Java threaddump collection

SIDECARS


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

SCALING

100s of customers

1000s of sandboxes

SCALING

-  Azure API rate limits can be hit on upgrades, so we limit each cluster to a few hundred nodes
- You could have bigger nodes too

Using Kubernetes Vertical and Horizontal Pod Autoscaler

SCALING: VPA

We use VPA to scale up/down on memory and CPU

JVM footprint is hard to reduce

Changes to requests need pod restarts to become effective

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

SCALING: HPA

HPA scales up on requests/minute

 Do not use same metrics as VPA

SCALING: HIBERNATION



For engineering environments and sandboxes that are seldomly used

Allows overbooking clusters and \$\$\$ savings

SCALING: 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

Ideally it would de-hibernate automatically on new request, more like Function as a Service, but JVM takes ~5 min to start

NETWORKING



I'M NETWORKING!

NETWORKING

Kubernetes networking is complex

Multitenancy is even more

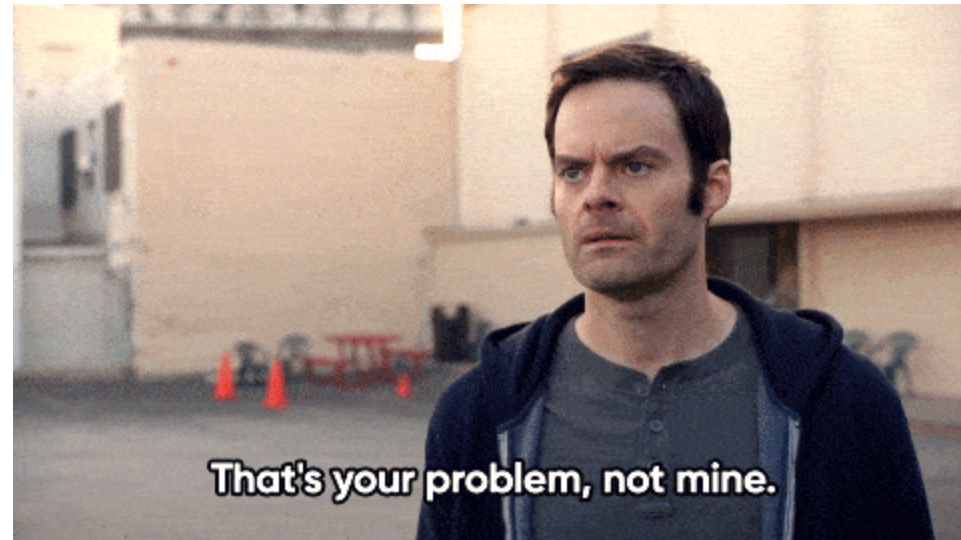
Services cannot connect to other namespaces

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

Using a Contour fork

- 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 `logstash/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
- [podAntiAffinity](#) 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


EXTERNAL SERVICES

PERSISTENCE

- External MongoDB
- Azure Blob Storage

PERSISTENCE

Kubernetes Persistent Volumes are a bit risky in Azure

-  Can get the cluster in a bad state due to Azure API rate limiting at 100s of Persistent Volumes
- It is supposed to improve with new versions of the Azure Kubernetes cluster controller

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 DEDICATED IP

Dedicated ip requested by some customers for firewall configuration

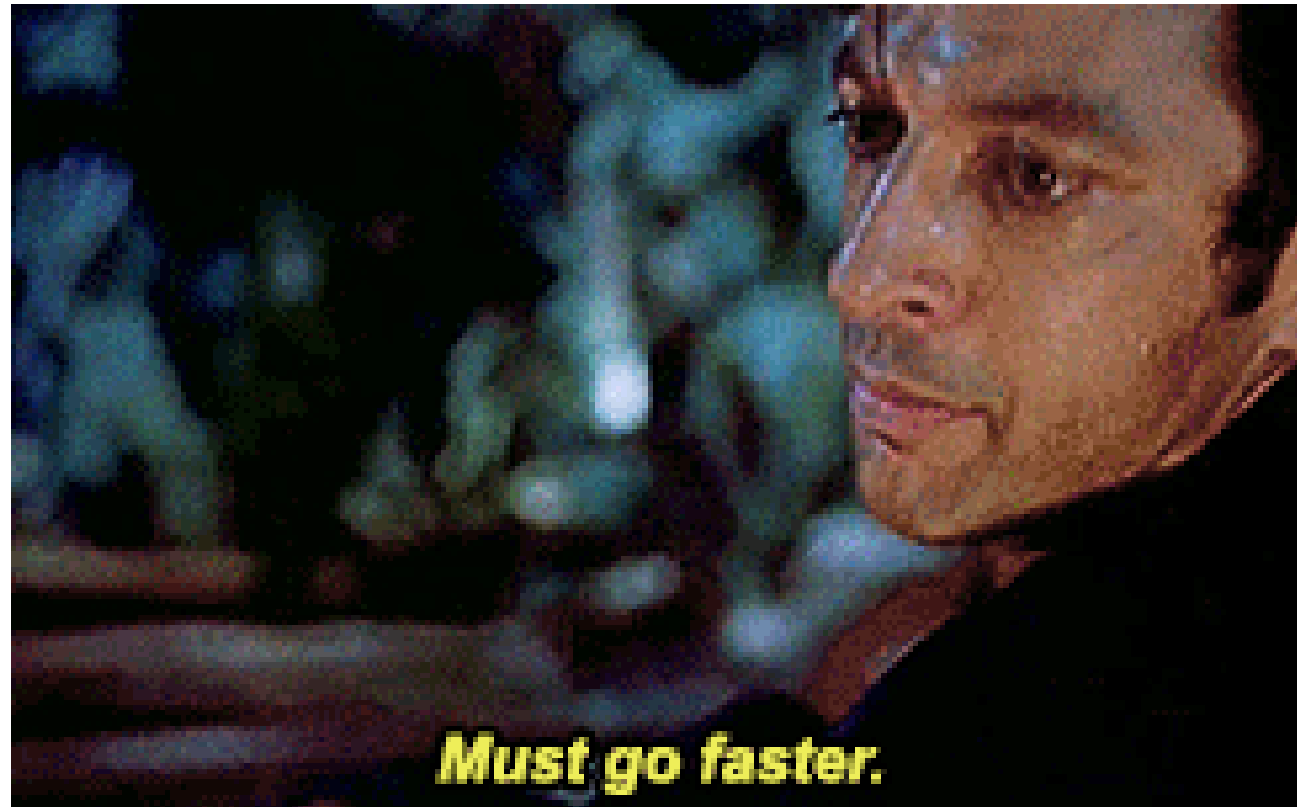
Or to avoid being throttled/blocked by other tenants

EGRESS DEDICATED IP

- Scale set of proxies dedicated per customer
- Dedicated egress load balancer that sets the outgoing ip
- Using network policies to allow only the customer to access its proxy

CONTINUOUS DELIVERY

AEM: From yearly to daily release



Using Jenkins for CI/CD

Using queues to trigger some jobs

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
- Need to push to all namespaces, would be easier pulling
- Moving to Helm operator that can be an improvement

KUBEVAL

Kubeval to validate Kubernetes schemas

We added schema validation of some custom CRDs

CONFTEST

`ConfTest` to validate policies while allowing developer autonomy

- security recommendations
- labelling standards
- image sources

csanchez.org

