OPTIMIZING RESOURCE USAGE IN KUBERNETES

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

AEM ON KUBERNETES

Running on Azure

50+ clusters and growing

Multiple regions: US, Europe, Australia, Singapore, Japan, India,...

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 ™

ENVIRONMENTS

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

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

CPU REQUESTS AND LIMITS

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

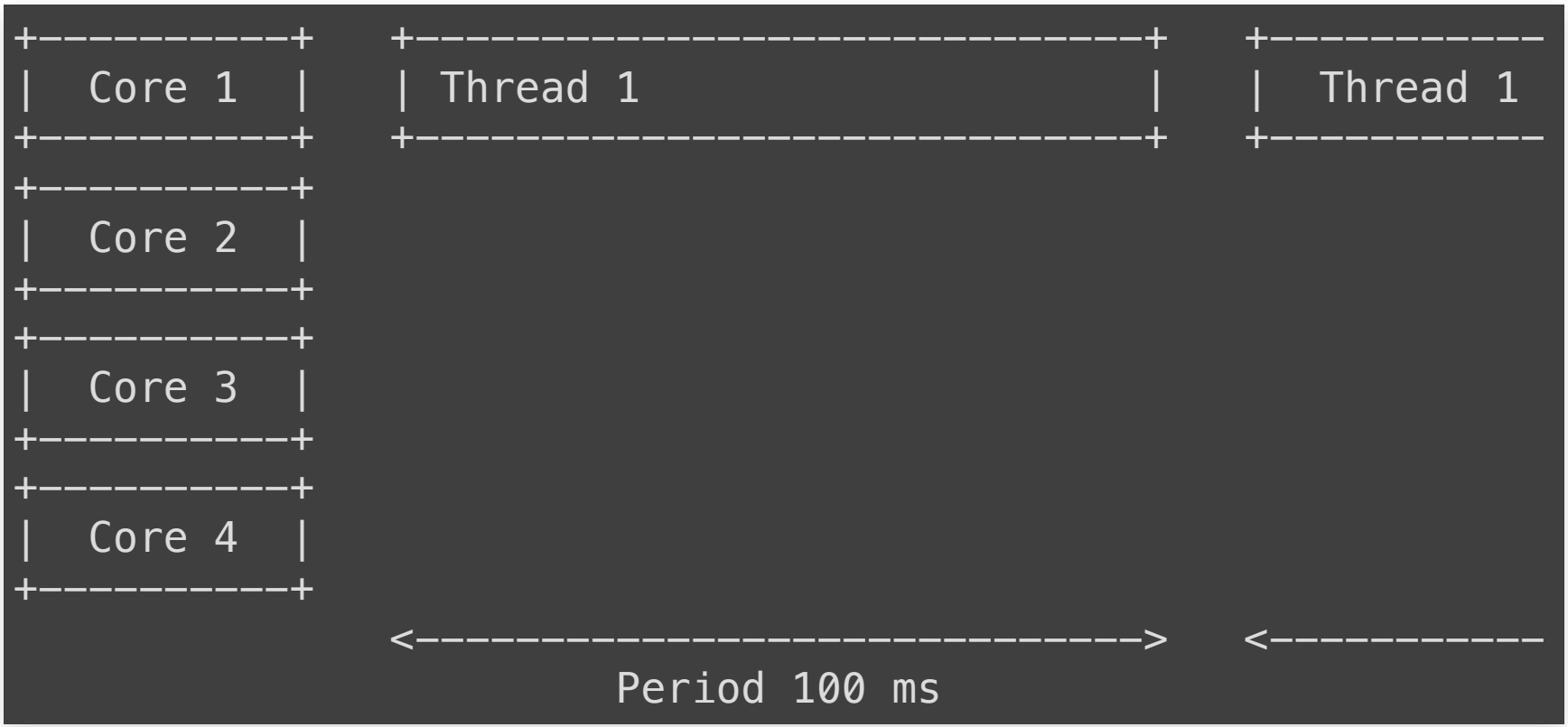
`1000m` in Kubernetes -> 100ms of CPU usage in each 100ms period

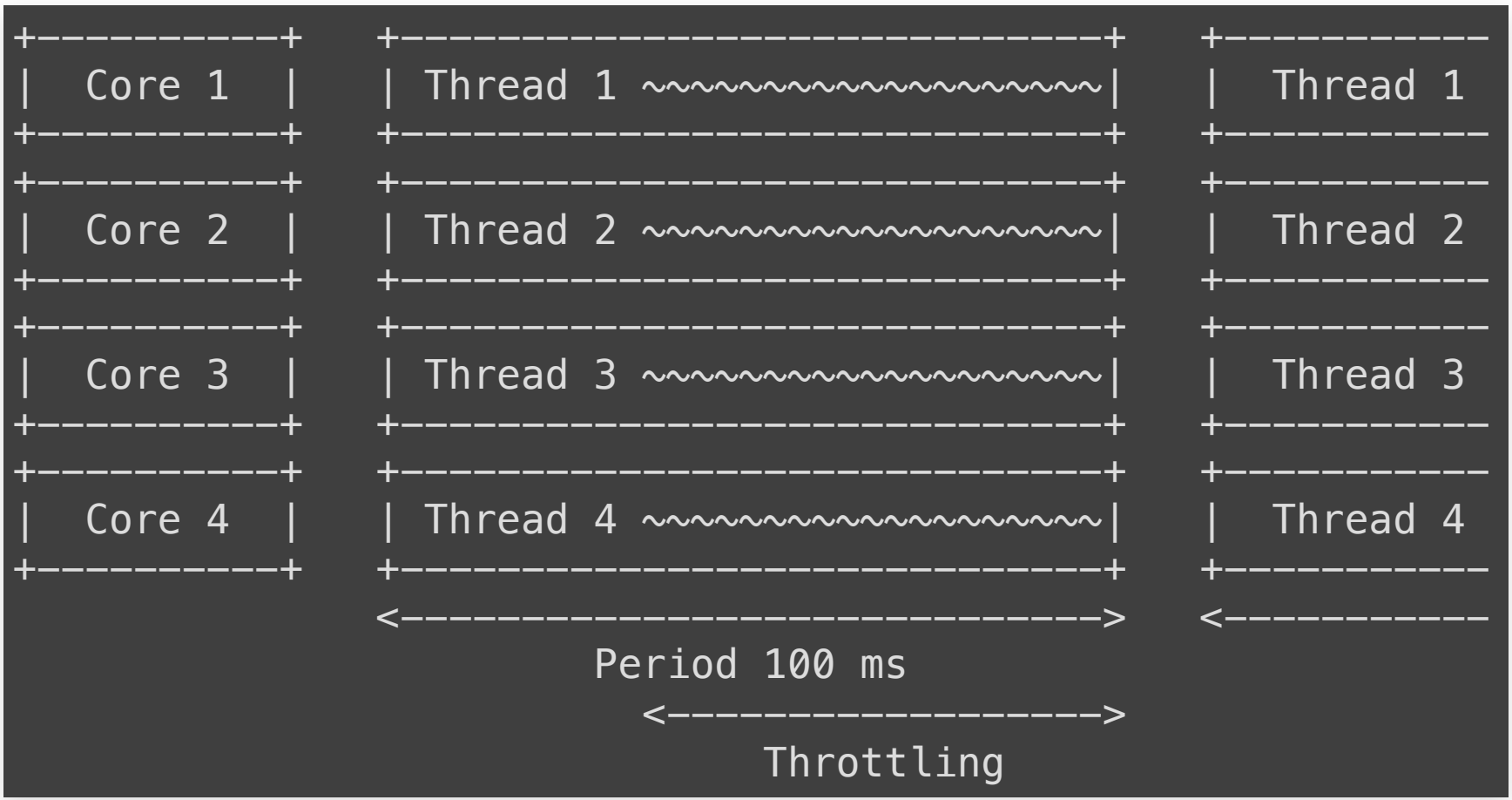
CPU LIMITS IN KUBERNETES

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


Core 1
Core 2
Core 3
Core 4
Thread 1
Thread 2
Thread 3
Thread 4
Thread 1
Thread 2
Thread 3
Thread 4
Period 100 ms
Throttling

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

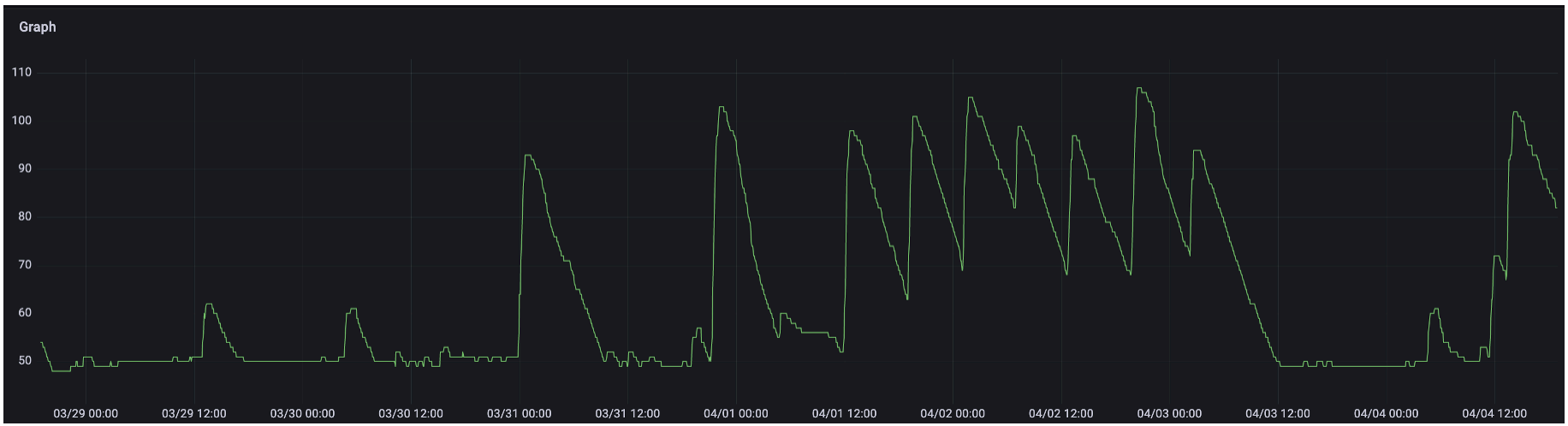
KUBERNETES CLUSTER AUTOSCALER

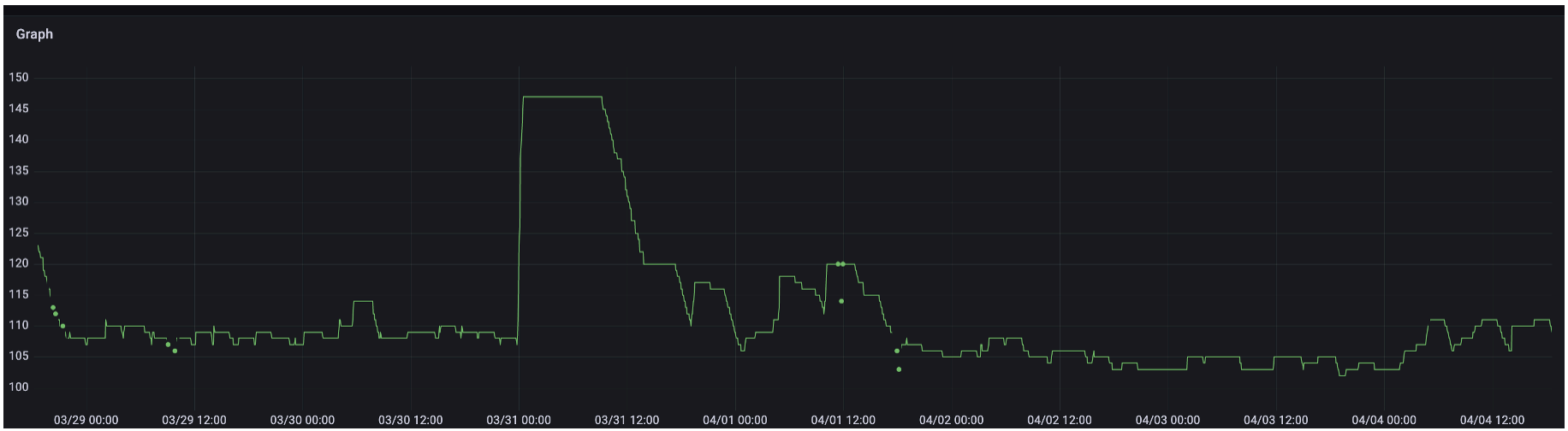
Max nodes managed at the cluster level

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

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)

In-place VPA alpha in 1.27

VPA

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

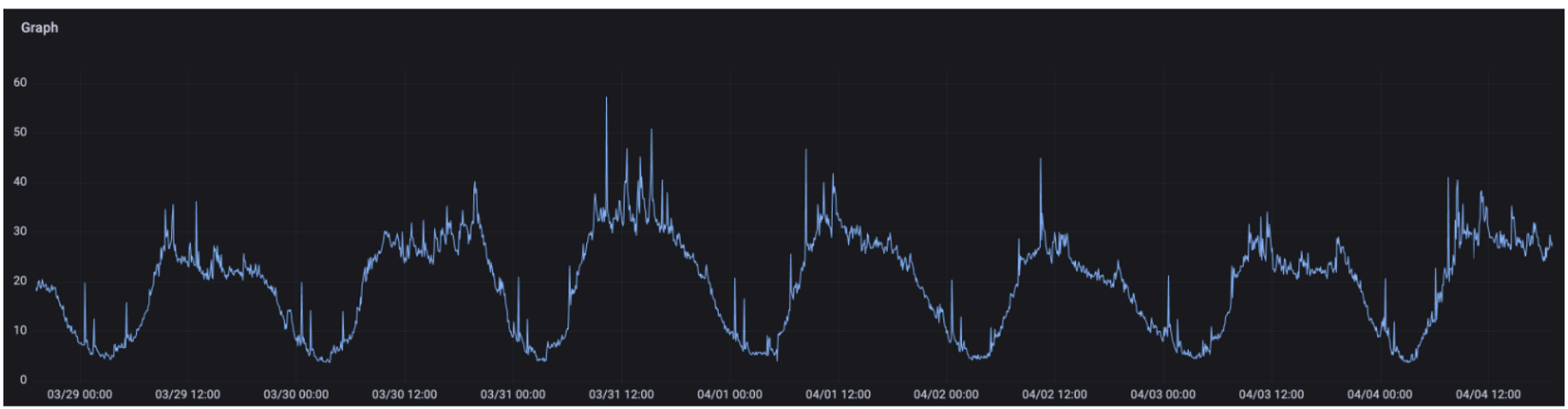
rpm autoscaling is better suited

As long as customers don’t have expensive requests

Savings: 50-75%

Pods vs RPM


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

ARM ARCHITECTURE

15-25% cost savings for the same performance

Easy switch for containerized Java and other languages

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%

Kubernetes-based Event Driven Autoscaler

Scale any container in Kubernetes based on events

KEDA SCALERS

- Kafka topic
- Cron schedule
- Azure, AWS, Google Cloud events
- Kubernetes Workload

...

OTHER SCALABILITY CONCERNS

ETCD LIMITS

Kubernetes is not a database

Stored objects capacity (secrets/configmaps) is not infinite

ETCD LIMITS

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!

MAX PODS PER NODE

The default is 110 pods per node

A lot of waste if you use big nodes and small pods

RESOURCE OPTIMIZATION IN KUBERNETES

From the application: tuning CPU, memory, GC

From Kubernetes ecosystem: Cluster autoscaler, HPA, VPA, KEDA

At application and infrastructure levels

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

csanchez.org

csanchez

carlossg

Adobe