KubeCon CloudNativeCon
Europe 2022

Resource Optimization in Kubernetes

Carlos Sanchez, Adobe
@csanchez

Cloud Engineer at Adobe Experience Manager Cloud Service

Author of Jenkins Kubernetes plugin

Long time OSS contributor at Jenkins, Apache Maven, Puppet,…

@csanchez

Adobe Experience Manager (AEM)

An existing distributed Java OSGi application

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

25+ clusters and growing

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

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 on Kubernetes

Each customer environment is a micro-monolith ™

Multiple teams building services

Need ways to scale that are orthogonal to the dev teams

Resources in Kubernetes

Kubernetes workloads must set resource requests and limits:

- Requests: how many resources are guaranteed
- Limits: how many resources can be consumed

Resources in Kubernetes

And are applied to

- CPU: may result in CPU throttling
- Memory: limit enforced, results in Kernel OOM killed
- Ephemeral storage: limit enforced, results in pod eviction

Resources in Kubernetes

AEM is a Java application

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

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

Multiple worker tiers defined as node groups

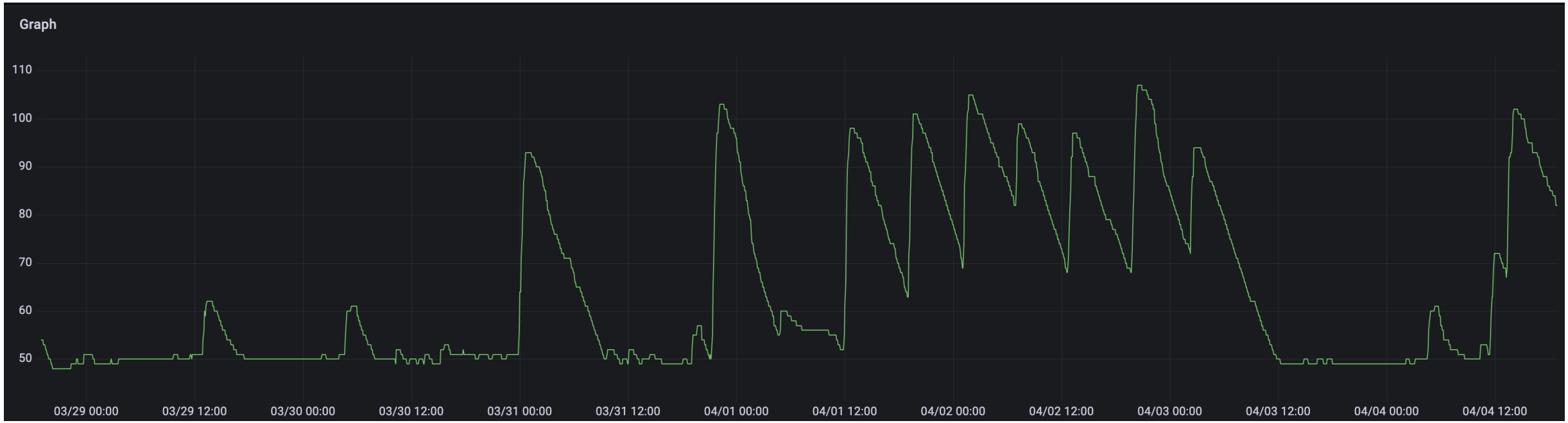
Max nodes managed at the cluster level

Least waste Scaling Strategy

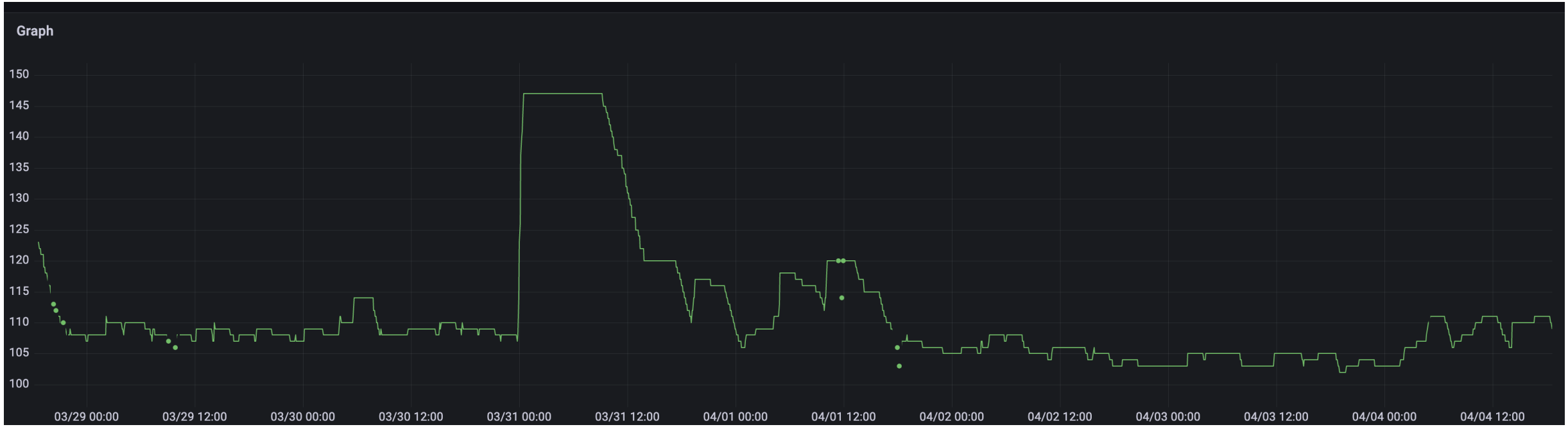
- Selects the node group with the least idle CPU after scale-up

Savings: 30-50%

Kubernetes Cluster Autoscaler



Kubernetes Cluster Autoscaler



Horizontal Pod Autoscaler

Creating more pods when needed

Horizontal Pod Autoscaler

AEM scales on CPU and http requests per minute (rpm) metrics

CPU autoscaling is problematic

Periodic tasks can spike the CPU, more pods do not help

Spikes on startup can trigger a cascading effect

Horizontal Pod Autoscaler

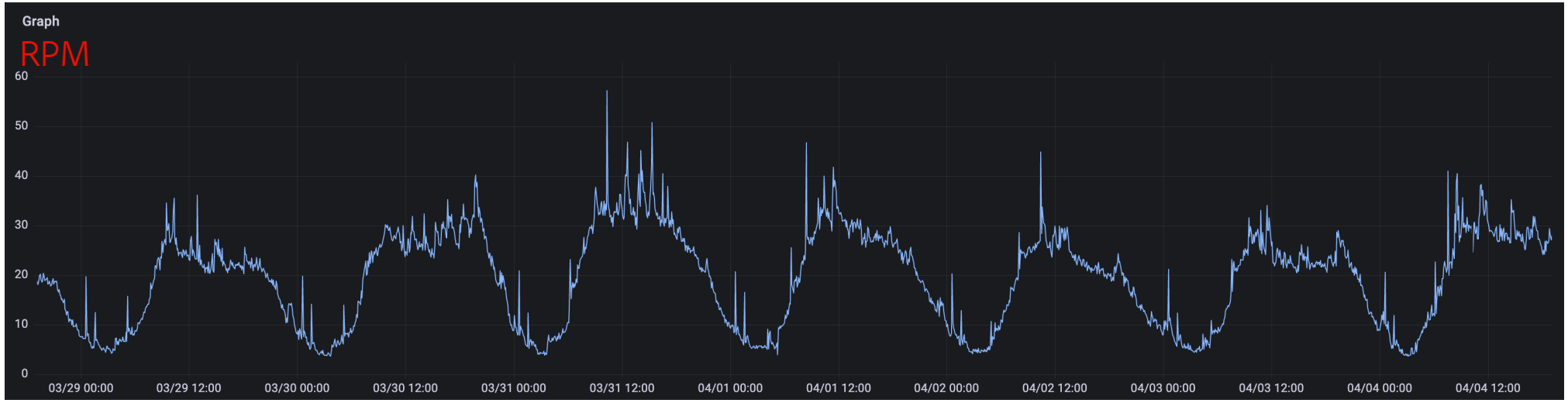
AEM needs to be warmed up on startup

rpm autoscaling is better suited

As long as customers don’t have expensive requests

Savings: 50-75%

Horizontal Pod Autoscaler



Vertical Pod Autoscaler

Increasing/decreasing the resources for each pod

Vertical Pod Autoscaler

Allows scaling resources up and down for a deployment

Requires restart of pods (automatic or on next start)

Makes it slow to respond, can exhaust resources in busy nodes

Vertical Pod Autoscaler

Only used in AEM dev environments to scale down if unused

And only for some containers

Savings: 5-15%

Hibernation

Scaling to zero environments not used

Hibernation

Scaling down multiple deployments associated to one “AEM environment”

Deleting ingress routes and other objects that may limit cluster scale

Hibernation

Cronjob that periodically checks for last access data in Prometheus

UI for user to dehibernate

Savings: 60-80%

Automatic Resource Configuration

ARC

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

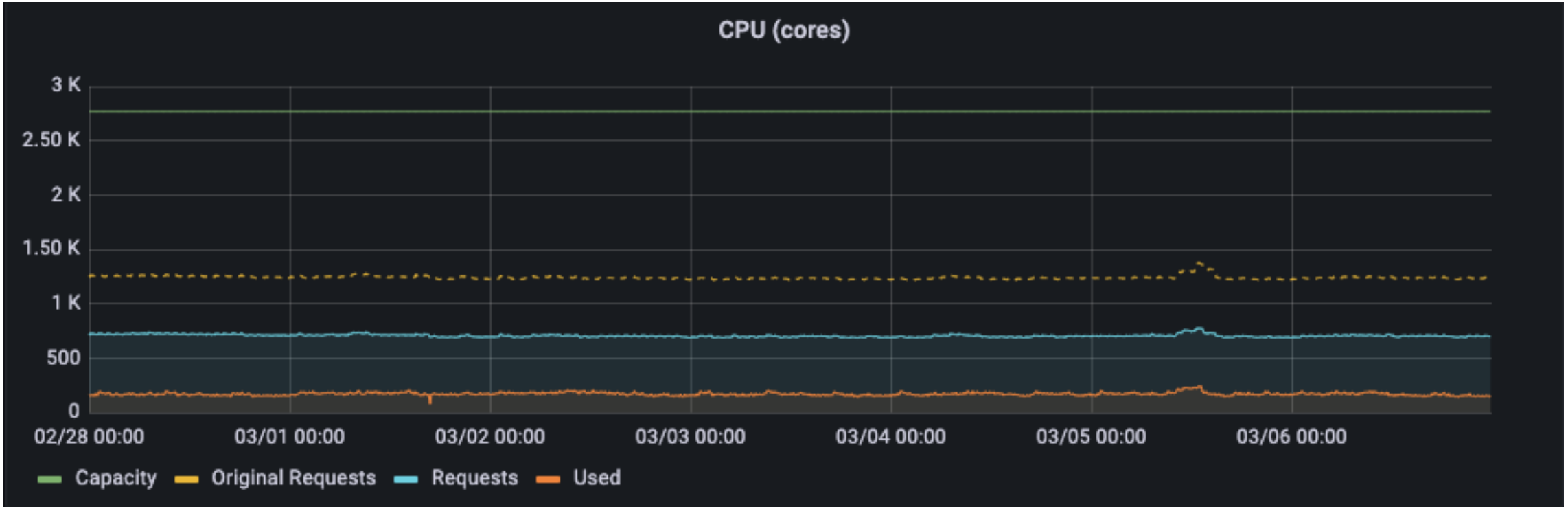
Can provide recommendations about optimization at deployment level based on actual usage

ARC Cluster Ratios

ARC can dial down resource requests at cluster/namespace level

Savings: 10-15%

ARC


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

ARC Recommender

Why ARC and not VPA recommender?

- Full control over recommendation algorithm
- Implementation at more global cluster level with deployment level recommendations

Resource Optimization in Kubernetes

From Kubernetes ecosystem:

Cluster autoscaler, HPA, VPA

Internal:

Hibernation, ARC

At application and infrastructure levels

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

@csanchez