
KubeCon
CloudNativeCon
North America 2023

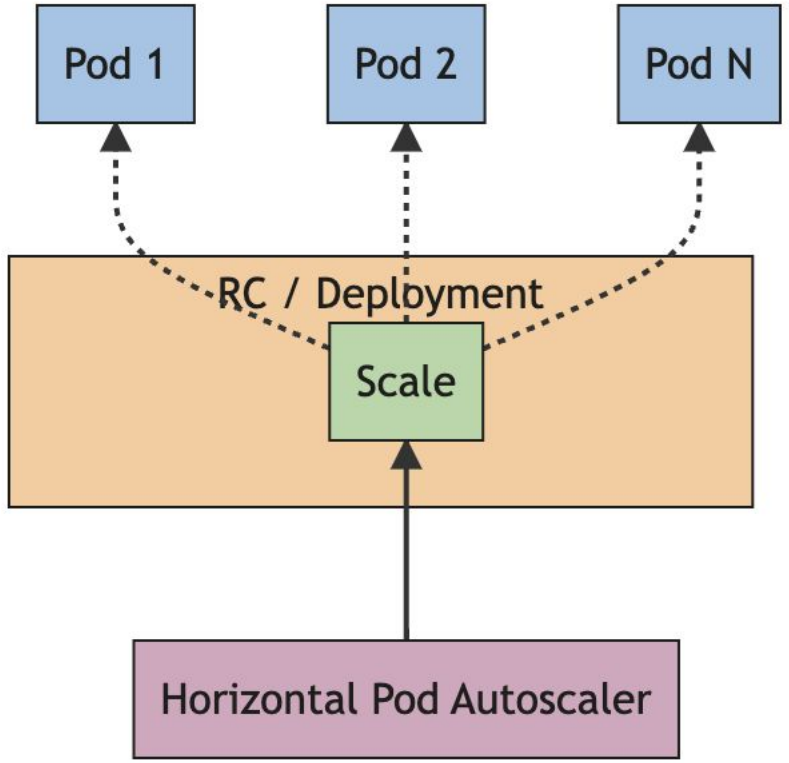
Break Through Cluster Boundaries to Autoscale Workloads Across Them on a Large Scale

XingYan Jiang, DaoCloud & Ying Zhang, Zendesk

Agenda

- HPA overview and benefits of autoscaling across clusters
- Introduction to Karmada
- Karmada feature - FederatedHPA
- Karmada new feature - CronFederatedHPA
- Q&A

HPA overview



Horizontal Pod Autoscaling (HPA) is a mechanism for automatically scaling the number of Pod replicas in a Kubernetes cluster to dynamically scale an application based on current load metrics.

HPA overview

HPA example:

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

The HPA controller will increase or decrease the number of Deployment replicas (by updating the Deployment) to maintain an average CPU utilization of 50% for all Pods.

Benefits of autoscaling across clusters

- **Unified** management of autoscaling operation across clusters
- Break through the **resource limitations** of a single cluster
- With **rich strategies** in API definition, scale workloads across multiple clusters, and meet different scenarios
- Cluster-level autoscaling for disaster recovery.

Introduction to Karmada

What is Karmada?

Karmada (Kubernetes Armada) is a Kubernetes management system that enables you to run your cloud-native applications across **multiple Kubernetes clusters** and **clouds**, with no changes to your applications. By speaking **Kubernetes-native APIs** and providing **advanced scheduling capabilities**, Karmada enables truly open, multi-cloud Kubernetes.

Karmada aims to provide turnkey automation for multi-cluster application management in multi-cloud and hybrid cloud scenarios, with key features such as **centralized multi-cloud management, high availability**, **failure recovery**, and **traffic scheduling**.

Introduction to Karmada

Build an infinitely scalable container resource pool using Karmada.

Enable developers to use multi-cloud as easily as using a K8s cluster.

Kubernates Native API Compatible

Open and Neutrality

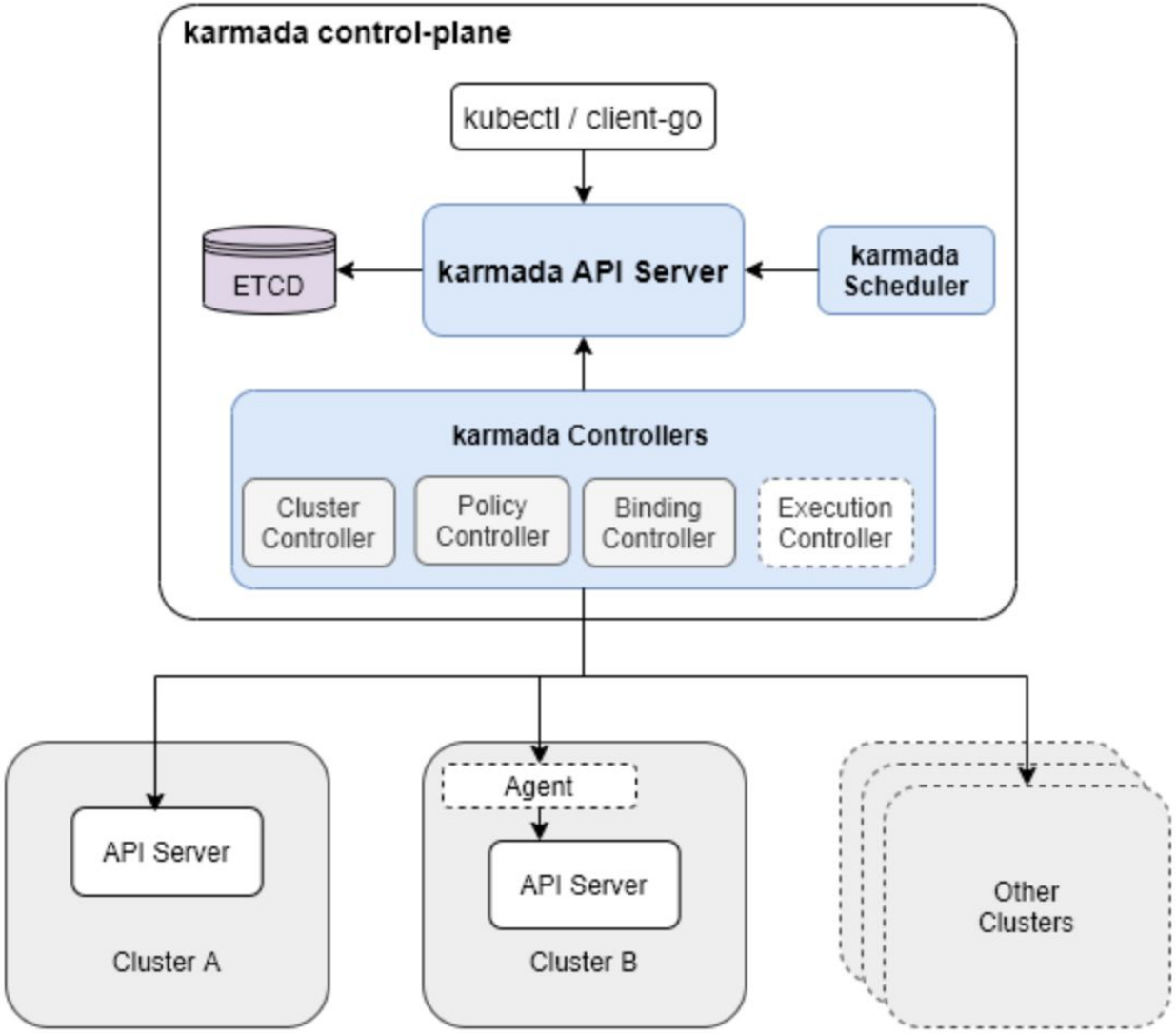
Avoid Vendor Lock-in

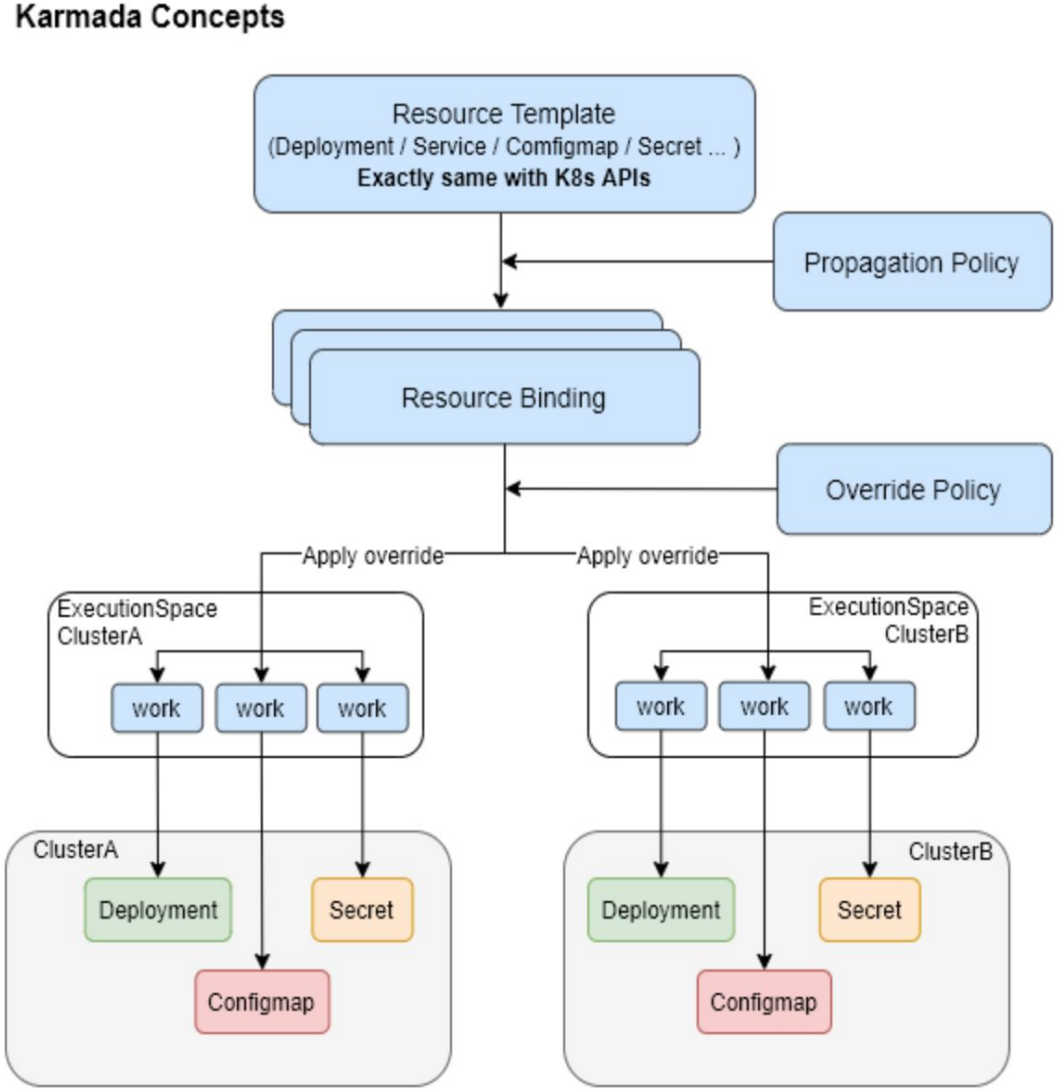
Out of the Box

Fruitful Scheduling Policies

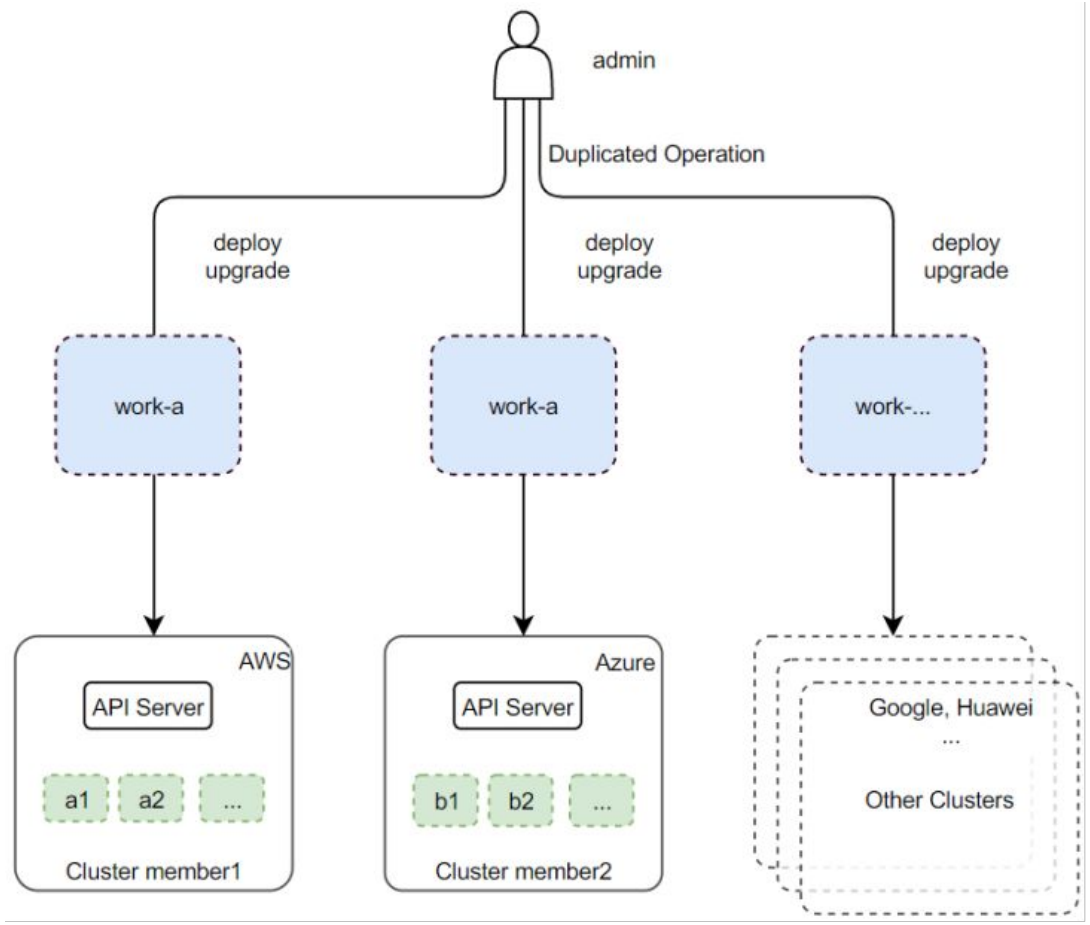
Centralized Management

Karmada Deep Dive

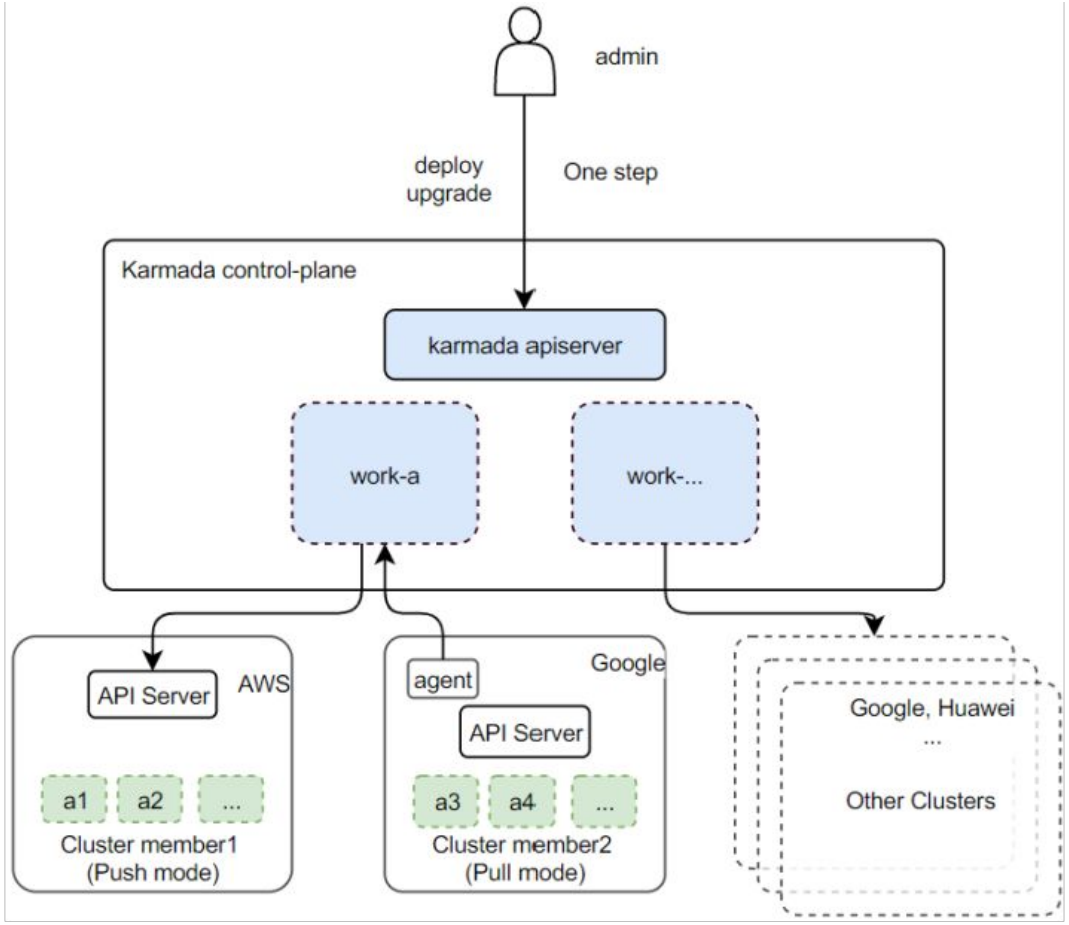




Multi-Cluster Management

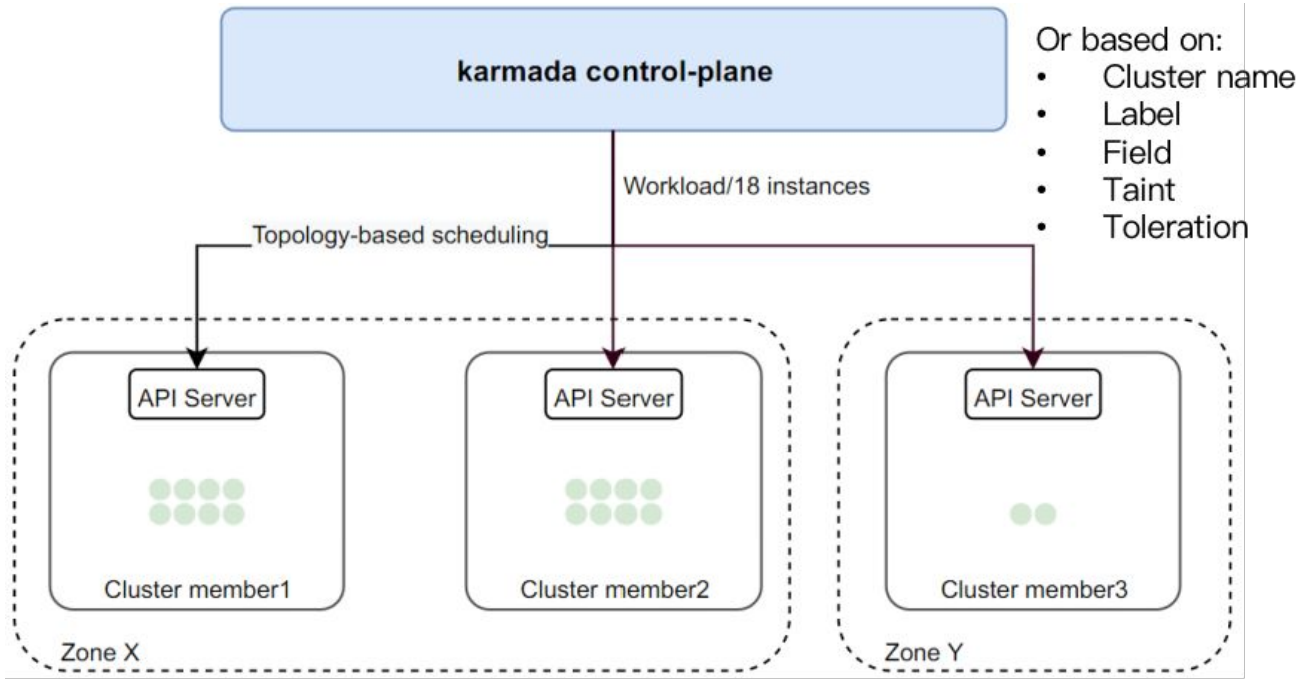


Repeated management operations

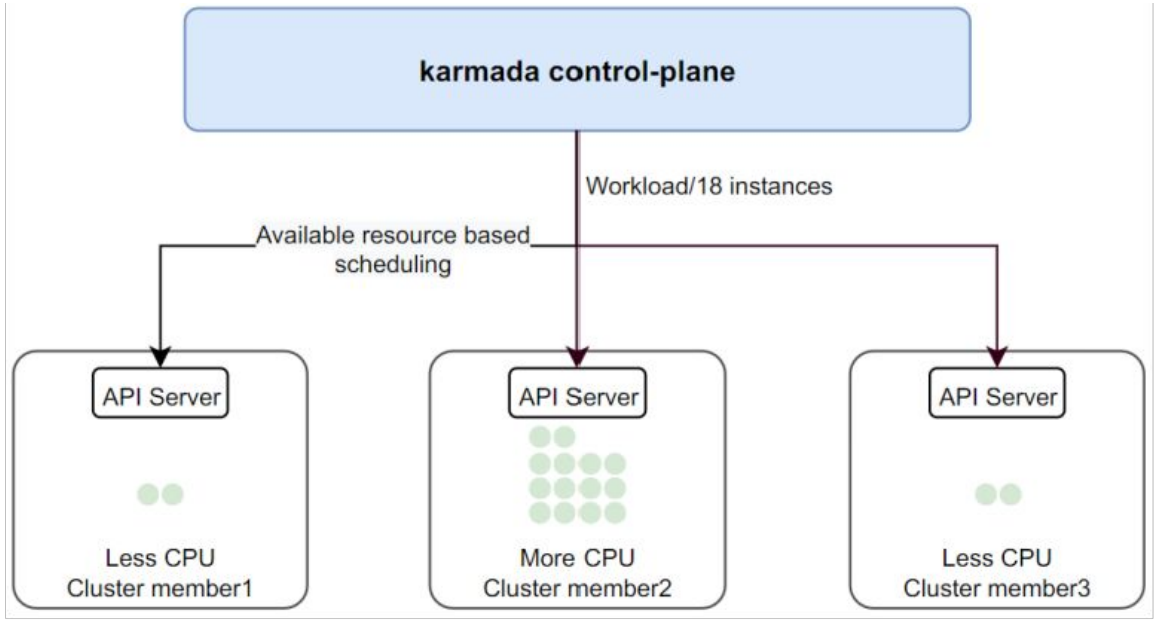


Manage multi-cluster workloads in one step

Workload Propagation Across Clusters



Schedule based on topology



Schedule based on available resources

Advanced policies meet various scheduling requirements

PopagationPoliy

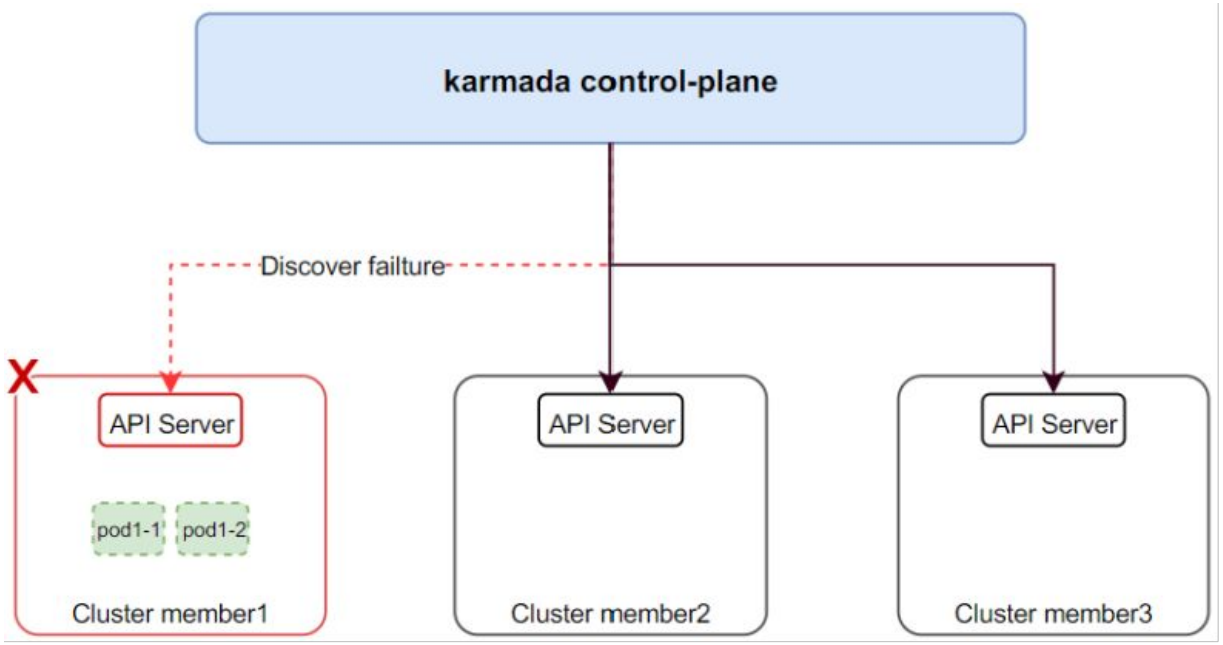

```yaml
# propagationpolicy.yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: example-policy # The default namespace is `default`.
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  placement:
    clusterAffiniy:
      clusterNames:
        - member1
        - member2
    replicaScheduling:
      replicaDivisionPreference: Weighted
      replicaSchedulingType: Divided
```

Two replicaSchedulingTypes

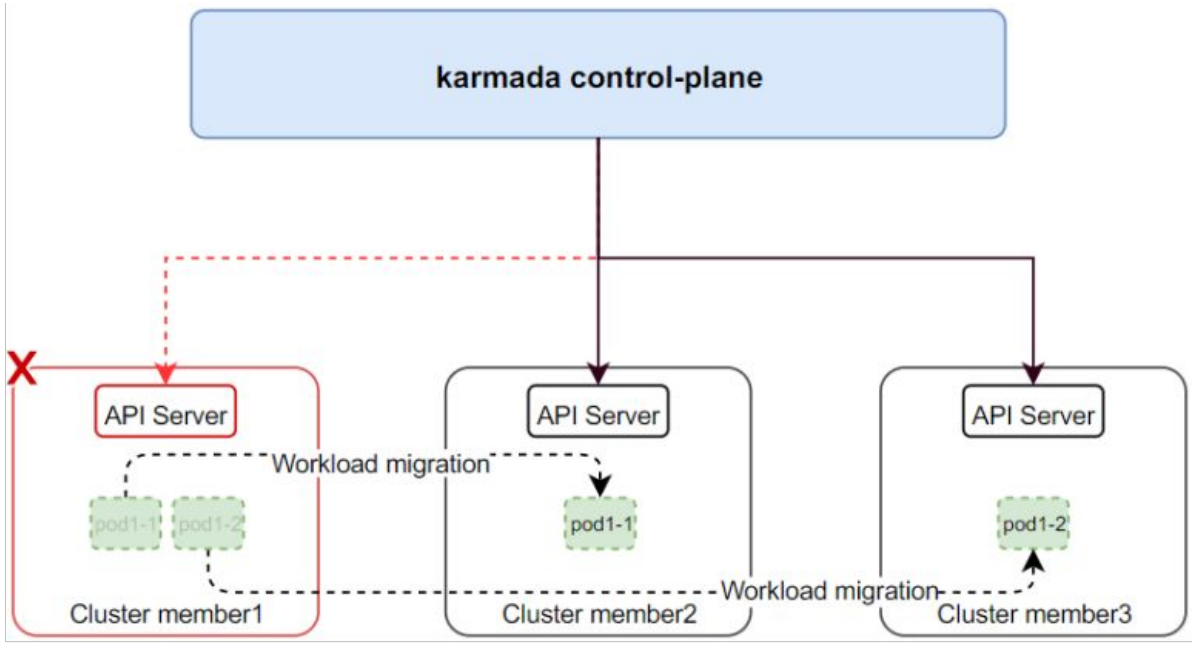
- Duplicated
- Divided

Refer to: https://karmada.io/docs/userguide/scheduling/resource-propagating/#multiple-strategies-of-replica-scheduling

Cross-Cluster Application Failover



Faulty cluster loses connection



Gracefully migrate workloads

Graceful migration ensures uninterrupted services

Agenda

- HPA overview and benefits of autoscaling across clusters
- Introduction to Karmada
- **Karmada feature - FederatedHPA**
- Karmada new feature - CronFederatedHPA
- Q&A

FederatedHPA


```go
type FederatedHPA struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec FederatedHPASpec `json:"spec"`

    Status autoscalingv2.HorizontalPodAutoscalerStatus `json:"status"`
}

type FederatedHPASpec struct {
    ScaleTargetRef autoscalingv2.CrossVersionObjectReference `json:"scaleTargetRef"`

    MinReplicas *int32 `json:"minReplicas,omitempty"

    MaxReplicas int32 `json:"maxReplicas"

    Metrics []autoscalingv2.MetricSpec `json:"metrics,omitempty"`

    Behavior *autoscalingv2.HorizontalPodAutoscalerBehavior `json:"behavior,omitempty"`
}
```

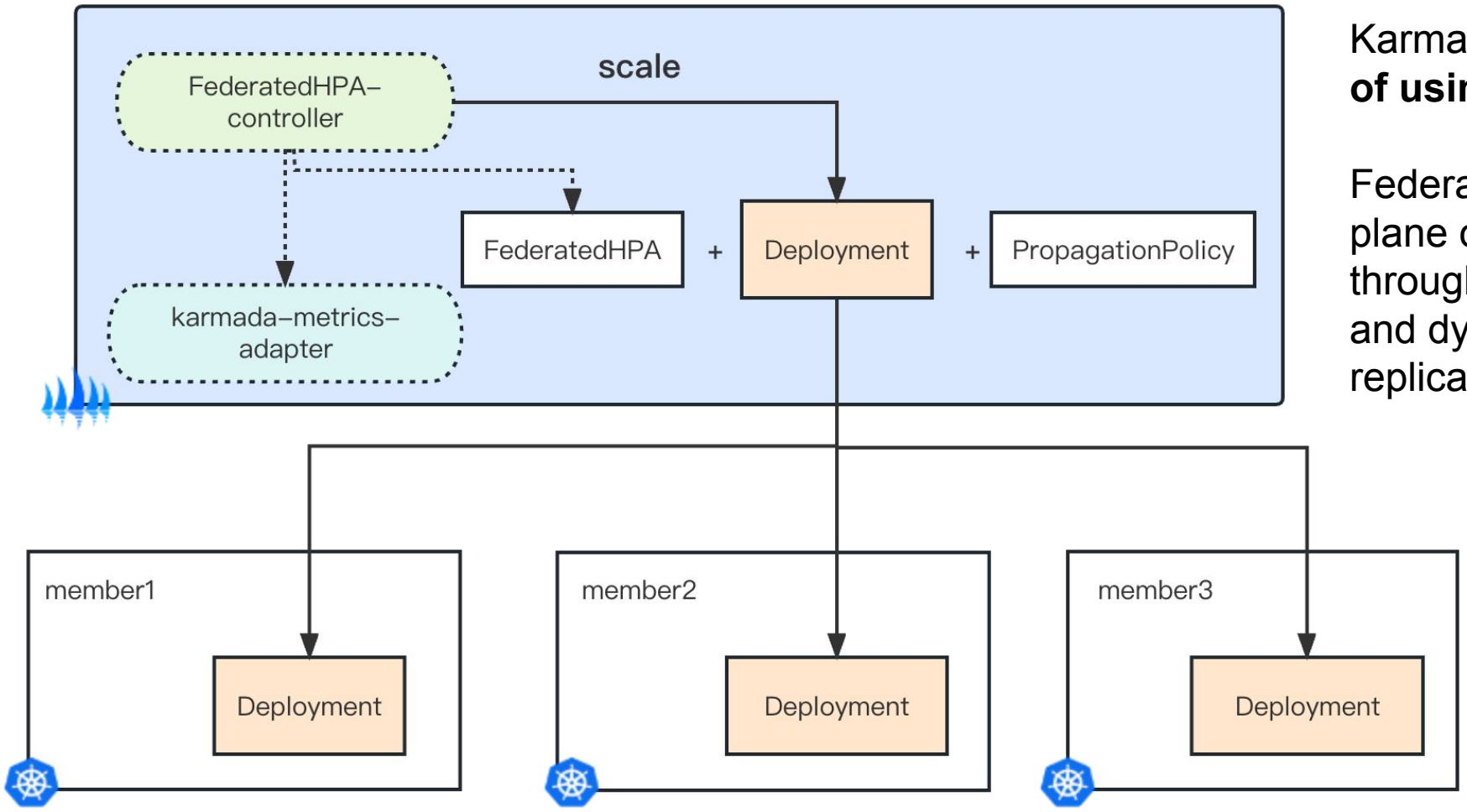
Keep **consistent** with the core API of K8s native HPA. It can be **migrated** to FederatedHPA at a very low cost.

FederatedHPA


```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```


```yaml
apiVersion: autoscaling.karmada.io/v1alpha1
kind: FederatedHPA
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

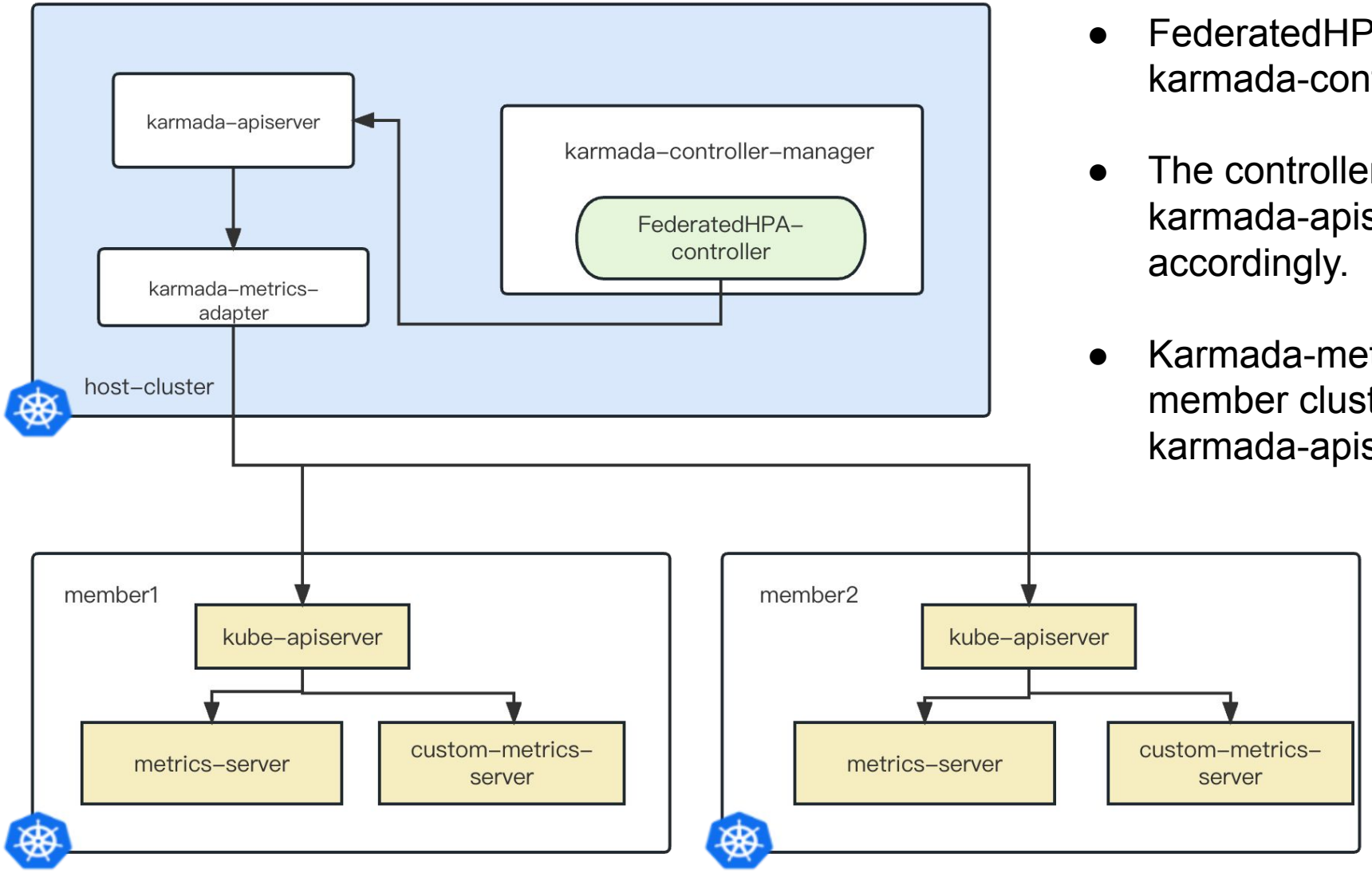
FederatedHPA workflow



Karmada FederatedHPA - **Experience of using single-cluster HPA.**

FederatedHPA controller on the control plane obtains metrics of Deployment through karmada-metrics-adapter, and dynamically scales the number of replicas for the Deployment.

FederatedHPA workflow



- FederatedHPA-controller is located in the karmada-controller-manager component.
- The controller queries metrics from karmada-apiserver and scale deployment accordingly.
- Karmada-metrics-adapter queries metrics from member clusters, aggregates them and returns to karmada-apiserver.

CronFederatedHPA

Extension - CronFederatedHPA

CronFederatedHPA is used for regular autoscaling actions. It can scale workloads that have a scale subresource or FederatedHPA.

Scenarios

- Proactively scale up a workload for predictable spikes in traffic
- Schedule the non time-sensitive workloads on weekends
- And more

CronFederatedHPA

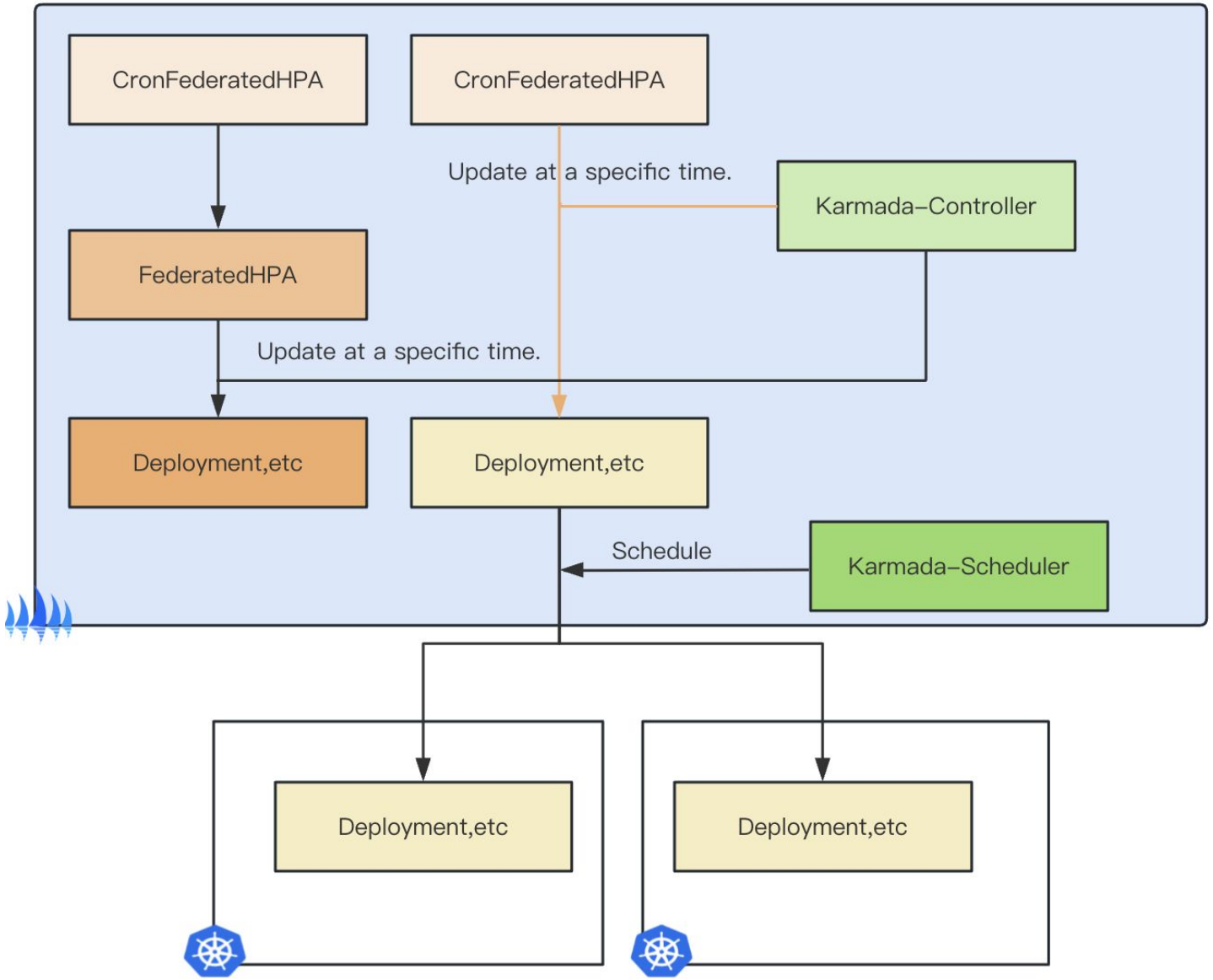

```yaml
apiVersion: autoscaling.karmada.io/v1alpha1
kind: CronFederatedHPA
metadata:
  name: cron-federated-hpa
spec:
  scaleTargetRef:
    apiVersion: autoscaling.karmada.io/v1alpha1
    kind: FederatedHPA
    name: shop-fhpa
  rules:
  - name: "Scale-Up"
    schedule: "30 08 * * *"
    targetMinReplicas: 1000
  - name: "Scale-Down"
    schedule: "0 11 * * *"
    targetMinReplicas: 1
```

Scale FederatedHPA

```yaml
apiVersion: autoscaling.karmada.io/v1alpha1
kind: CronFederatedHPA
metadata:
  name: nginx-cronfhpa
  namespace: default
spec:
   scaleTargetRef:
      apiVersion: apps/v1
      kind: Deployment
      name: nginx
   rules:
  - name: "Scale-Up"
    schedule: "30 08 * * *"
    targetReplicas: 1000
  - name: "Scale-Down"
    schedule: "0 11 * * *"
    targetReplicas: 1
```

Scale Deployment,etc

CronFederatedHPA workflow



CronFederatedHPA with FederatedHPA

Scaling workloads with scale subresource

CronFederatedHPA is capable of scaling workloads that have a **scale subresource**, such as Deployment and StatefulSet.

Scaling FederatedHPA

CronFederatedHPA scales resources at a specified time. When a workload is scaled only by CronFederatedHPA, it can't handle more requests until that time. To **proactively** meet peak loads and real-time demands, we recommend starting with **CronFederatedHPA to scale FederatedHPA,** and then**, FederatedHPA can scale workloads** based on their metrics.

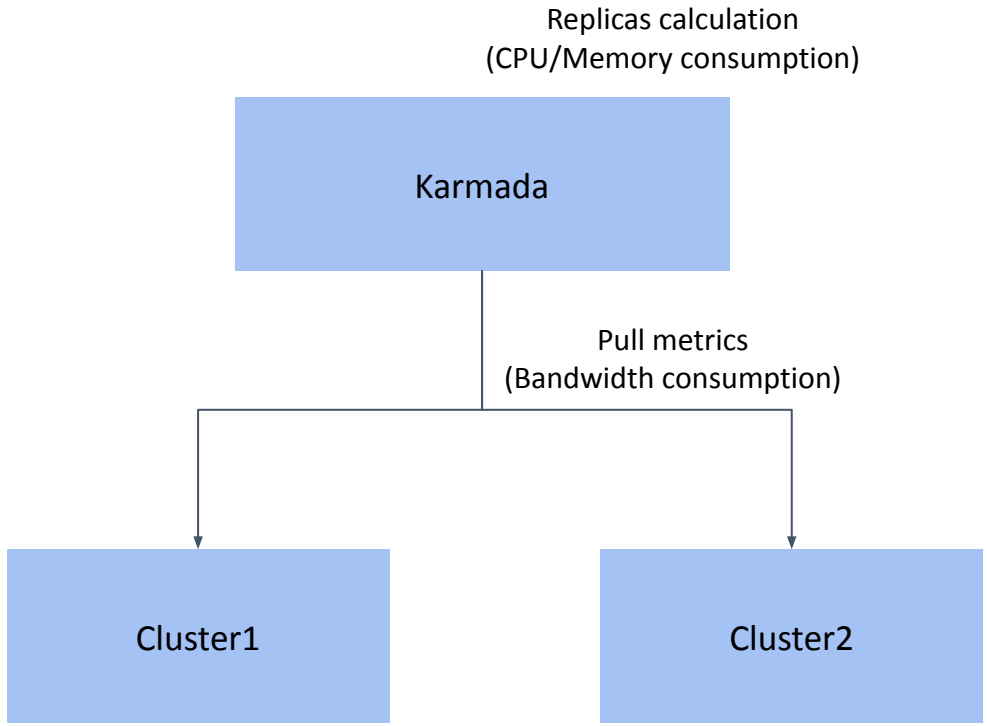
Advantages and Notice

Advantages of FederatedHPA:

- The API and K8s native HPA are almost identical, with the same user experience and low migration cost.

Notice for using FederatedHPA:

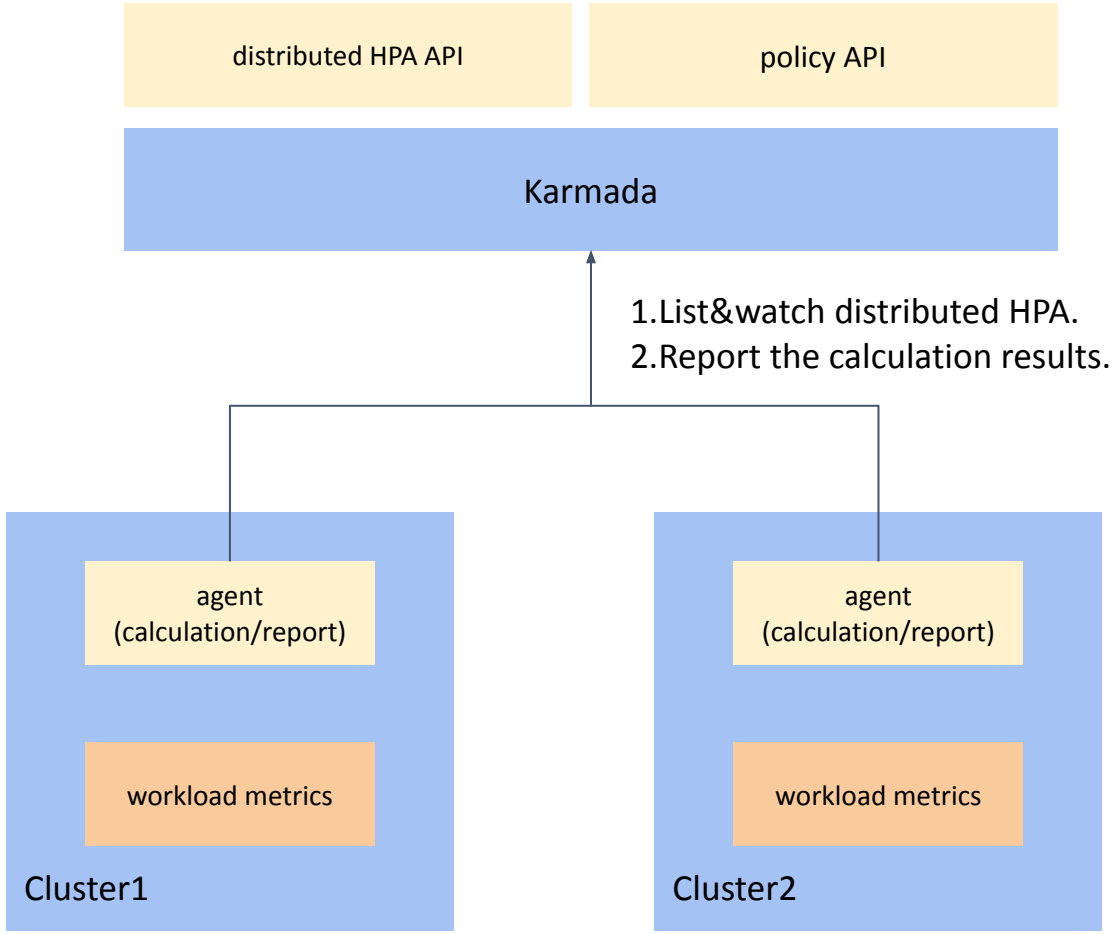
- FederatedHPA is a type of centralized multi-cluster HPA.
- When scaling concurrently on a large scale, a larger bandwidth is required (maximum concurrent scaling metrics pull: 500M bandwidth, 200,000 Pods).
- Storing and computing corresponding data requires much CPU/Memory.



Explore distributed autoscaling

Design :

- Define the metrics for scaling in DistributedHPA API.
- Member cluster **agents will list&watch the distributed HPA** and calculate the intermediate calculation state.
- Update the calculated intermediate state to the "status" of distributed HPA.
- Karmada processes data based on reports from member clusters, **calculates final results, and adjusts replicas**.
- Based on policy configuration and **real-time states** of clusters, Karmada schedules scaled instances to member clusters to achieve cross-cluster autoscaling.



Key Takeaways

- Regular HPA cannot break through cluster boundaries to scale workloads
- FederatedHPA and CronFederatedHPA can autoscale workloads across multiple clusters
- Karmada enables you to run your cloud-native applications across multiple Kubernetes clusters and clouds

Join us

https://karmada.io

https://github.com/karmada-io/karmada

https://slack.cncf.io (#karmada)

Q&A

Thank you!