



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

How Many Spark Applications Can Your etcd Really Handle?

João Azevedo & João Soares, Feedzai



Who are we?



João Azevedo
Platform Engineer



João Soares
Platform Engineer



feedzai[↑]

The use-case:

- Real-time fraud detection platform for banks and payment processors.
- **~90B** events processed and **\$8T** in payments secured last year.
- Data must stay fresh for decisions to be accurate.



Spark Workloads:

- **Backbone layer** for batch processing, feature engineering and model training.
- Delaying execution of Spark Workloads = **degraded** transaction **scoring** + possible **undetected fraud**.

Our scale:

- **150+ Production YARN clusters** (isolated per client/use-case).
- **4,000 Spark applications** launched **per day** (**~3 / minute**) across several regions, and growing.

Moving away from YARN



KubeCon



CloudNativeCon

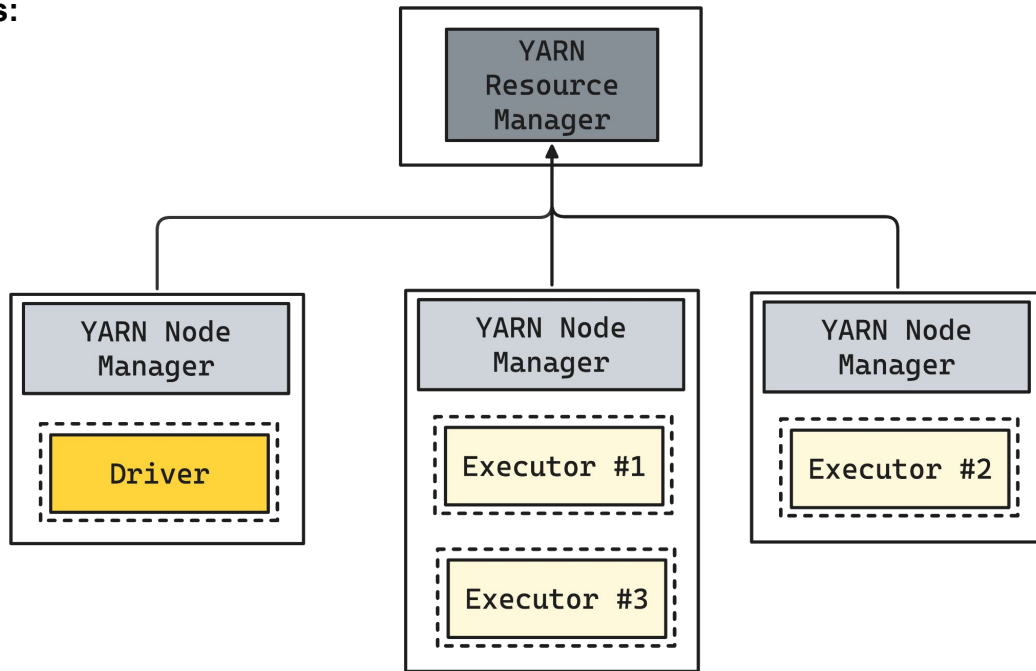
Europe 2026

Our previous setup presented several challenges:

- **Low Resource Efficiency:**
 - 1 Cluster / client and/or use-case
 - No "multi-tenancy" concept.
- **Lack of self-serviceability:**
 - Job debugging and troubleshooting.
 - History server access and replay.
- **Independent cluster lifecycle:**
 - Upgrade coordination.
 - Configuration management.

Also, we are undergoing a **strategic shift**:

- Moving towards **Kubernetes consolidation**.
- **Single** operational model.
- **Cloud-native** approach.
- Increased **cost-efficiency**.



* ... this is repeated for every client/use-case



KubeCon



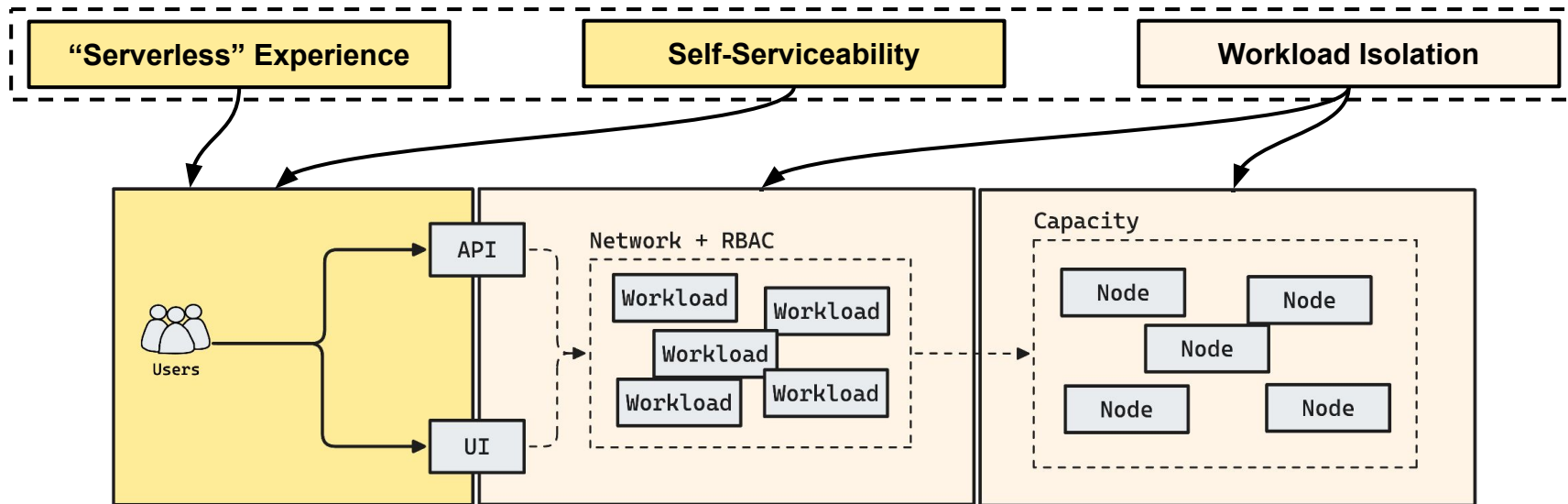
CloudNativeCon

Europe 2026

From YARN to Kubernetes

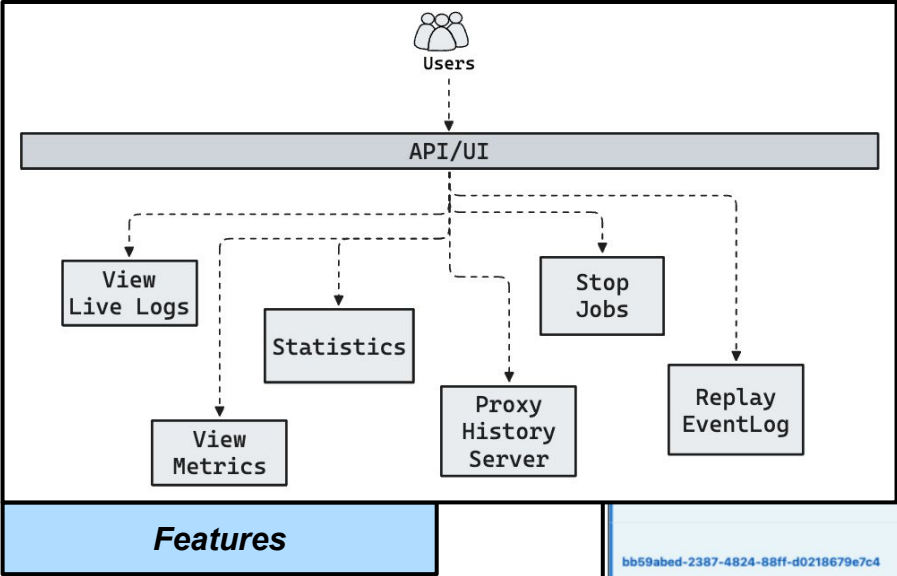


Spark as a Service on Kubernetes



Key Takeaway: Workloads live in the **service's namespace**, not in the user's namespaces.

User-facing layer



op
bs

Replay
EventLog

SPARK SERVICE UI

COMPLETED2

FAILED39

PENDING SUBMISSION0

SUBMITTED1

PENDING DELETION0

PENDING STOP0

STOPPED27

All Clusters

SYSTEM: All Systems

STATUS: All Statuses

Cluster	Queue	System	Name	Created At	Last Submission	Duration	Status	Actions
				2026-02-17 12:45:40	2026-02-17 12:46:38	✓ 18 minutes, 13 seconds	COMPLETED	
bb59abed-2387-4824-88ff-d0218679e7c4				2026-02-17 12:31:20	2026-02-17 12:32:16	✓ 17 minutes, 7 seconds	COMPLETED	
ad065160-cee9-4156-a1a5-d8a8fe530922				2026-02-17 11:48:28	2026-02-17 11:49:42	⌚ 2 hours, 15 minutes, 7 seconds	RUNNING	
134e75d0-44e0-4b6e-80ad-6d6776ee2acc				2026-02-17 11:29:14	2026-02-17 11:29:54	✓ 14 minutes, 7 seconds	COMPLETED	

Orchestration Layer

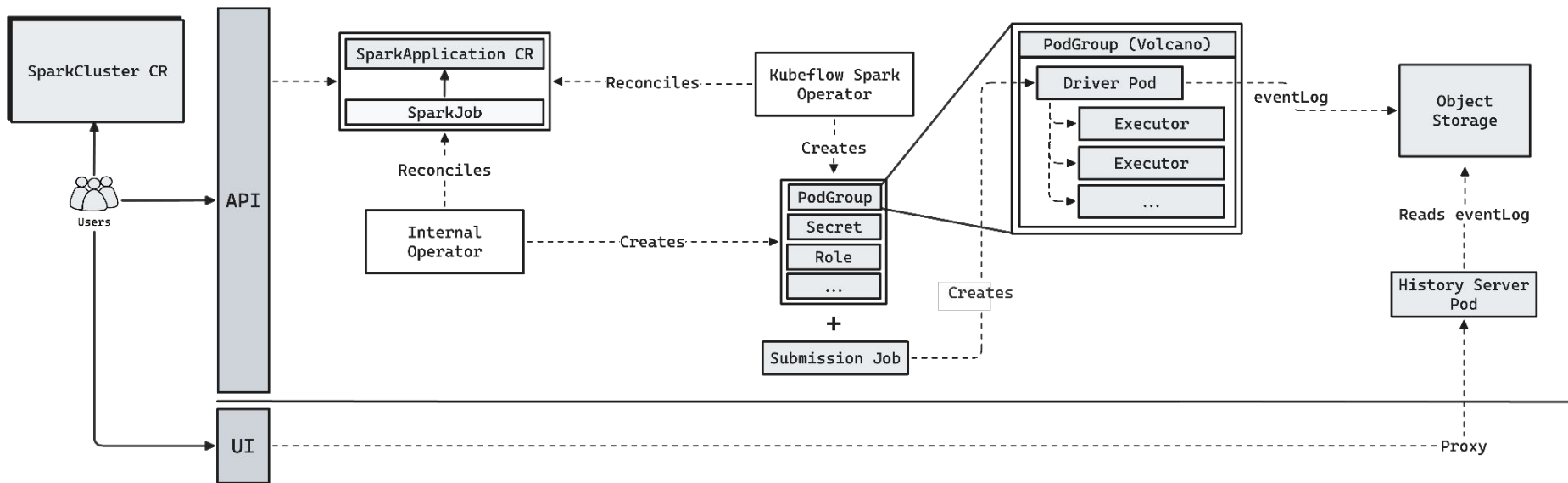


KubeCon



CloudNativeCon

Europe 2026



- **SparkCluster CR** (“Cluster Abstraction”): An abstraction of a “virtual capacity” for user workloads, where they can submit jobs.
- **Kubeflow Spark Operator**: We adapted the submission flow to use a K8s job (for scalability and security reasons).
- **History Server Replay**: A fluentd daemonset runs in the drivers node group to ship event logs for Object Storage.

Scheduling Layer



KubeCon



CloudNativeCon

Europe 2026

Volcano System:

- Batch scheduling for K8s.
- Gang Scheduling, Hierarchical Queues.
- Priority Scheduling, Preemption, etc.

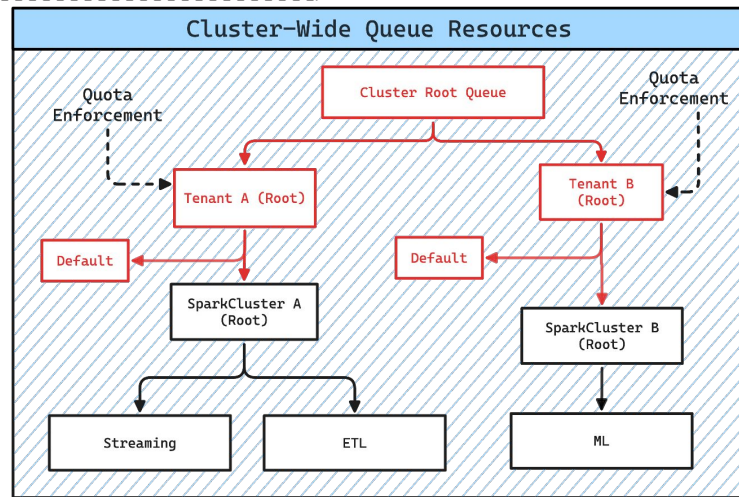
Hierarchical Queues:

- Similar to YARN Capacity Scheduler.
- Provided by Volcano's Capacity Plugin.

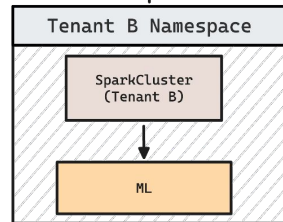
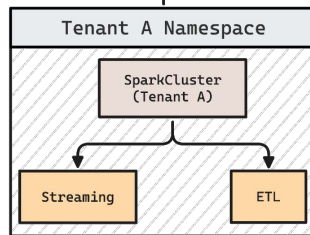
Quota Enforcement:

- Volcano provides cluster-wide Queue.
- We created a Namespaced version.
- Quotas are enforced at tenant's root queue.

Cluster view



Users view





KubeCon



CloudNativeCon

Europe 2026

What We Discovered at Scale



Anatomy of a Spark Application



KubeCon



CloudNativeCon

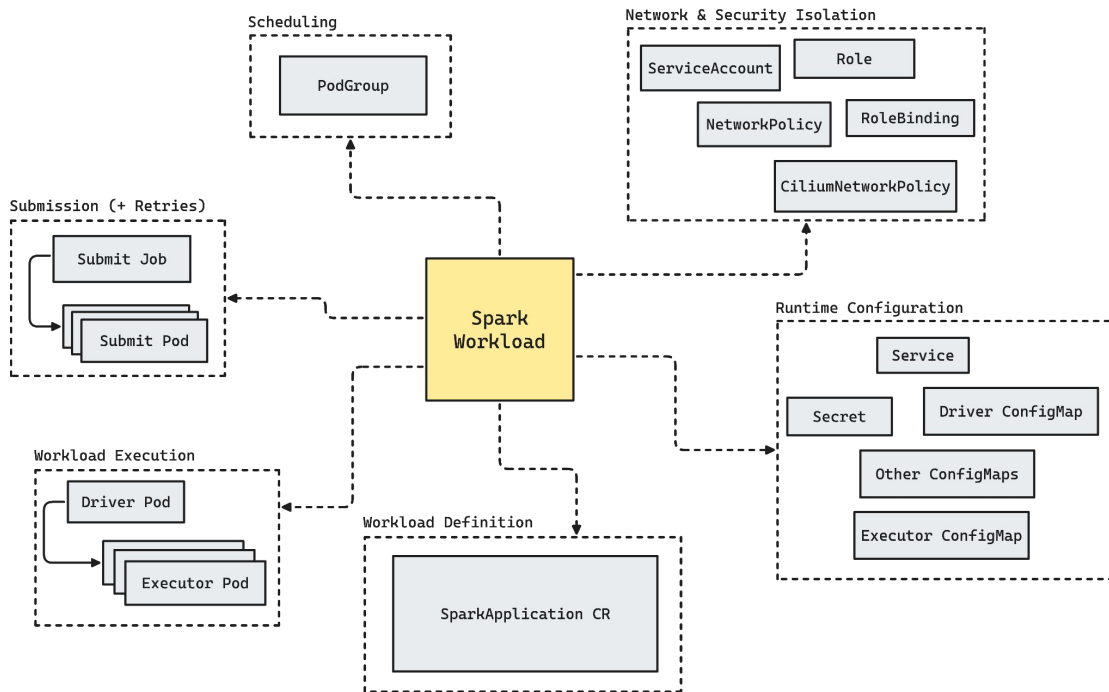
Europe 2026

1 Job can have +15 K8s Objects! Why?

- Workload Isolation.
- Gang Scheduling.
- Platform features.
- Configuration!

For us a typical job has 150 KiB
and this in turn generates 4.5x more

- Breakdown of created objects:
 - **50%** Submission Objects
 - **20%** ConfigMaps (Driver)
 - **20%** SparkApplication CR
 - **10%** Others
- **Running** Jobs require all these objects!
- **When jobs terminate, we require:**
 - SparkApplication CRs.
 - Debugging purposes.



The accumulation problem

Running Jobs

# Jobs	Total YAML Size	etcd (200 MiB)	etcd (2 GiB)	etcd (8 GiB)	etcd (16 GiB)
1	0.00065 GiB	0.33%	0.03%	0.01%	0.00%
100	0.06537 GiB	32.68%	3.19%	0.80%	0.40%
1k	0.65365 GiB	326.83%	31.92%	7.98%	3.99%
5k	3.26827 GiB	1634.14%	159.58%	39.90%	19.95%
8k	5.22923 GiB	2614.62%	255.33%	63.83%	31.92%
10k	6.53654 GiB	3268.27%	319.17%	79.79%	39.90%

* ...we assume ~40% as a maximum usage **safe threshold**.

The accumulation problem

Terminated Jobs (Rate: 1 Job / Minute)

Retention Days	# Jobs	Total YAML Size	etcd (200 MiB)	etcd (2 GiB)	etcd (8 GiB)	etcd (16 GiB)
7	~10k	1.43649 GiB	718.25%	70.14%	17.54%	8.77%
14	~20k	2.87298 GiB	1436.49%	140.28%	35.07%	17.54%
30	~43k	6.15639 GiB	3078.20%	300.60%	75.15%	37.58%

Terminated Jobs (Rate: 5 Jobs / Minute)

Retention Days	# Jobs	Total YAML Size	etcd (200 MiB)	etcd (2 GiB)	etcd (8 GiB)	etcd (16 GiB)
7	~50k	7.18245 GiB	3591.23%	350.71%	87.68%	43.84%
14	~100k	14.3649 GiB	7182.45%	701.41%	175.35%	87.68%
30	~216k	30.7819 GiB	15390.95%	1503.02%	375.76%	187.88%

Benchmarking: 3k Jobs, ~5x Storage



KubeCon



CloudNativeCon

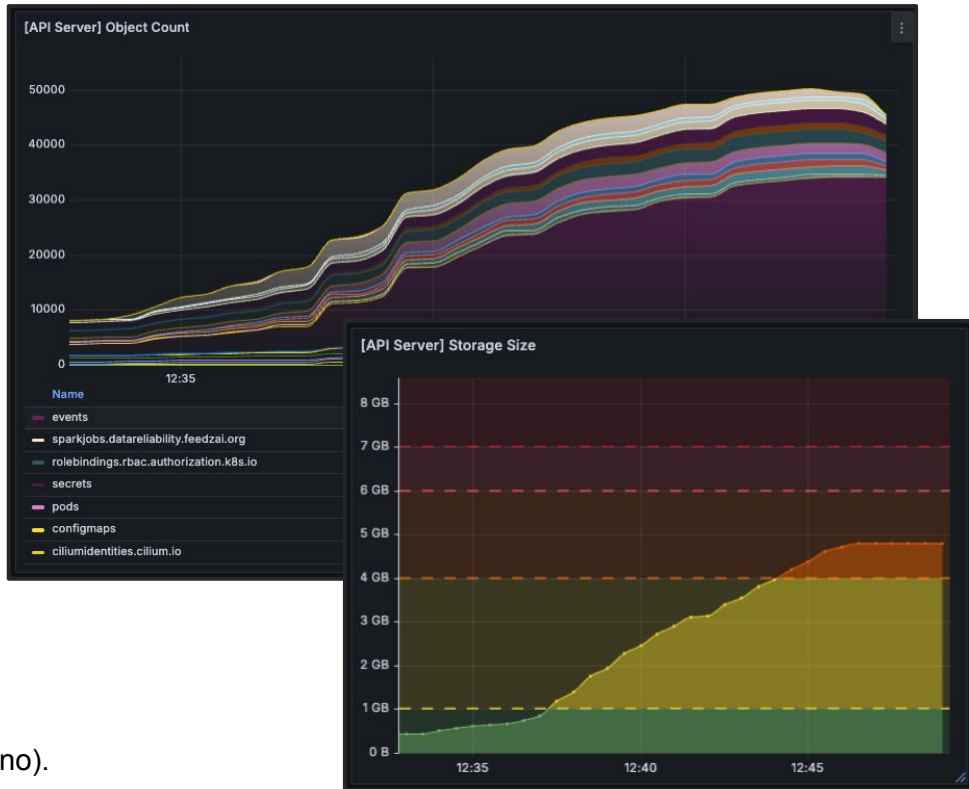
Europe 2026

For the same 150 KiB payload, 3k jobs submitted:

- ~15 minutes after benchmark start:
 - ~30k Events.
 - 800 SparkApplication CRs.
 - 700 Driver Pods.
 - 1.4k Secrets, Roles, RoleBindings.
 - ...~55k objects in total!
- Storage:
 - Baseline: **~270 MiB**
 - Logical size of the 55k objects: **~700 MiB**.
 - Size reported after ~15 mins: **~4.80 GiB**.
- Metric used: *apiserver_storage_size_bytes*

Side Effects:

- Increase in API Server and etcd latencies (LIST, POST).
- Aborted requests, timeouts, etc.
- Other services in the chain impacted (e.g., Cilium, Kyverno).



Why 4 GiB?

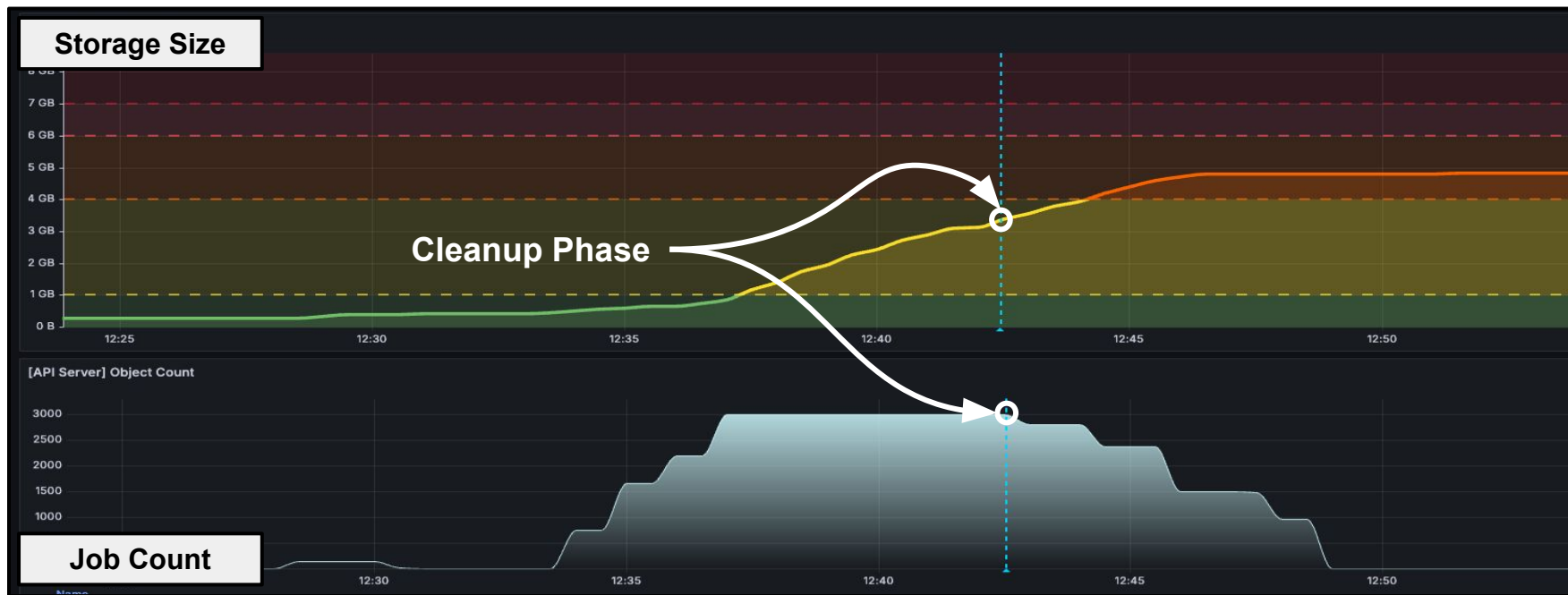


KubeCon



CloudNativeCon

Europe 2026



K8s Events

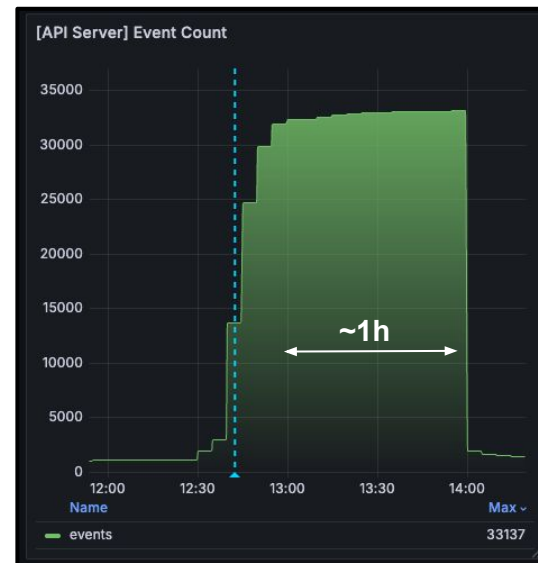
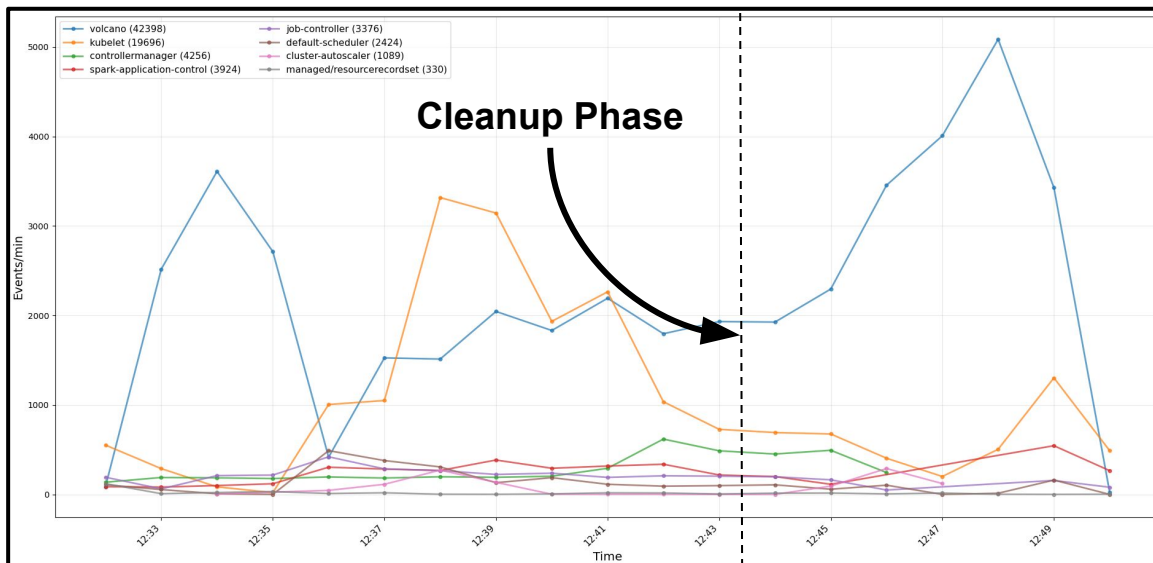
Huge Objects

Churn

Updates

Fragmentation

K8s Events



Events are K8s objects typically with a TTL of 1h.

MVCC: Every Update is a Full Copy

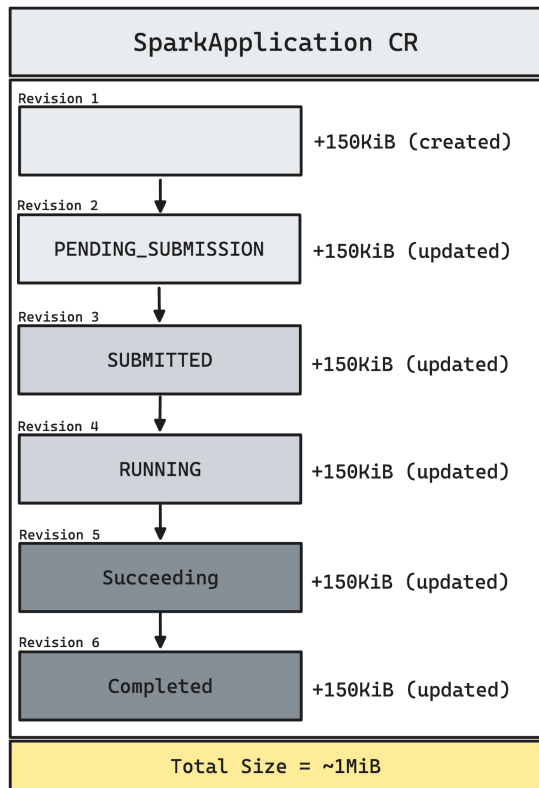


KubeCon



CloudNativeCon

Europe 2026



MVCC in etcd:

- Multi-Version Concurrency Control.
- Each revision is a complete copy of the object.

It's not just SparkApplications updating...

- Submission Pods (pending → scheduled → running → completed).
- PodGroups (pending → inqueue → running).
- Driver/Executor Pods (pending → scheduled → running → completed).
- **Bench. Update Counts:** 11k SparkApplications. 10K Pods. 4k PodGroups etc.

Object lifecycle in etcd:

- **Delete** does not actually delete data, it only creates **tombstones** (new revision).
- **Compaction** marks old revisions as "**free space**" (logically).
- Physical space is only removed with **defragmentation**.
- **Defragmentation** is an **expensive operation** (locks database briefly).
- When space is exceeded, a **nospace alarm** freezes the control plane.

Storage Fragmentation

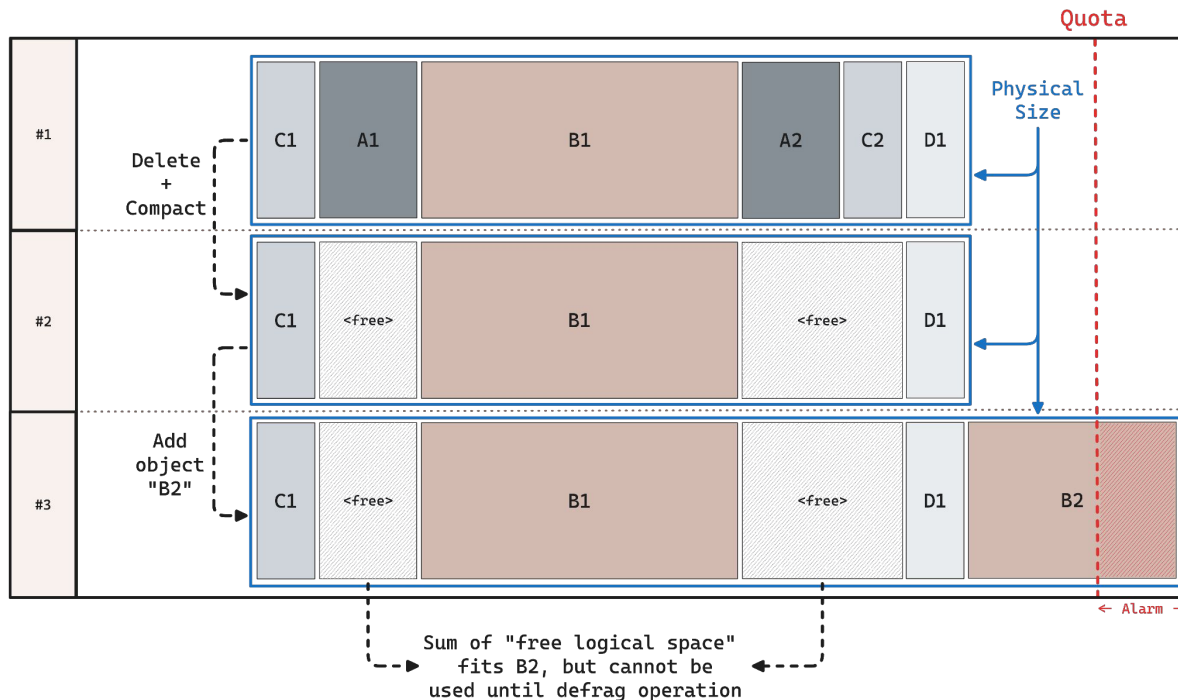


KubeCon



CloudNativeCon

Europe 2026



etcd backend size can grow as the space gets more fragmented – defrag is required but expensive.



KubeCon



CloudNativeCon

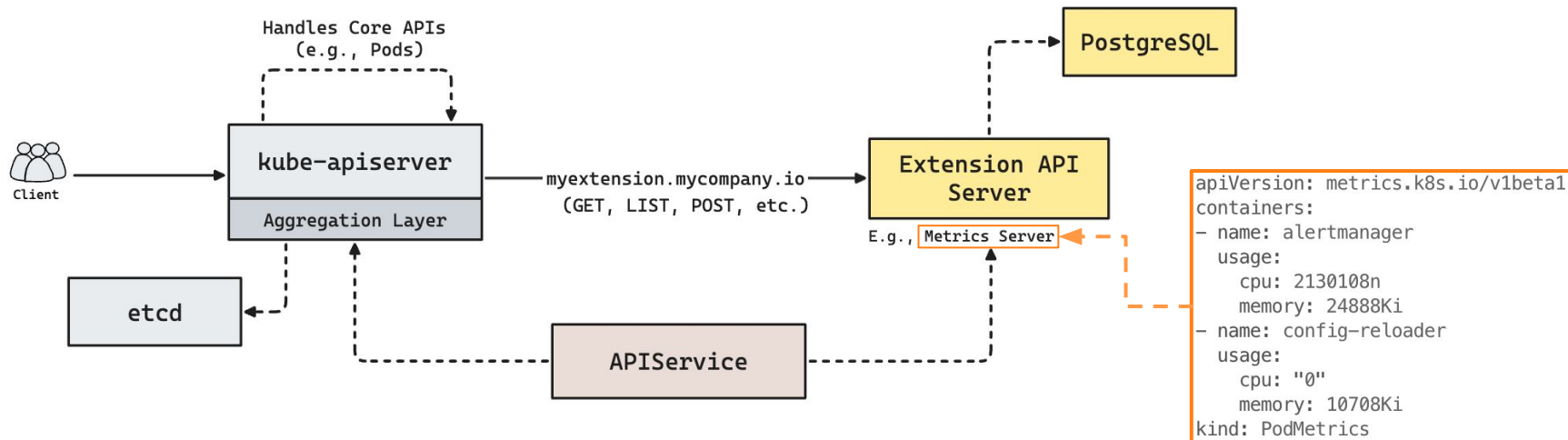
Europe 2026

Beyond etcd: API Aggregation Layer



We may not need etcd for every object

But wait, don't all Kubernetes resources live in etcd?



*Keda and Metrics Server are two examples of projects that use the **API Aggregation Layer**.*

Demo!



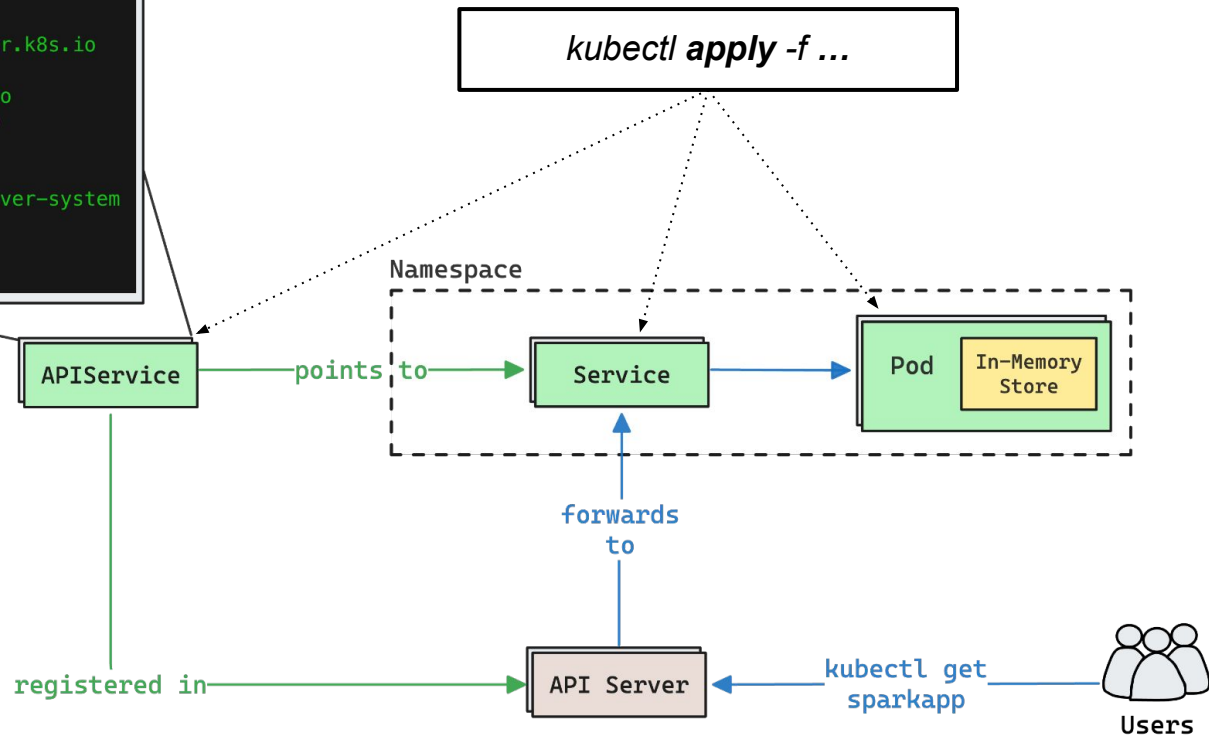
KubeCon



CloudNativeCon

Europe 2026

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1beta2.sparkoperator.k8s.io
spec:
  group: sparkoperator.k8s.io
  groupPriorityMinimum: 1000
  service:
    name: spark-api-server
    namespace: spark-api-server-system
    port: 443
  version: v1beta2
  versionPriority: 100
```



Benchmarking again...



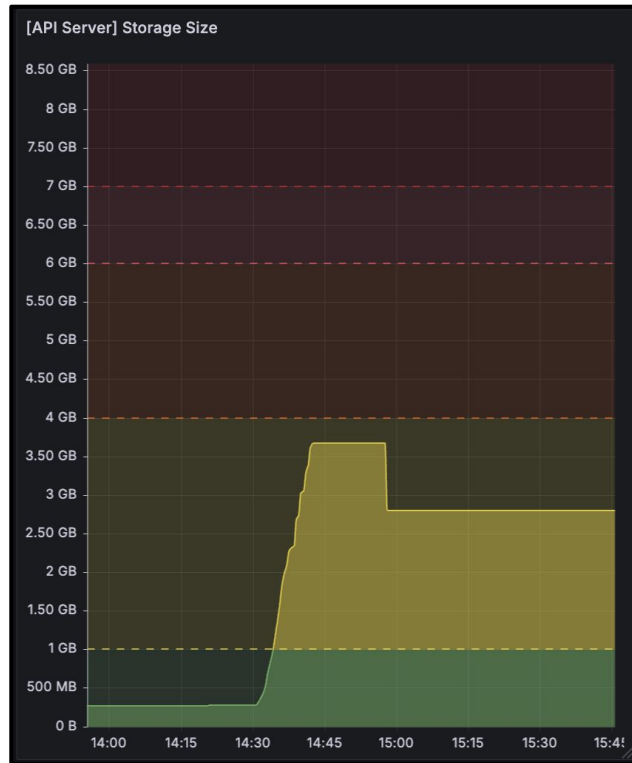
KubeCon



CloudNativeCon

Europe 2026

	<i>etcd-only</i>	<i>With Aggregation Layer</i>	<i>Delta</i>
<i>Objects created</i>	~55k	~78k	~ 30% more (23k) (+20k K8s events, more SparkApplications, etc.)
<i>Logical Size</i>	~700 MiB	~830 MiB	~ 20% more (100 MiB) (This test reached +500 SparkApplication CRs)
<i>Physical size (etcd)</i>	~4.8 GiB	~2.8 GiB	~ 45% less (2GB decrease in etcd physical size)





KubeCon



CloudNativeCon

Europe 2026

Lessons Learned



Wrap Up!



KubeCon



CloudNativeCon

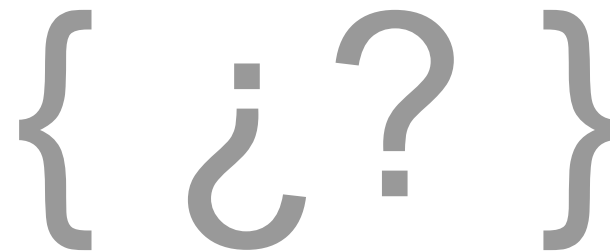
Europe 2026

CRDs are not free:

- Size cost: Large objects (>10KB) x MVCC revisions.
- Churn cost: Frequent status updates create revisions.
- Retention cost: Historical objects stay in etcd.

Design principles:

- Keep custom resources small.
- Minimize status updates.
- Design for deletion: TTL controllers, finalizers with timeouts.



When should you use API Aggregation Layer?

- Multi-cluster state synchronization (e.g., ADL / blueprint).
- Move heavy, high-churn, large status objects OUT of etcd.

Retention belongs in a long-term storage database, not etcd.



KubeCon



CloudNativeCon

Europe 2026

Give us your feedback



Thank you