



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Multi-cluster Orchestration System: Karmada Updates and Use Cases

Hongcai Ren, Huawei
Michas Szacillo, Bloomberg
Tessa Pham, Bloomberg
Zongqing Li, Ctrip.com





Hongcai Ren
Maintainer of Karmada



Tessa Pham
Senior Software Engineer
Bloomberg



Zongqing Li
Senior Software Engineer
Trip.com



Michas Szacillo
Senior Software Engineer
Bloomberg

Agenda



KubeCon



CloudNativeCon

Europe 2026

- **Karmada Overview**
- **New Features**
- **Real-World Case Studies**
- **Community Growth**
- **Roadmap of 2026**
- **Q&A**

Karmada Overview

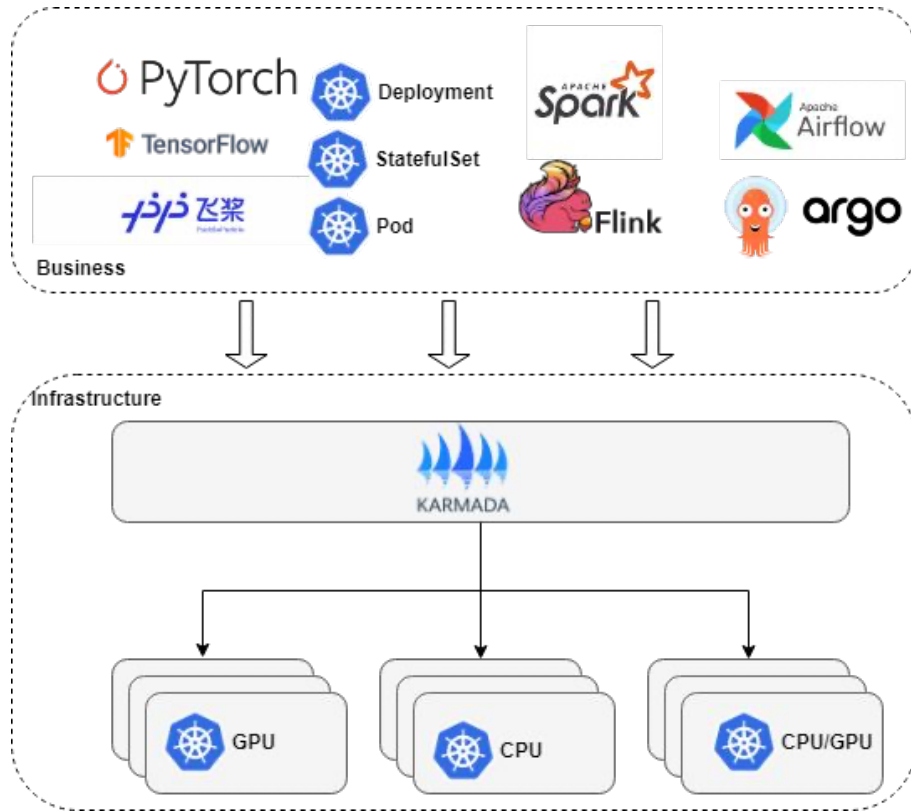


KubeCon



CloudNativeCon

Europe 2026



Karmada is an open source Kubernetes management system that enables you to run your cloud-native applications across multiple Kubernetes clusters and cloud providers.

What's New

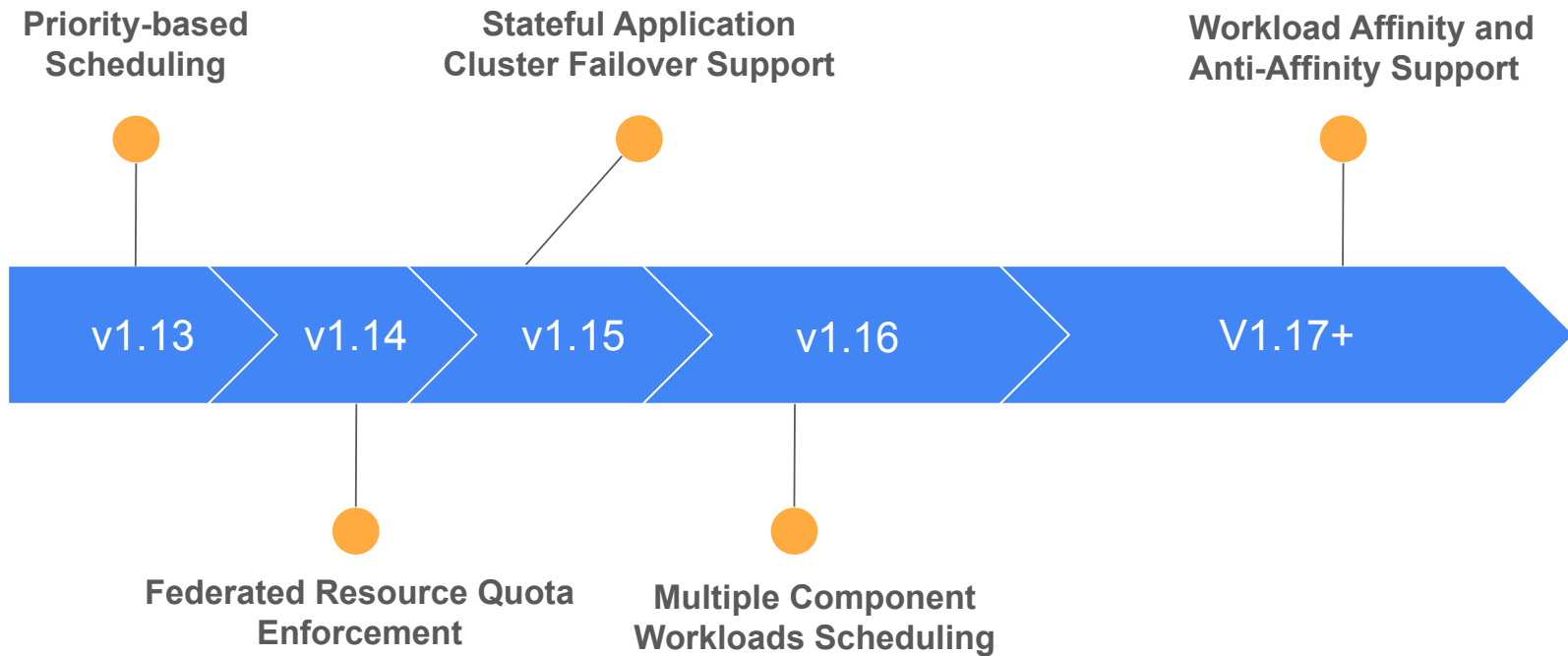


KubeCon



CloudNativeCon

Europe 2026



Stateful Application Cluster Failover

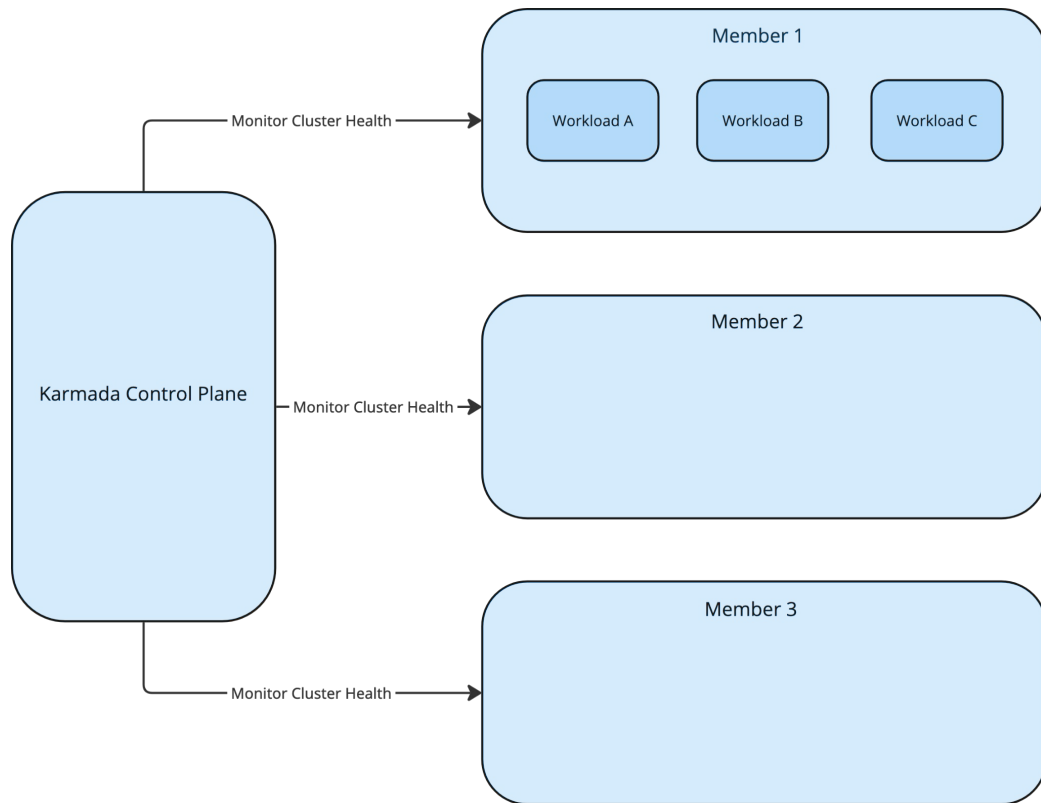


KubeCon



CloudNativeCon

Europe 2026



Stateful Application Cluster Failover

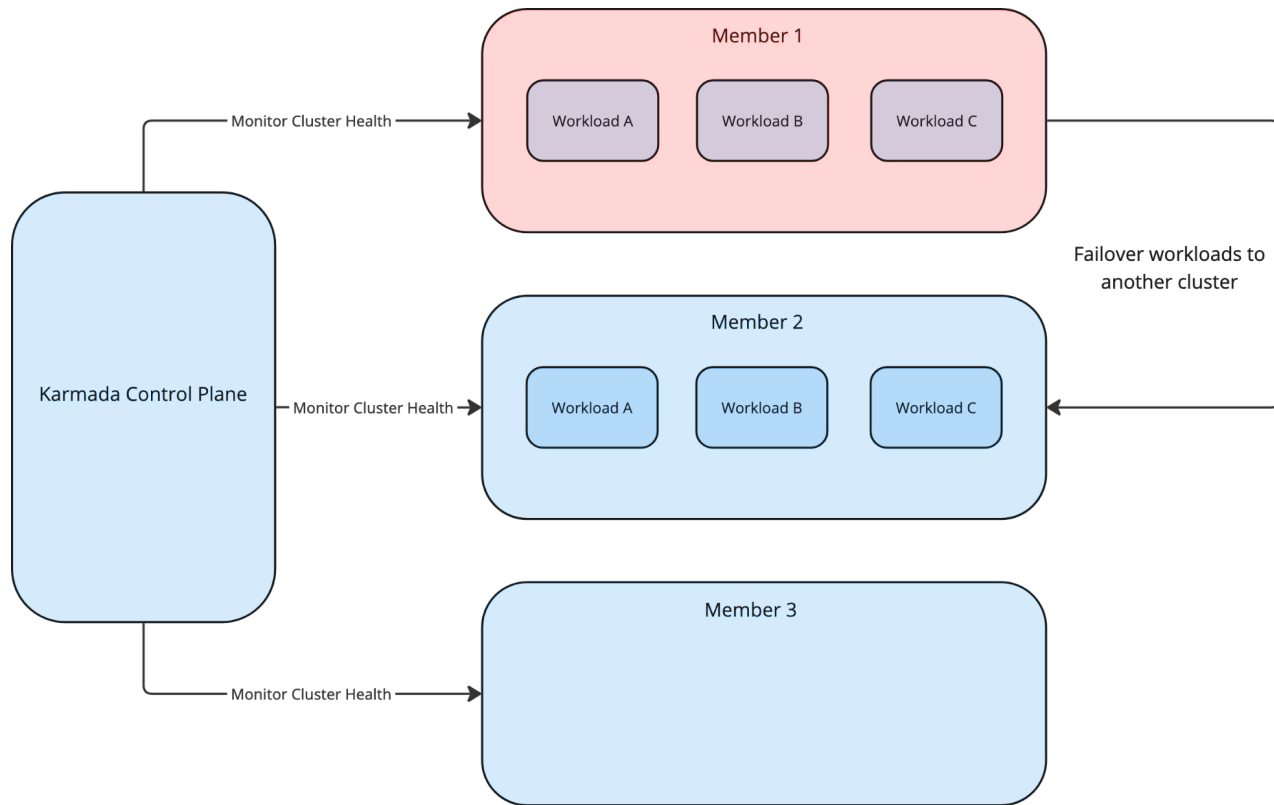


KubeCon



CloudNativeCon

Europe 2026



Stateful Application Cluster Failover



KubeCon



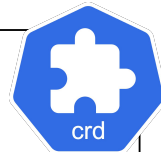
CloudNativeCon

Europe 2026

The cluster failover API has been extended to support **state preservation**!

1. A **JSONPath** expression targeting specific pieces of state data to be preserved during failover events
2. An **AliasLabelName** serving as a label key when the preserved data is passed to the new cluster

```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: flinkdeployment-propagation-policy
  namespace: user-ns
spec:
  failover:
    cluster:
      purgeMode: Directly
      statePreservation:
        rules:
          - aliasLabelName:rb.karmada.io/failover-jobid
            jsonPath: '{ .jobStatus.jobId }'
  resourceSelectors:
    - apiVersion: flink.apache.org/v1beta1
      kind: FlinkDeployment
      namespace: user-ns
      schedulerName: default-scheduler
```



Federated Quota Enforcement

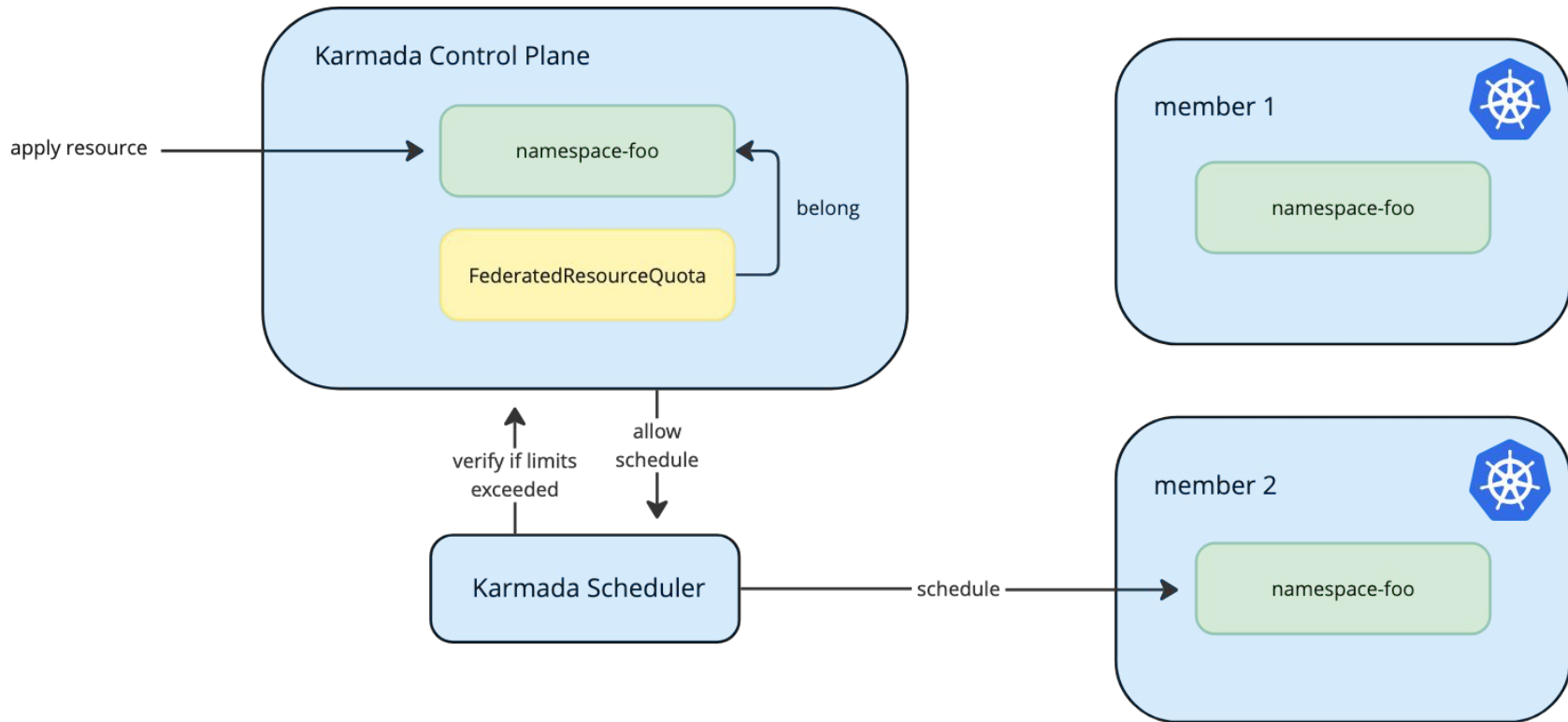


KubeCon



CloudNativeCon

Europe 2026



Federated Quota Enforcement

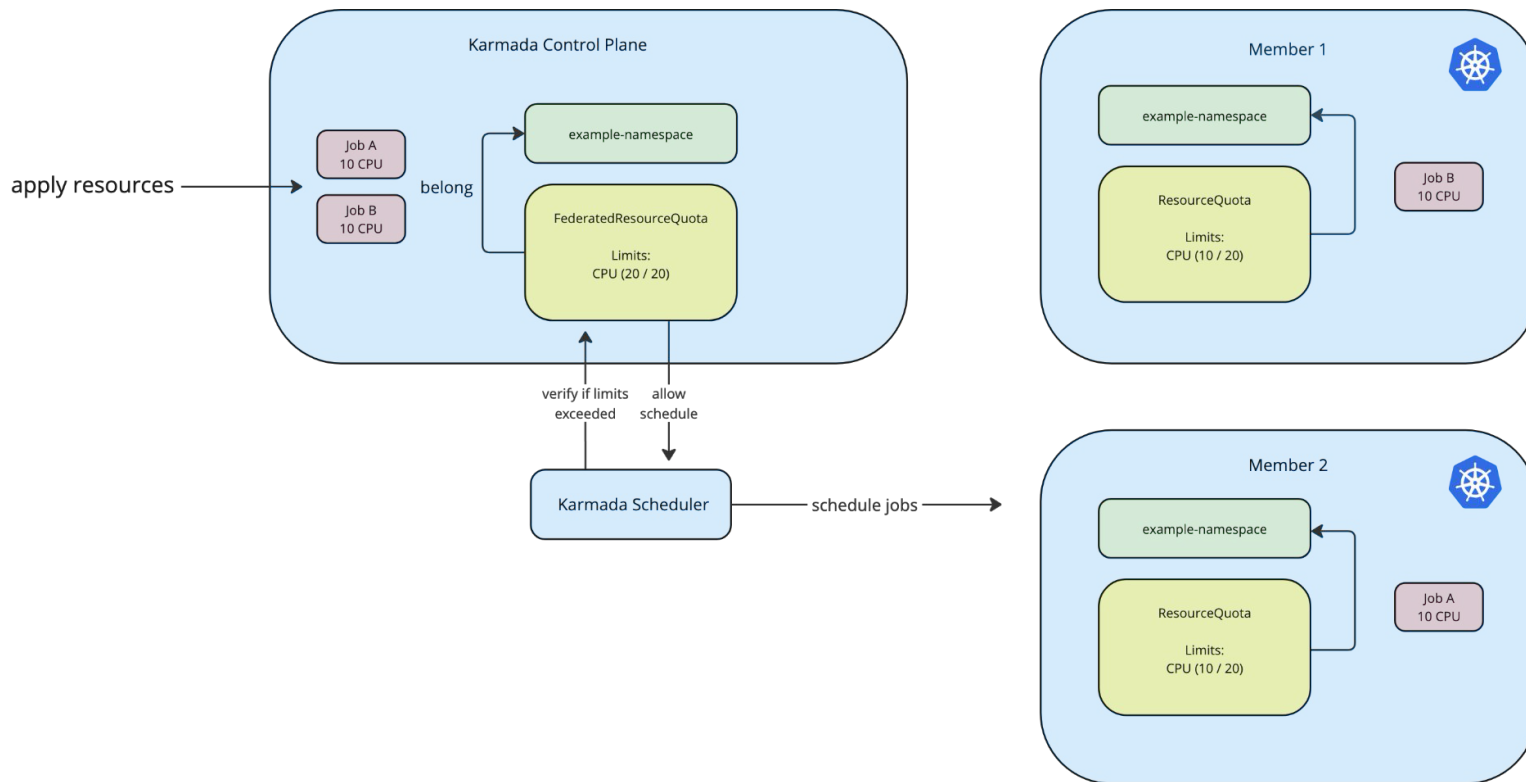


KubeCon



CloudNativeCon

Europe 2026



Federated Quota Enforcement

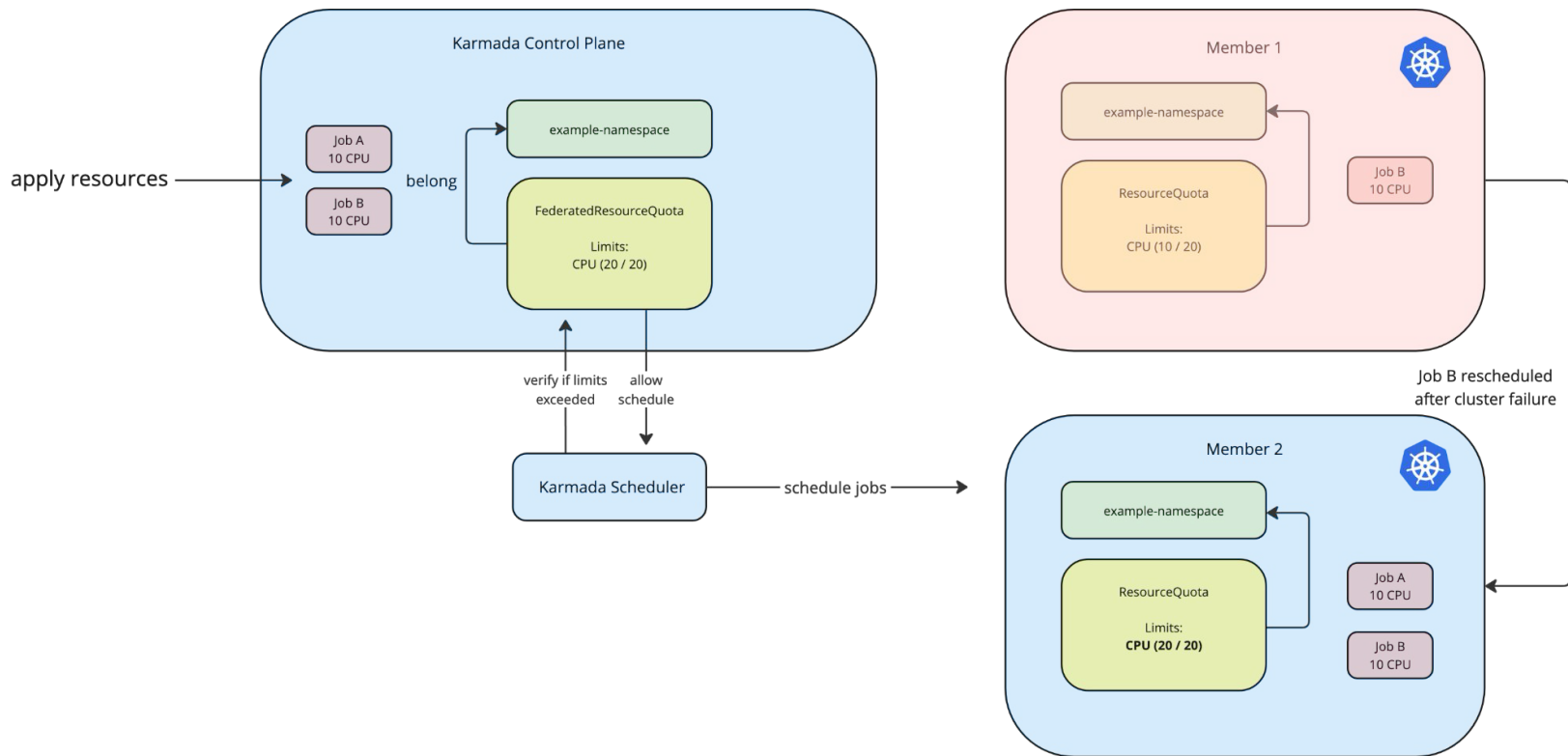


KubeCon



CloudNativeCon

Europe 2026



Workload Anti-Affinity



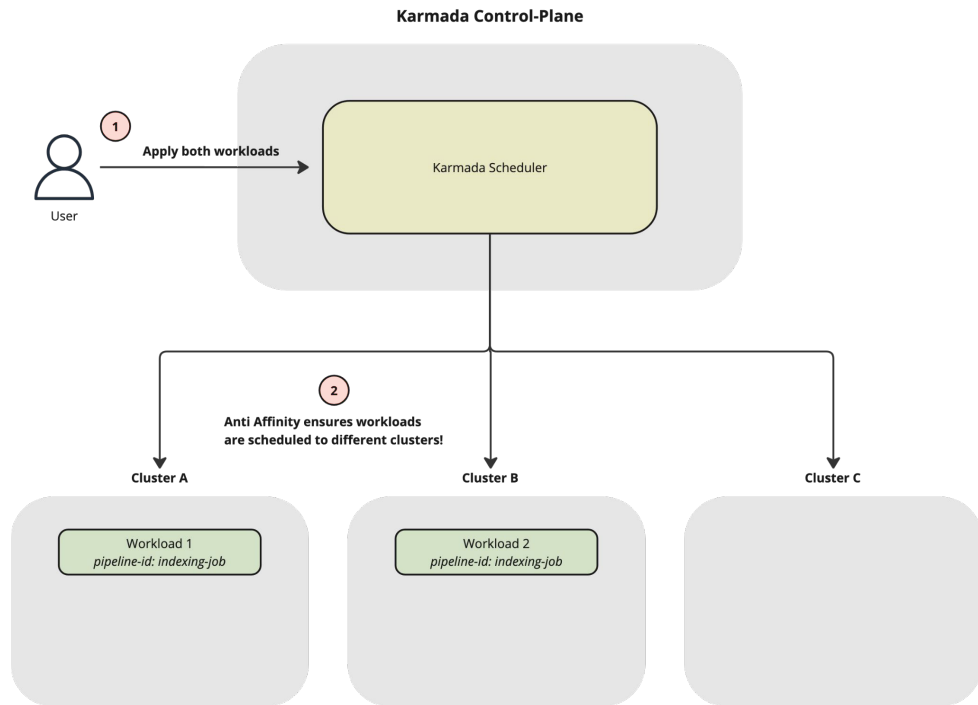
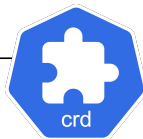
KubeCon



CloudNativeCon

Europe 2026

```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: workload-propagation-policy
  namespace: user-ns
spec:
  resourceSelectors:
    - apiVersion: flink.apache.org/v1beta1
      kind: FlinkDeployment
      labelSelector:
        matchLabels:
          ha.enabled: "true"
  placement:
    clusterAffinity:
      clusterNames:
        - cluster-a
        - cluster-b
        - cluster-c
    workloadAffinity:
      antiAffinity:
        groupByLabelKey: pipeline-id
```



Workload Affinity



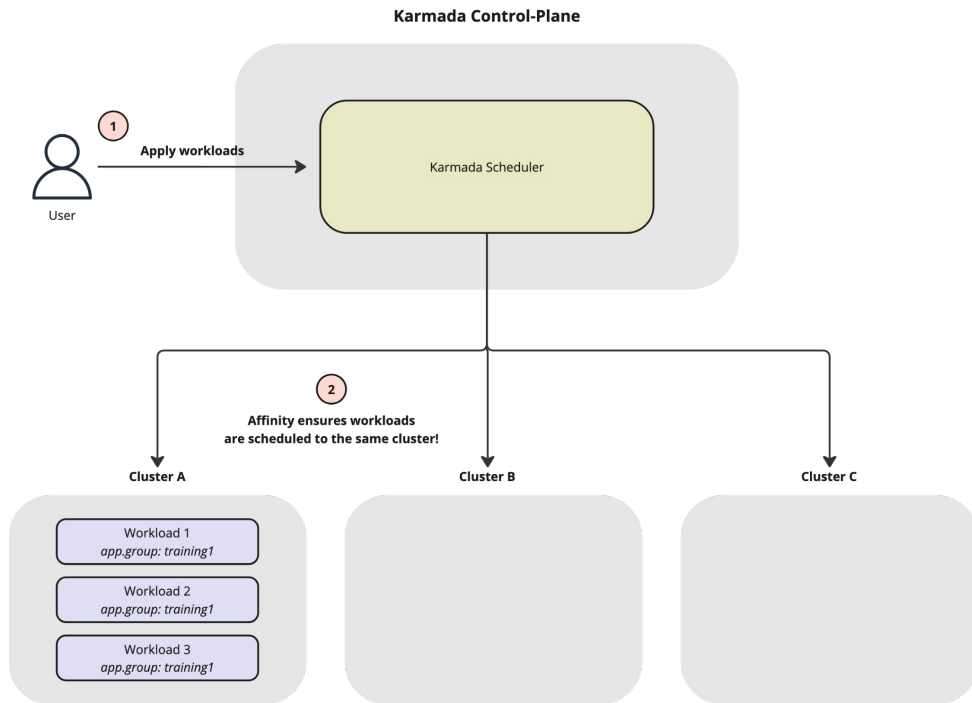
KubeCon



CloudNativeCon

Europe 2026

```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: workload-propagation-policy
  namespace: user-ns
spec:
  resourceSelectors:
  - apiVersion: batch/v1
    kind: Job
    labelSelector:
      matchLabels:
        workload.type: training
  placement:
    clusterAffinity:
      clusterNames:
      - cluster-a
      - cluster-b
      - cluster-c
    workloadAffinity:
      affinity:
        groupByLabelKey: app.group
```



Multi-Components Workloads Scheduling



KubeCon



CloudNativeCon

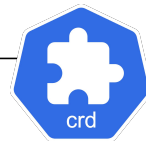
Europe 2026

Before v1.16, Karmada's scheduler assumed all replicas in a workload were identical. But this assumption does not hold for many CRDs!

For instance, examine a **FlinkDeployment**:

- Use JM for replica, we *underestimate* total resource usage!
- Use TM for replica, we *overestimate* total resource usage!
- Either way we get inaccurate federated resource quota usage

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: example-flink-job
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  job:
    parallelism: 2
    state: running
  jobManager:
    resource:
      cpu: 1
      memory: 1024m
  taskManager:
    resource:
      cpu: 4
      memory: 6Gi
```



Multi-Components Workloads Scheduling

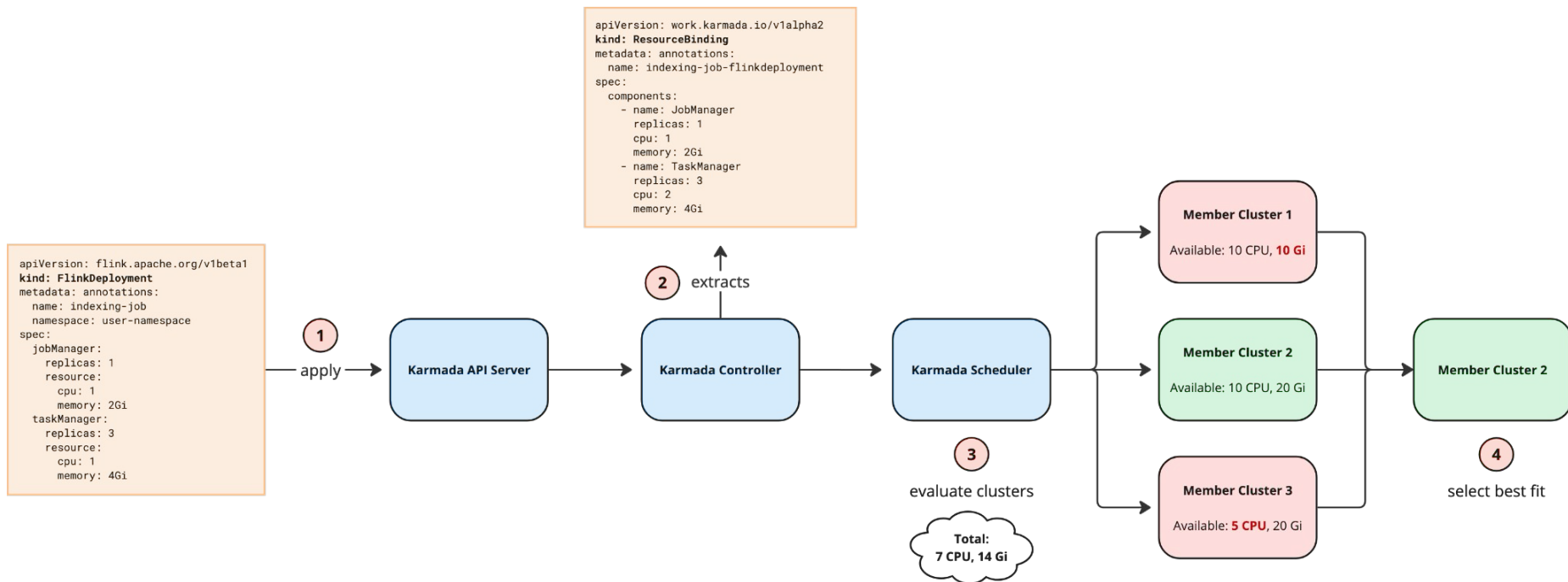


KubeCon



CloudNativeCon

Europe 2026



Priority-Based Scheduling



KubeCon



CloudNativeCon

Europe 2026

When multiple workloads compete for scheduling, priority scheduling ensures **critical workloads get scheduled first**.

1. Create a standard k8s *PriorityClass* with a numeric priority value.
2. In *PropagationPolicy*, add a *schedulePriority* field referencing that *PriorityClass*.
3. Karmada scheduler queues bindings by priority: high → low, FIFO for equal.
4. If no *PriorityClass* specified → priority = 0.



```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: critical-app-policy
  namespace: default
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: critical-nginx # name of the workload to propagate
  schedulePriority:
    priorityClassSource: KubePriorityClass
    priorityClassName: high-priority-example
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
```

Priority-Based Scheduling

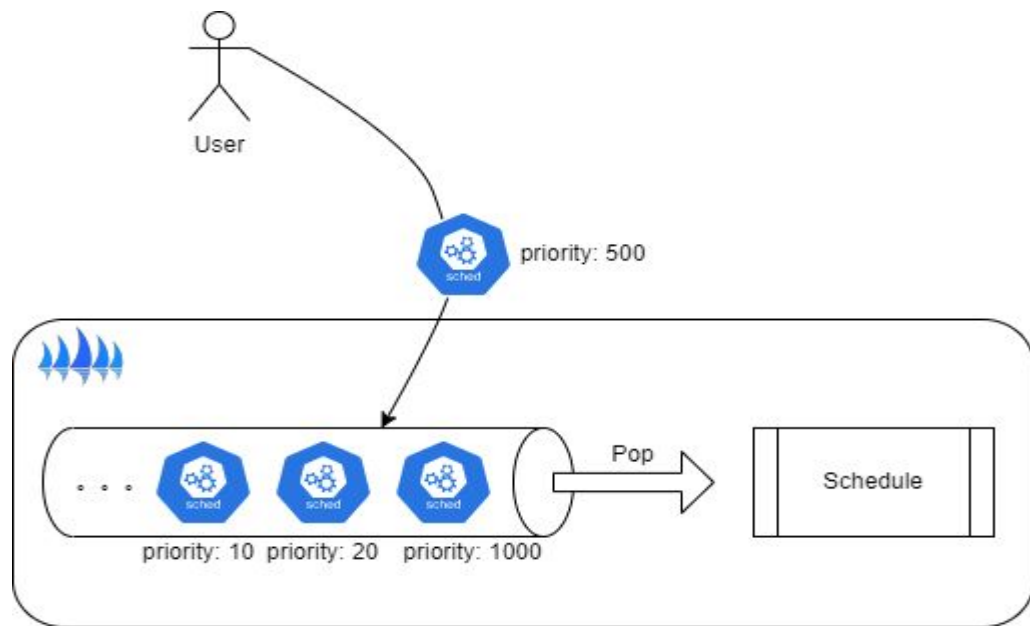


KubeCon



CloudNativeCon

Europe 2026



```
apiVersion:
policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: critical-app-policy
  namespace: default
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: critical-nginx
  # The name of the workload to
  # propagate
  schedulePriority:
    priorityClassSource:
      KubePriorityClass
    priorityClassName:
      high-priority-example #
  # Reference to the PriorityClass
  # defined above
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
```



KubeCon



CloudNativeCon

Europe 2026

Real-World Use Cases





KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Karmada at Bloomberg

Model Cache & Jupyter Notebook



Model Cache



KubeCon



CloudNativeCon

Europe 2026

Caching a model on every node to save startup time.

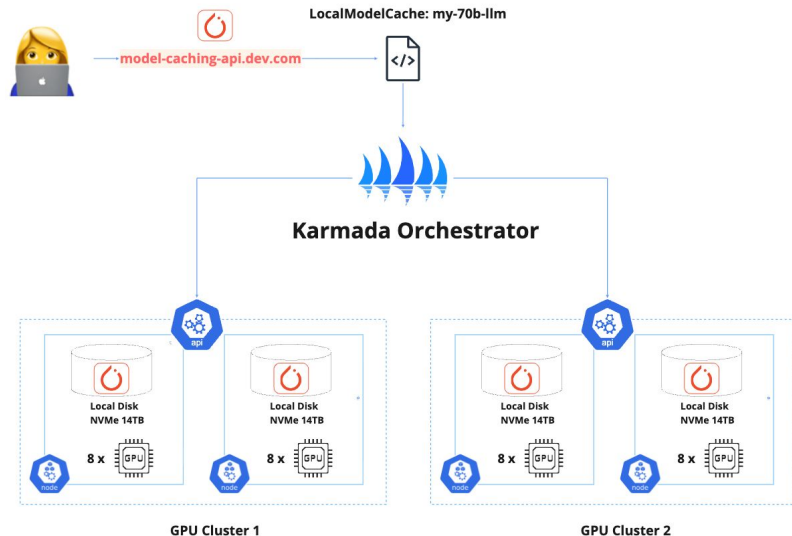
- **Model Cache API on top of Karmada**

A thin API layer sits above Karmada: request → API → propagate to all nodes and clusters

- **Every node caches, every cluster covered**

Each cluster member pulls the model to local node storage – zero per-cluster manual steps.

```
{
  "createdAt": "2025-02-27T20:04:49Z"
  "gpuTypes": ["gpu-cv", "gpu-nlp"],
  "modelSize": "100Gi",
  "modelName": "llm-70b",
  "modelSourceUri": "s3://models/llm-70b"
  "status": {
    "failedNodes": 0,
    "readyNodes": "NodeDownloaded"
  },
  "tier": "dev"
}
```



Model Cache



KubeCon



CloudNativeCon

Europe 2026

Caching a model on every node to save startup time.

- **Model Cache API on top of Karmada**

A thin API layer sits above Karmada: request → API → propagate to all nodes and clusters.

- **Every node caches, every cluster covered**

Each cluster member pulls the model to local node storage — zero per-cluster manual steps.

Why Karmada for Model Cache?



KubeCon



CloudNativeCon

Europe 2026

Karmada's multi-cluster federation is the natural fit for model caching at scale.

1

Multi-Cluster Propagation

Karmada natively fans out resources to all registered clusters – no custom fan-out logic or per-cluster scripts needed.

Zero fan-out boilerplate

2

Flexible Placement Control

PropagationPolicy targets caches by cluster profile, GPU availability, region, or any label — decide where each model lives.

Policy-driven placement

3

Metrics-Driven Caching (Roadmap)

Future: feed inference traffic metrics into scheduling decisions to prioritize caching where demand is highest.

Demand-aware scheduling

Out-of-the-Box Karmada Features



KubeCon



CloudNativeCon

Europe 2026

The ModelCache CRD gets these Karmada capabilities for free.

Global Status

Query model cache state across all clusters from a single Karmada control plane – no per-cluster kubectl needed.

Aggregated Status

AggregatedStatus in ResourceBinding surfaces per-cluster state in one API call – instant federation-wide health view.

Disaster Recovery

When a cluster fails, Karmada auto-detects and reconciles – model cache rescheduled to healthy clusters automatically.

Dynamic Reschedule

Update a PropagationPolicy and Karmada reschedules the model cache to new target clusters without any manual steps.

Jupyter Notebook



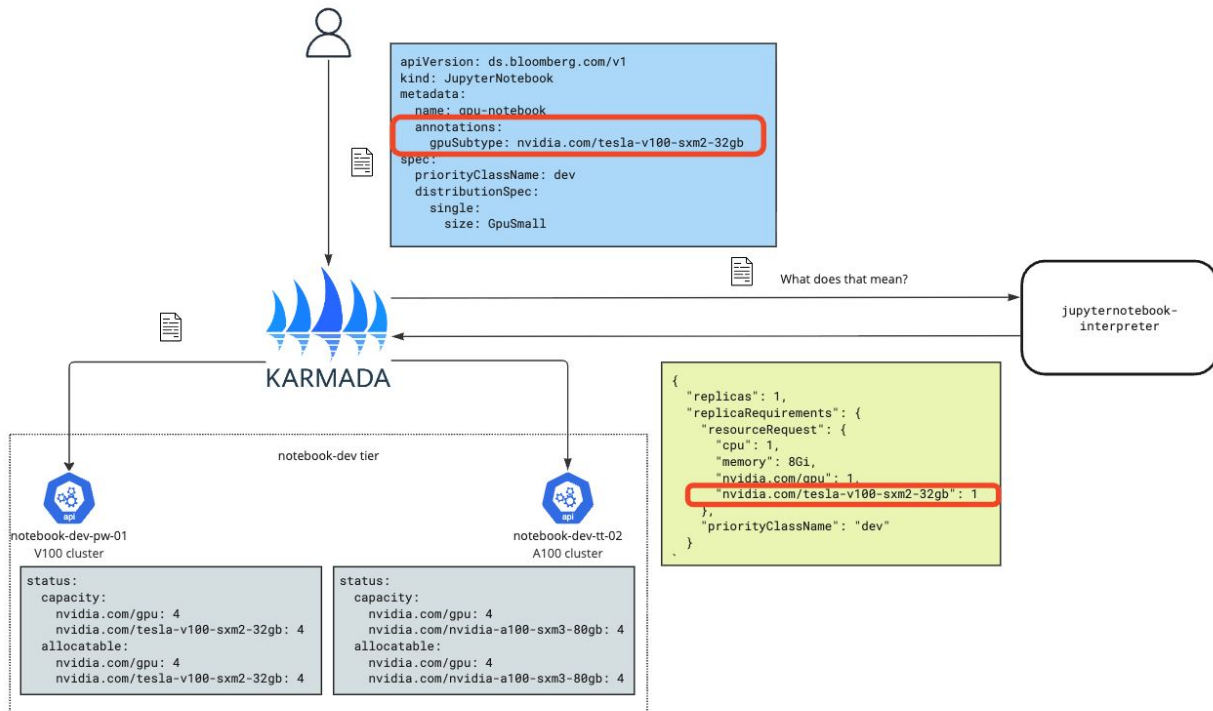
KubeCon



CloudNativeCon

Europe 2026

Scheduling a Jupyter Notebook on any cluster that has available resources.



Why Karmada for Jupyter Notebooks?



KubeCon



CloudNativeCon

Europe 2026

Intelligently placing a Jupyter Notebook given resource requirements.

1

Smart Propagation

Karmada natively picks a suitable cluster to federate the resource to – no custom scheduling logic or per-cluster scripts needed.

Zero scheduling logic

2

Flexible Placement Control

PropagationPolicy targets by cluster profile, GPU availability, region, or any label – decide where a notebook lives.

Policy-driven placement

Out-of-the-Box Karmada Features



KubeCon



CloudNativeCon

Europe 2026

The JupyterNotebook CRD gets these Karmada capabilities for free.

Global Status

Query JupyterNotebook status across all clusters from a single Karmada control plane – no per-cluster kubectl needed.

Aggregated Status

AggregatedStatus in ResourceBinding surfaces per-cluster state in one API call – instant federation-wide health view.

Disaster Recovery

When a cluster fails, Karmada auto-detects and reconciles – model cache rescheduled to healthy clusters automatically.



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Karmada In Trip.com

**Cluster Baseline Rollout, Resilient Architecture,
and Smooth Migration**



What We Built with Karmada



KubeCon



CloudNativeCon

Europe 2026

Karmada became the policy-driven layer for rollout, resilience, and migration across member clusters.

1

Automated Baseline Rollout

Roll out operators, monitoring, and logging consistently after a new cluster is provisioned.

Faster onboarding

2

Resilient Architecture

Keep services available when a member cluster fails through traffic failover and local HPA recovery.

Higher availability

3

Smooth Migration

Shift workloads by policy weights instead of asking application owners to redeploy manually.

Multi-cluster elasticity

Automated Baseline Rollout



KubeCon

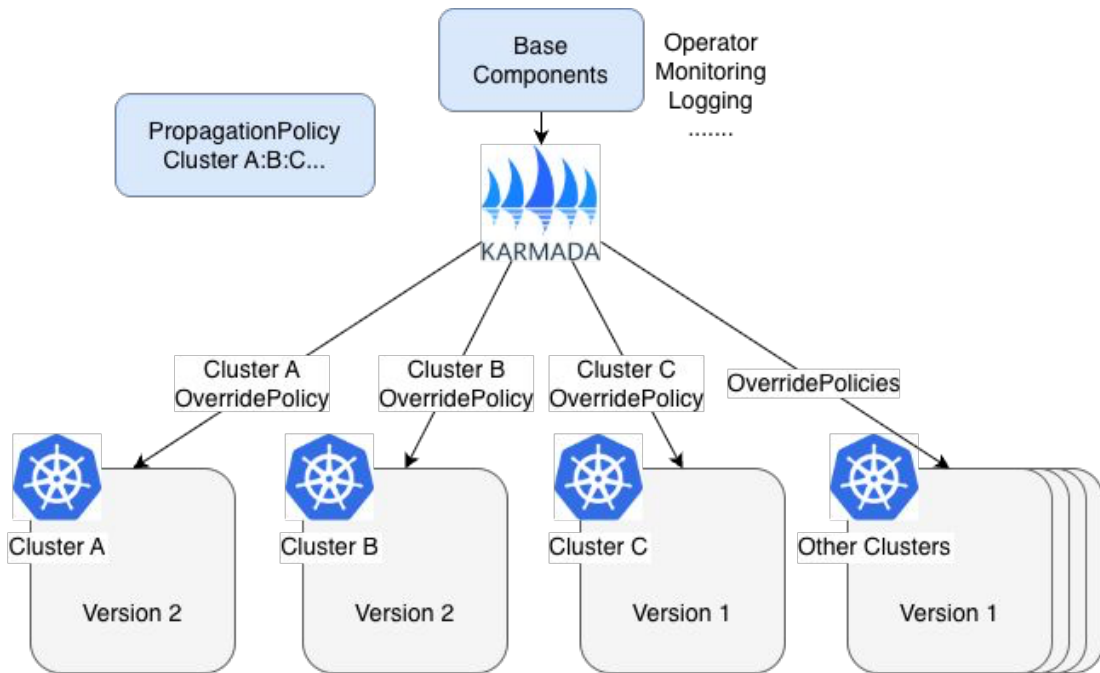


CloudNativeCon

Europe 2026

- New clusters were previously slowed down by manual rollout of operators, monitoring, and other platform add-ons
- With Karmada, platform teams update policy once and distribute the baseline components in a unified way.
- Faster baseline readiness and much less cross-team coordination for every new cluster.

**< 4 hour
new-cluster baseline ready**



Resilient Architecture



KubeCon

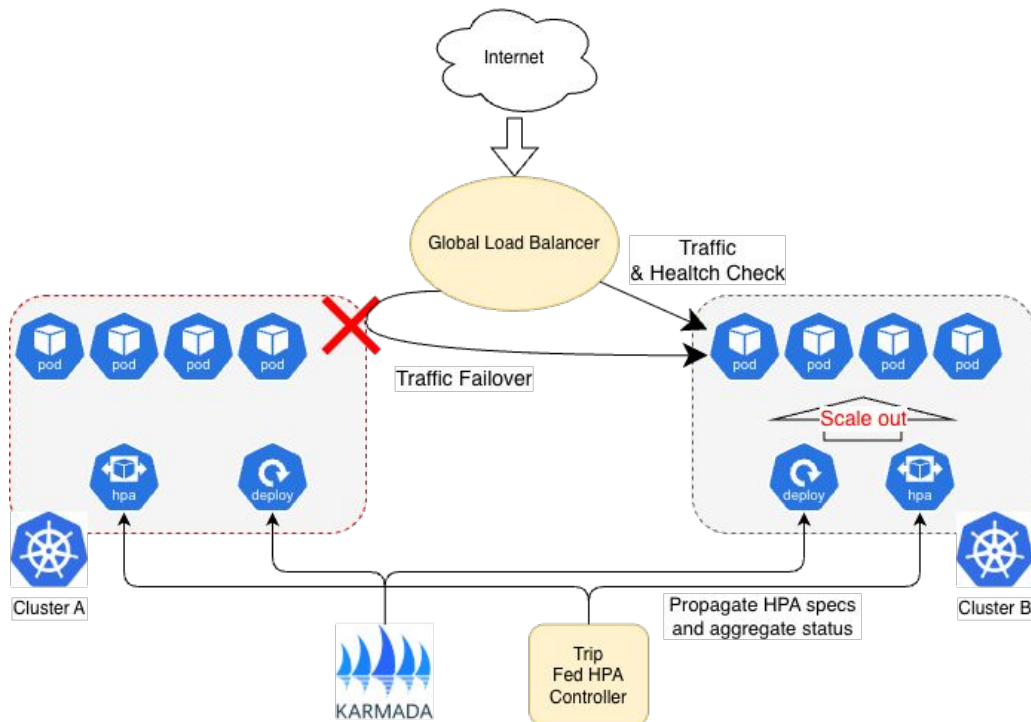


CloudNativeCon

Europe 2026

- Stateless applications run across two or more member clusters, and each cluster keeps its own HPA active.
- When one cluster fails, traffic fails over to healthy clusters based on health checks.
- At the same time, local HPA scales out in the surviving clusters to complete self-healing.
- Even if the federation control plane is briefly unavailable, normal workload scaling continues.

**Traffic failover + local HPA
= service continuity**



Smooth Cross-Cluster Migration



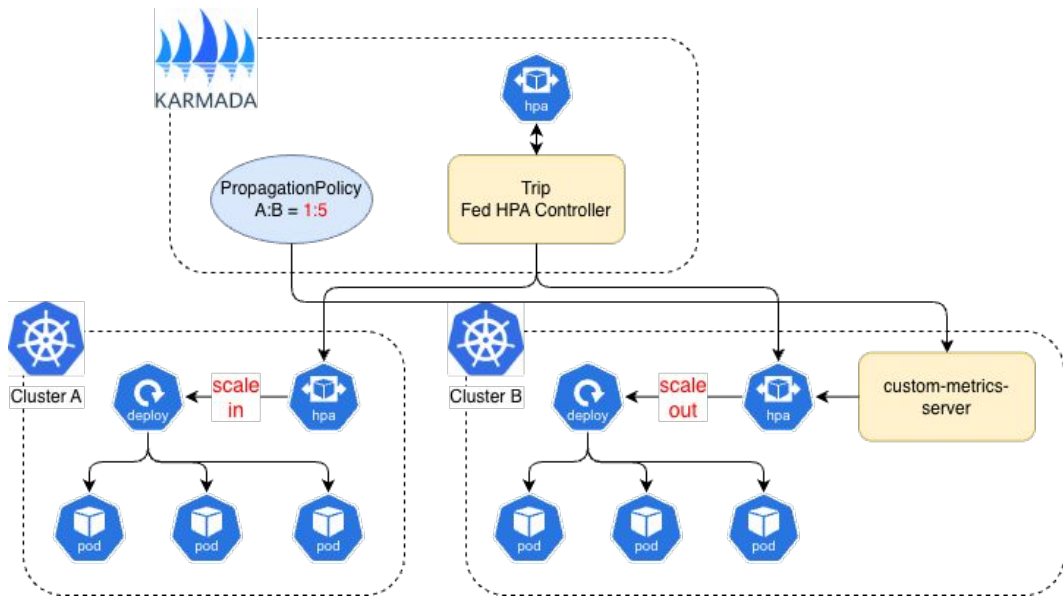
KubeCon



CloudNativeCon

Europe 2026

- Some production clusters grew too big, splitting workloads into new clusters became a frequent operational task.
- Instead of asking application owners to redeploy, operators adjust weights in Karmada PropagationPolicy.
- The target cluster scales out gradually while the source cluster scales in, enabling smooth migration with minimal service disruption.



300K pods migrated

Impact with Karmada



KubeCon



CloudNativeCon

Europe 2026

These capabilities are already running across multiple regions in production.

< 4 hour

Baseline components ready after a new cluster is provisioned.

**40+
clusters**

Baseline components consistently distributed with Karmada.

**300K
pods**

Workloads moved stably across clusters using rebalance.

**Failure
isolated**

Temporary federation control-plane issues do not block normal workload scaling.

- **Milestones**

- **2021.04** Open Source
- **2021.09** CNCF Sandbox project
- **2023.12** CNCF Incubating project

- **Community**

- 5K+ GitHub Stars
- 900+ contributors
- 44 Public Adopters (In production)
- 8 Cloud Vendors

- **SIG-Scalability**

- **v1.14** Controller code start 7min -> 4min
- **v1.15** Scheduler QPS +46%
- **v1.16** Large Scale processing

- **SIG-UI (Dashboard)**

- **2025.02** v0.1.0
- **2025.10** v0.2.0
- **2026.03** v0.3.0

Public Adopters In Production



KubeCon



CloudNativeCon

Europe 2026



Bloomberg

Engineering



中科类脑



中国电子



中国电子云



DaoCloud



中国移动



移动云



擘文集团



HUAWEI



艾莫尔研究院
AIML INSTITUTE



iFLYTEK



iQIYI 爱奇艺



智能软件研究中心
Intelligent Software Research Center



京东云



家家悦 JIAJIAYUE



Kwai



理想



Hong Kong
LONGBRIDGE



马蜂窝



NCCN
HIGH-PERFORMANCE HOSTING



网易 NETEASE



360



rednote



Shopee



sina 新浪



SK



同程旅行



Trip.com



联通云
Unicom Cloud



SCATTER LAB



商汤
sensetime



数之联



VIP KID



vivo



金山办公
KINGSOFT OFFICE



有赞



之江实验室
ZHEJIANG LAB



中通快递
ZTO EXPRESS

RoadMap 2026

- ~~Priority-Based Scheduling~~ and Preemption
- Multi-cluster Queue
- Extended Cluster Affinities
- Precise Cluster Resource-Aware Scheduling
- More



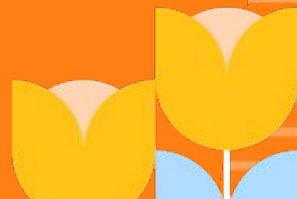
KubeCon



CloudNativeCon

Europe 2026

Q&A





KubeCon



CloudNativeCon

Europe 2026

