
KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023

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

Wei Jiang, Huawei Cloud & XingYan Jiang, DaoCloud

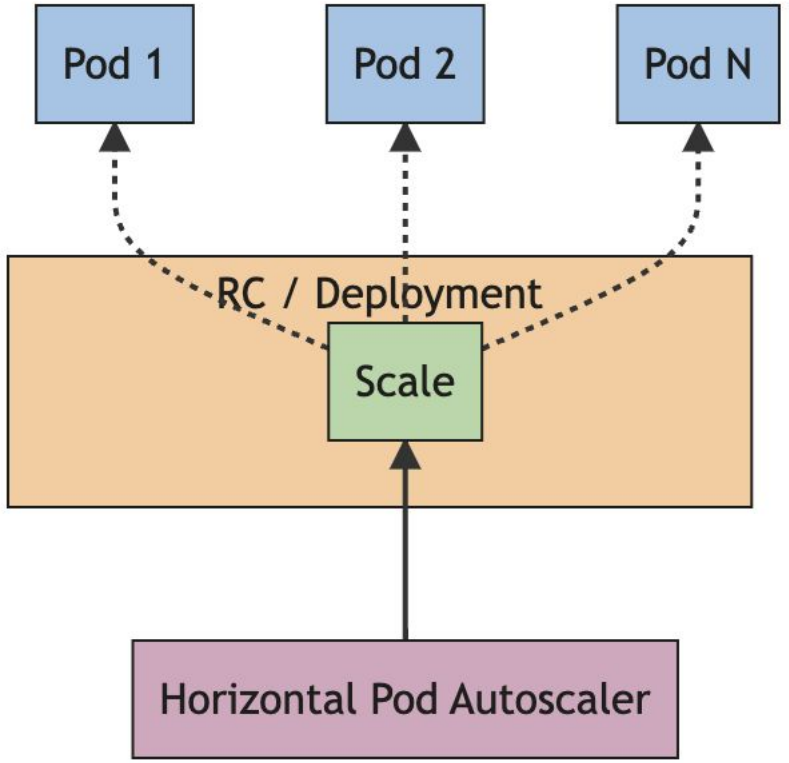
Agenda

- HPA overview and limitations
- Benefits of autoscaling across clusters
- Introducing Karmada
- Introducing FederatedHPA
- Explore distributed autoscaling
- Applicable scenarios
- Q&A

HPA overview and limitations

What is HPA?

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.



During each time period, the HPA controller manager queries resource utilization based on metrics specified in each HorizontalPodAutoscaler definition. The controller manager identifies the target resource defined by **scaleTargetRef**, selects Pods based on **.spec.selector** labels from the target resource, and retrieves metrics from either the Resource Metrics API (for per-Pod resource metrics) or Custom Metrics API (for all other metrics).

HPA overview and limitations

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.

HPA overview and limitations

HPA limitations:

- As a user, I use HPA to scale my application services within a single cluster. Due to limitations of the single cluster such as **resource constraints**, excessive requests may cause service unavailability. Therefore, I hope to improve the **scalability and stability of the service by autoscaling cross-cluster** methods.

- As a user, I deploy my applications in multiple clusters. As the workload of these applications may vary with request concurrency, I hope to use **rich cross-cluster scaling strategies** to achieve different goals.

- As a user, I use HPA to scale my applications across multiple clusters. Unlimited expansion of these applications may result in unexpected cloud costs and resource congestion that could lead to interruptions in other application services. Therefore, I hope to **set maximum instance numbers for multi-cluster applications**.

- As a user, for multi-cluster services, an HPA is configured for each cluster for autoscaling of services. However, managing them separately can be complex and redundant; therefore, I hope to have **unified configuration and management of autoscaling configuration**.

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.

Introducing 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

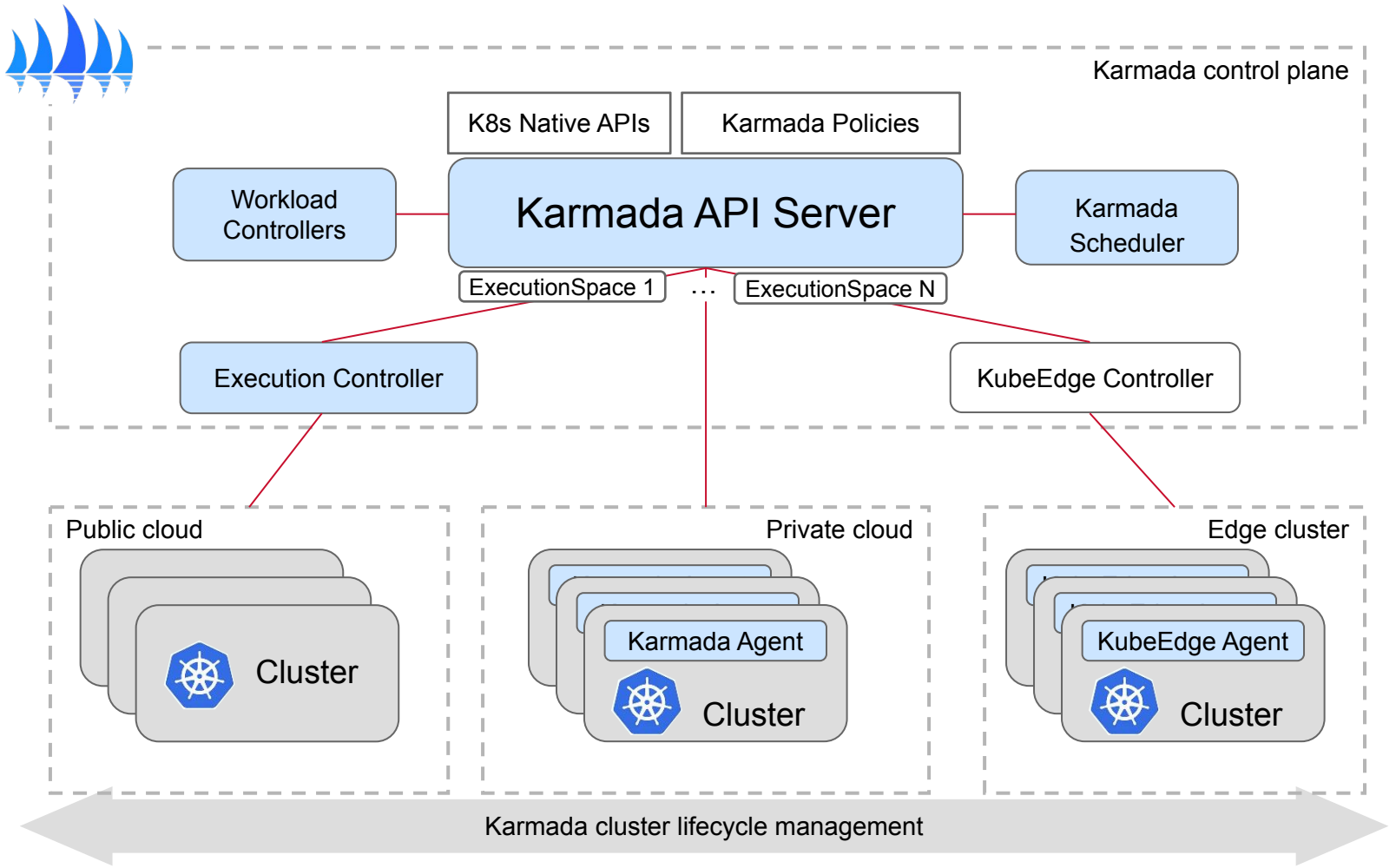
Avoid Vendor Lock-in

Out of the Box

Fruitful Scheduling Policies

Centralized Management

Introducing Karmada



Introducing 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"`
}
```

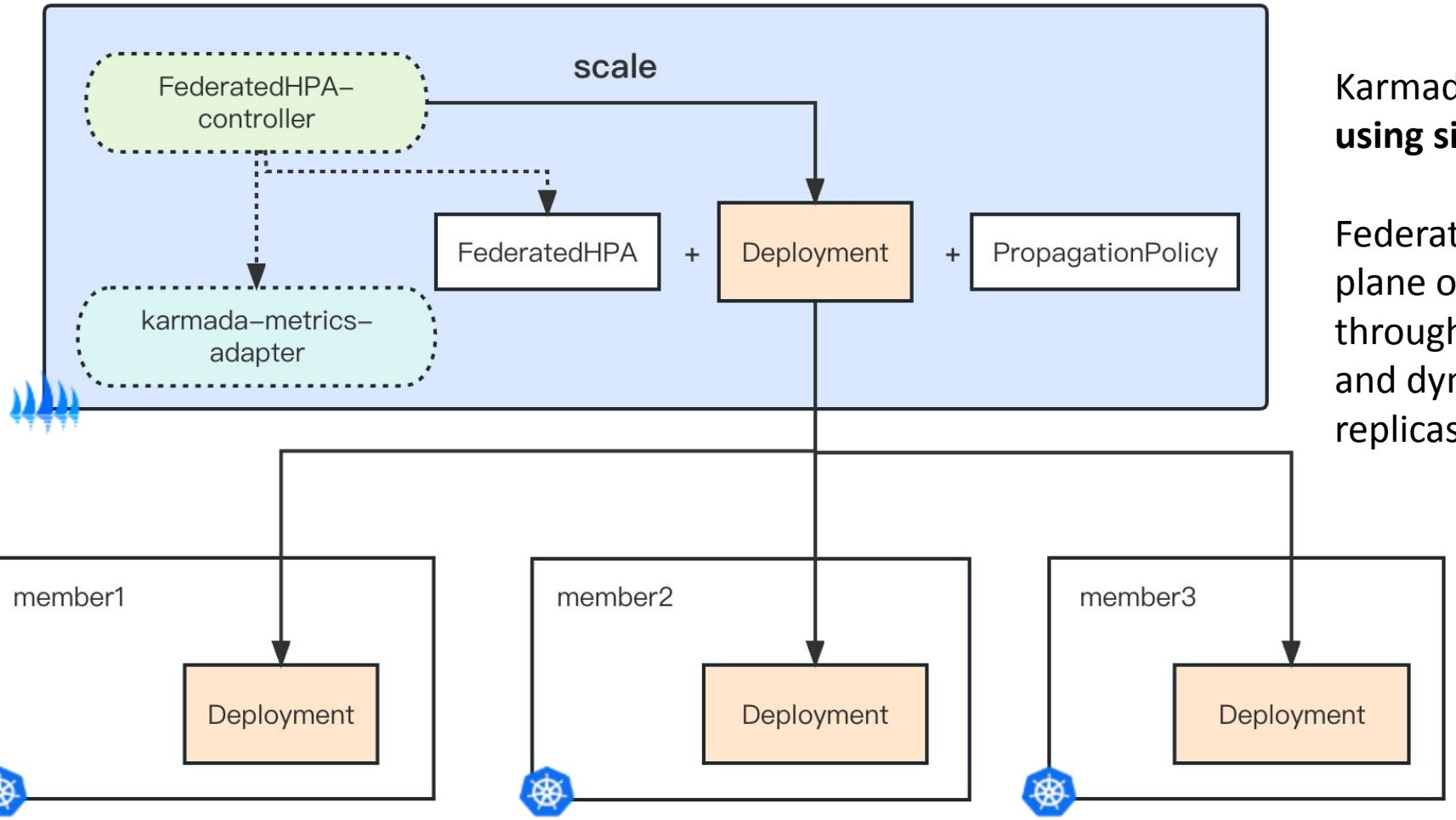
Keep **consistent** with the core API of K8s native HPA. It can be **migrated** to FederatedHPA at a very low cost.

Introducing 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
```

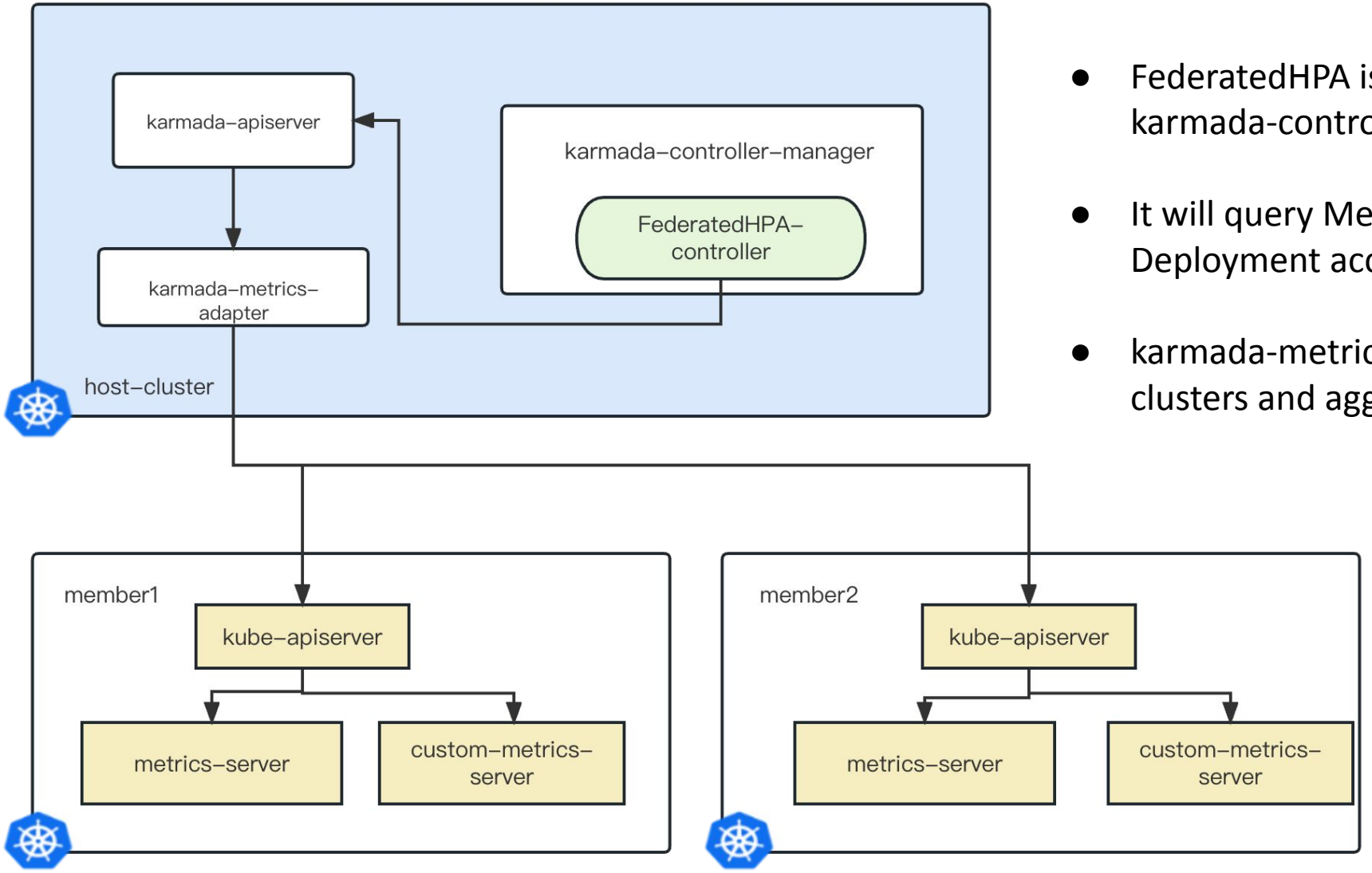
Introducing FederatedHPA



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 of Deployment.

Introducing FederatedHPA



- FederatedHPA is located in the karmada-controller-manager component.
- It will query Metrics from karmada-apiserver and scale Deployment accordingly.
- karmada-metrics-adapter queries metrics from member clusters and aggregates them.

Introducing FederatedHPA

Extension - CronFederatedHPA:

Used for periodic scaling operations. It can scale workloads with scale sub-resources or FederatedHPA.

Scale FederatedHPA

```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 Deployment,etc

```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
```

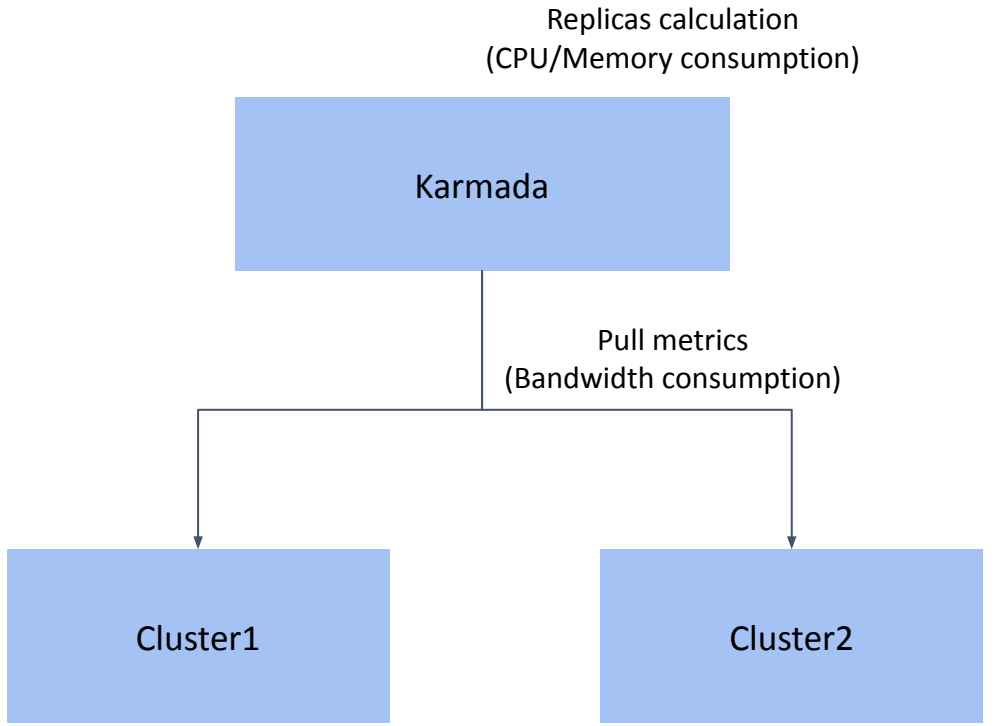
Explore distributed autoscaling

Advantages of FederatedHPA:

- The API and K8s HPA v2 versions are almost identical, with low migration costs.

Notes for using FederatedHPA:

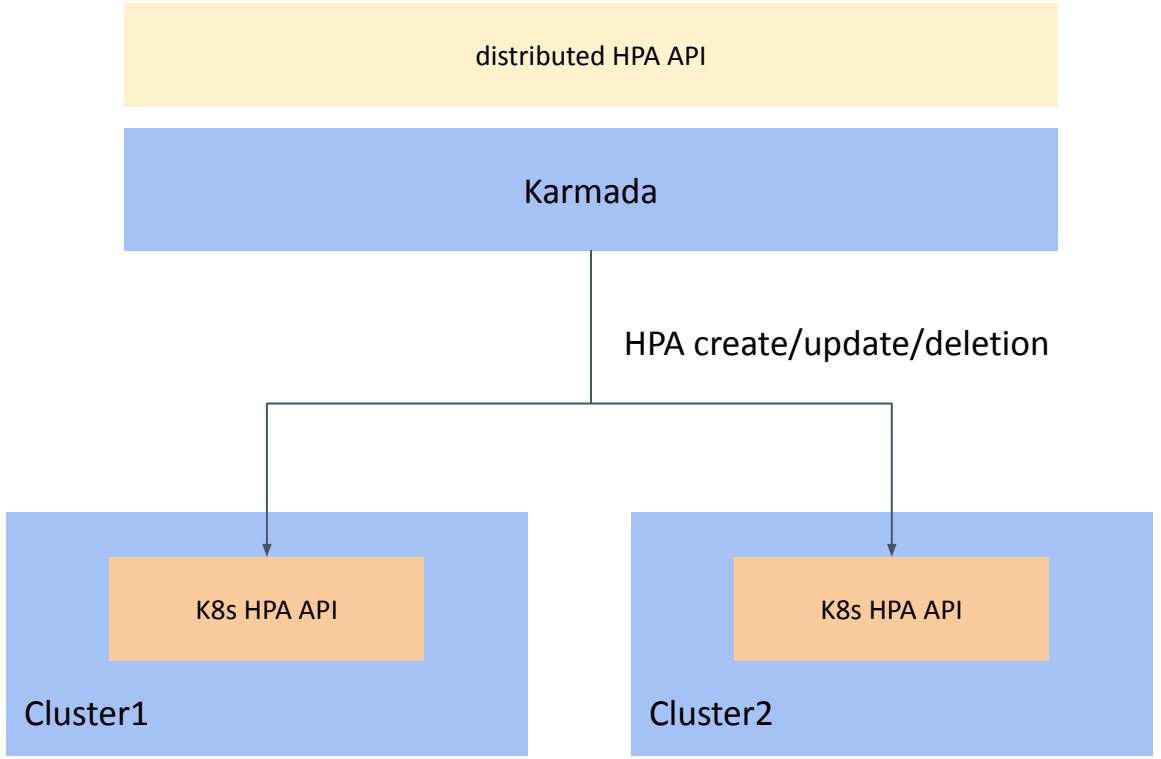
- FederatedHPA is a type of centralized multi-cluster HPA.
- When scaling concurrently on a large scale, a larger bandwidth is required (maximum concurrent scaling metrics pull: 500M bandwidth, 200,000 Pods).
- Storing and computing corresponding data requires much CPU/Memory.



Explore distributed autoscaling

Design 1:

- Through the distributed HPA API, define strategies for managing HPA in member clusters.
- The HPA controller of the member cluster scales the workload based on the propagated HPA configuration.

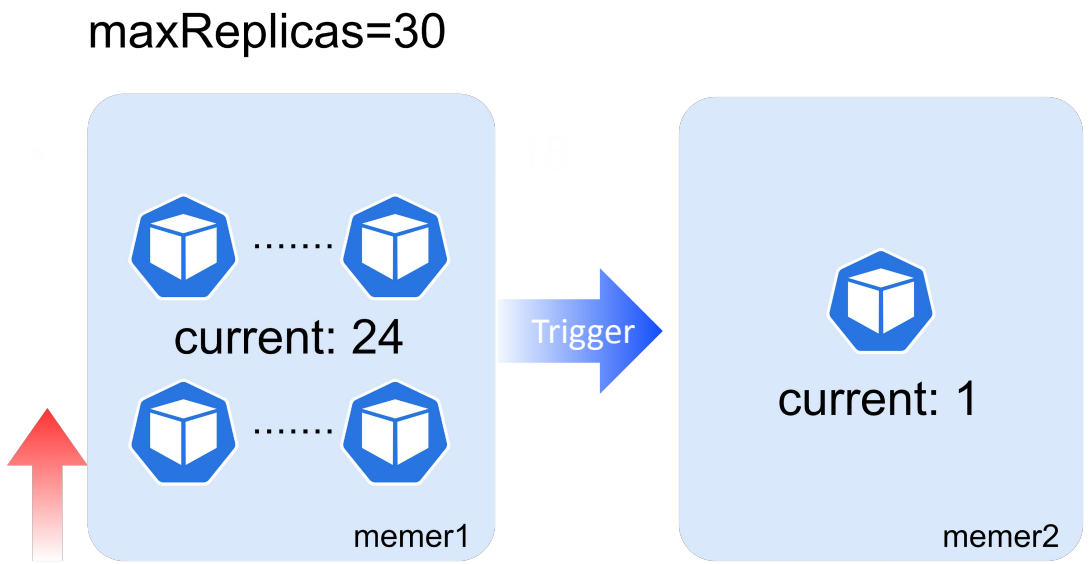


Shortcomings:

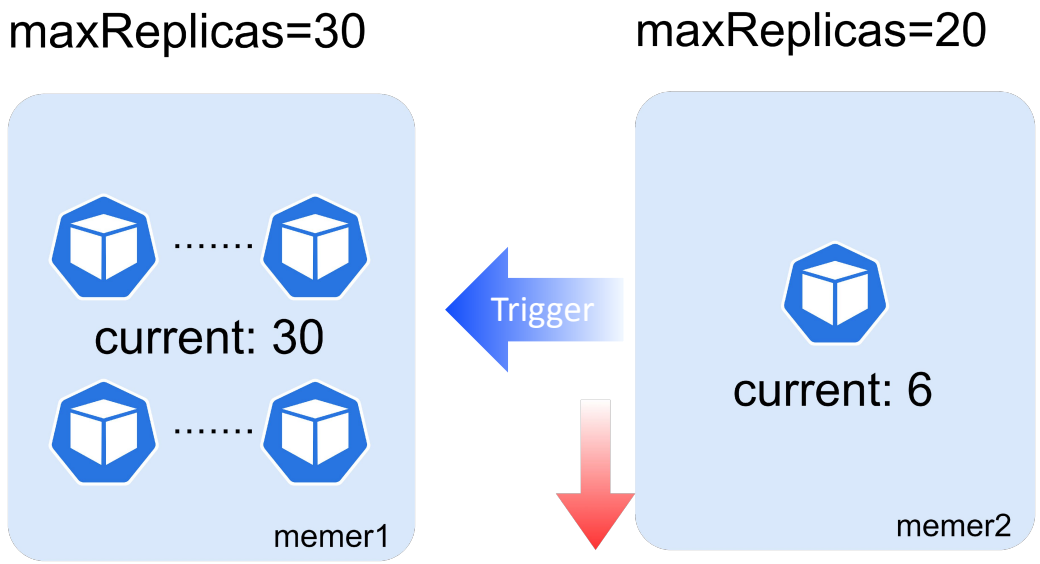
- Karmada needs to control over scaling behaviors in different clusters, so, Karmada need to modify (min/max) or dynamically create/delete member cluster HPAs, which is a complex logic.
- Encompasses the logic for determining the placement of scaled replicas (implemented by HPA management) and shares overlapping functionality with the scheduling design.

Explore distributed autoscaling

Scenario examples (priority):



- Set the **minReplicas/maxReplicas** for scaling each cluster, and set the threshold (such as 24 or 80%) for cross-cluster scaling trigger.
- When the threshold for cross-cluster expansion is reached during cluster expansion, enable HPA scaling for the next cluster.



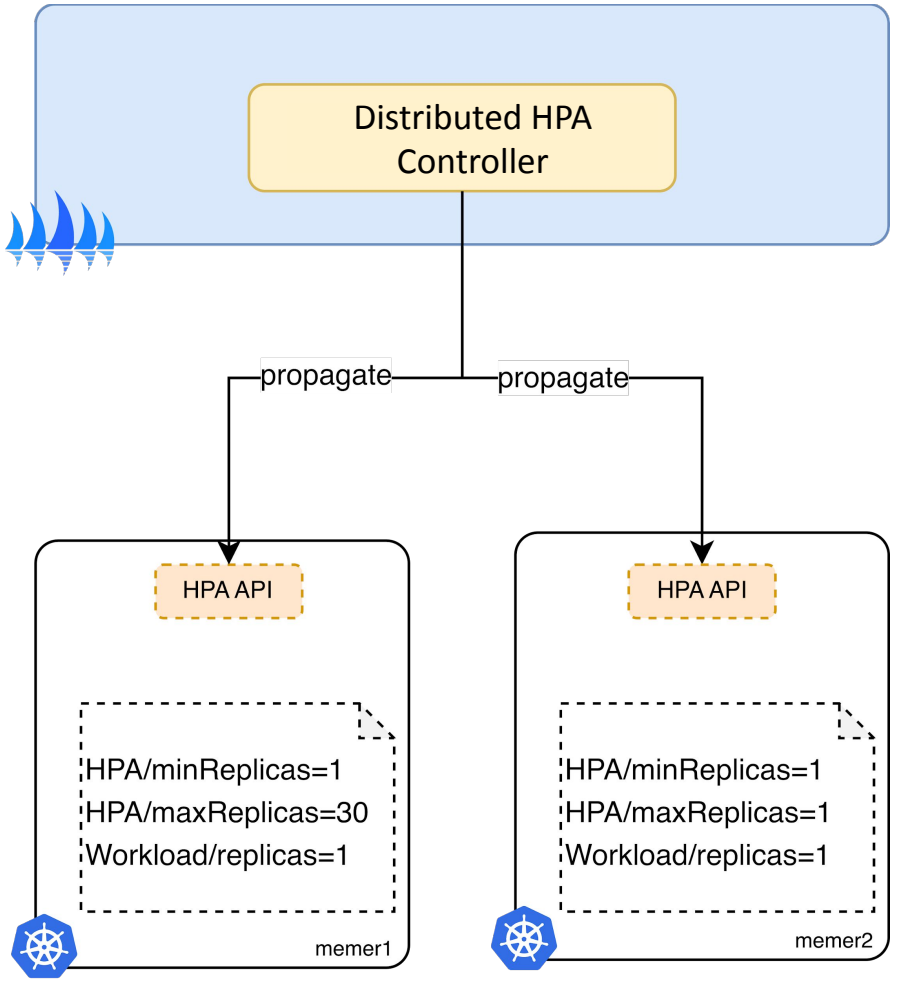
- Set the **minReplicas/maxReplicas** for scaling of each cluster, and the threshold for cross-cluster scaling (such as 6 or 30%) trigger.
- When cross-cluster scaling threshold is reached during cluster scaling, HPA scaling of the before cluster will be enabled.

Explore distributed autoscaling

- Initial :

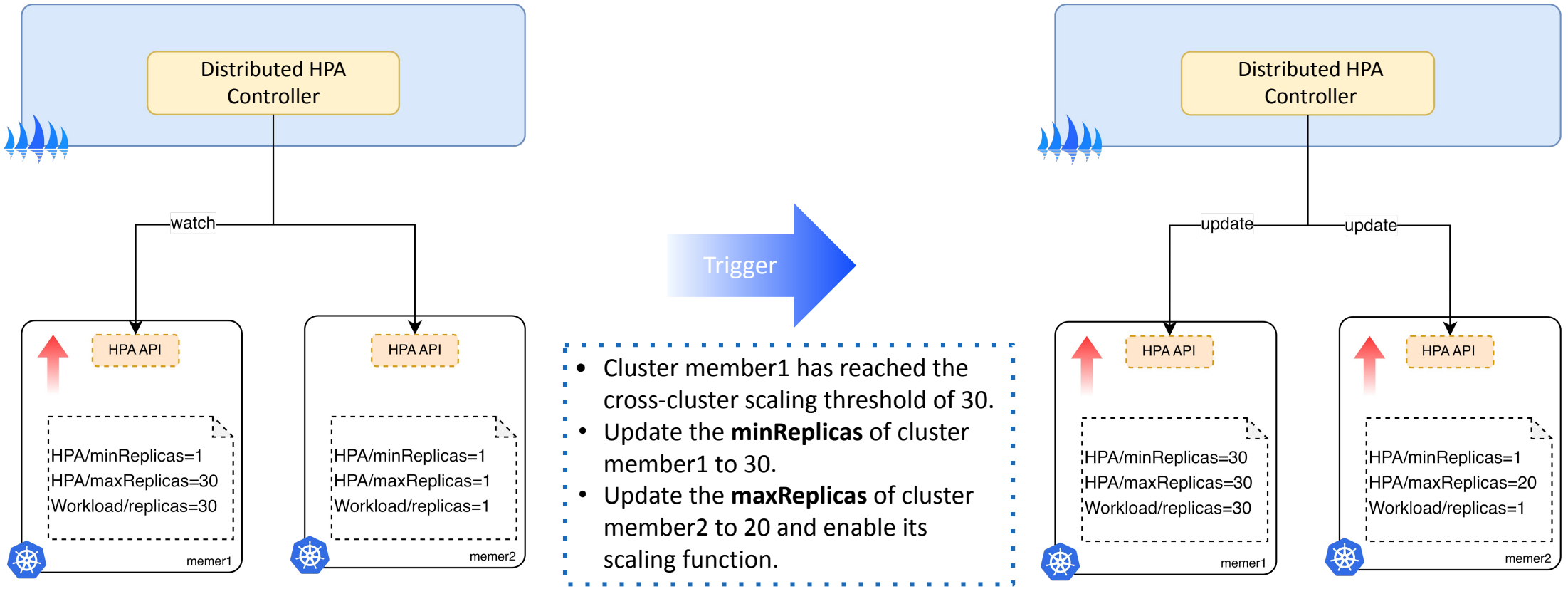
According to the Distributed HPA **min/max** and strategy configuration.

Initial



Explore distributed autoscaling

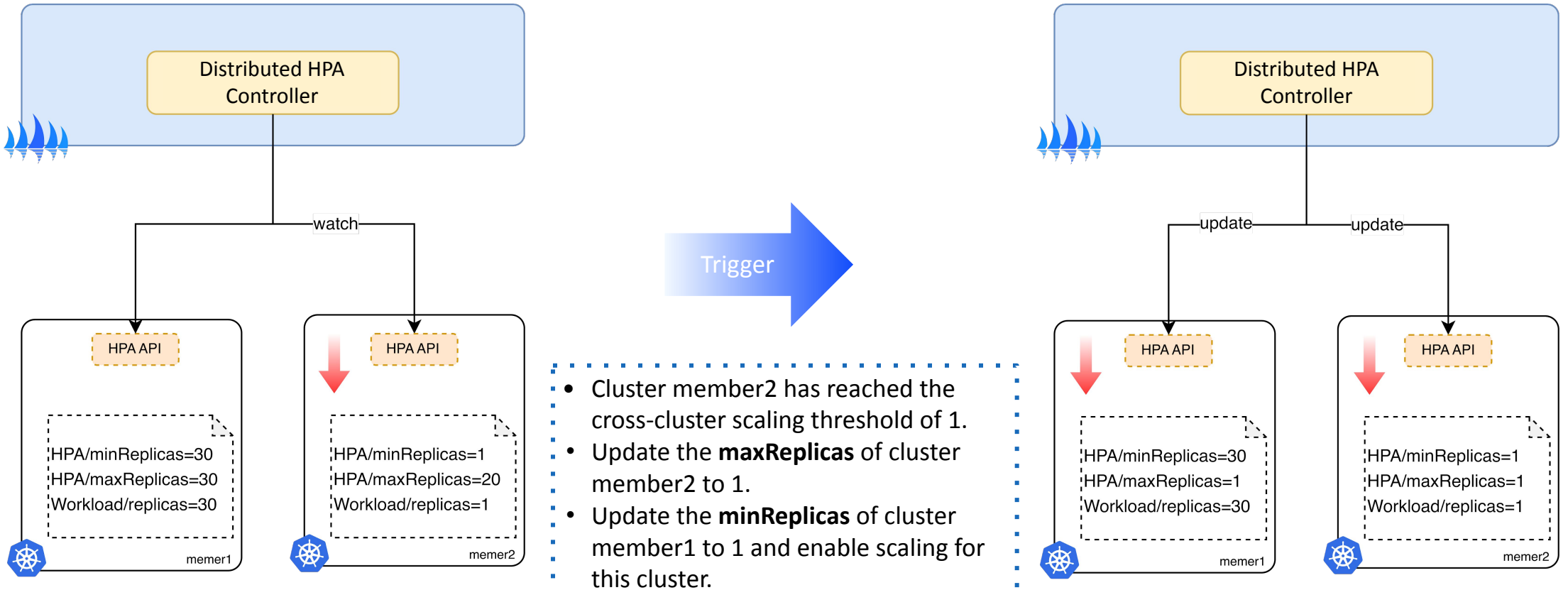
- Scale-Up Steps:



- Cluster member1 has reached the cross-cluster scaling threshold of 30.
- Update the **minReplicas** of cluster member1 to 30.
- Update the **maxReplicas** of cluster member2 to 20 and enable its scaling function.

Explore distributed autoscaling

- Scale-Down Steps:



- Cluster member2 has reached the cross-cluster scaling threshold of 1.
- Update the **maxReplicas** of cluster member2 to 1.
- Update the **minReplicas** of cluster member1 to 1 and enable scaling for this cluster.

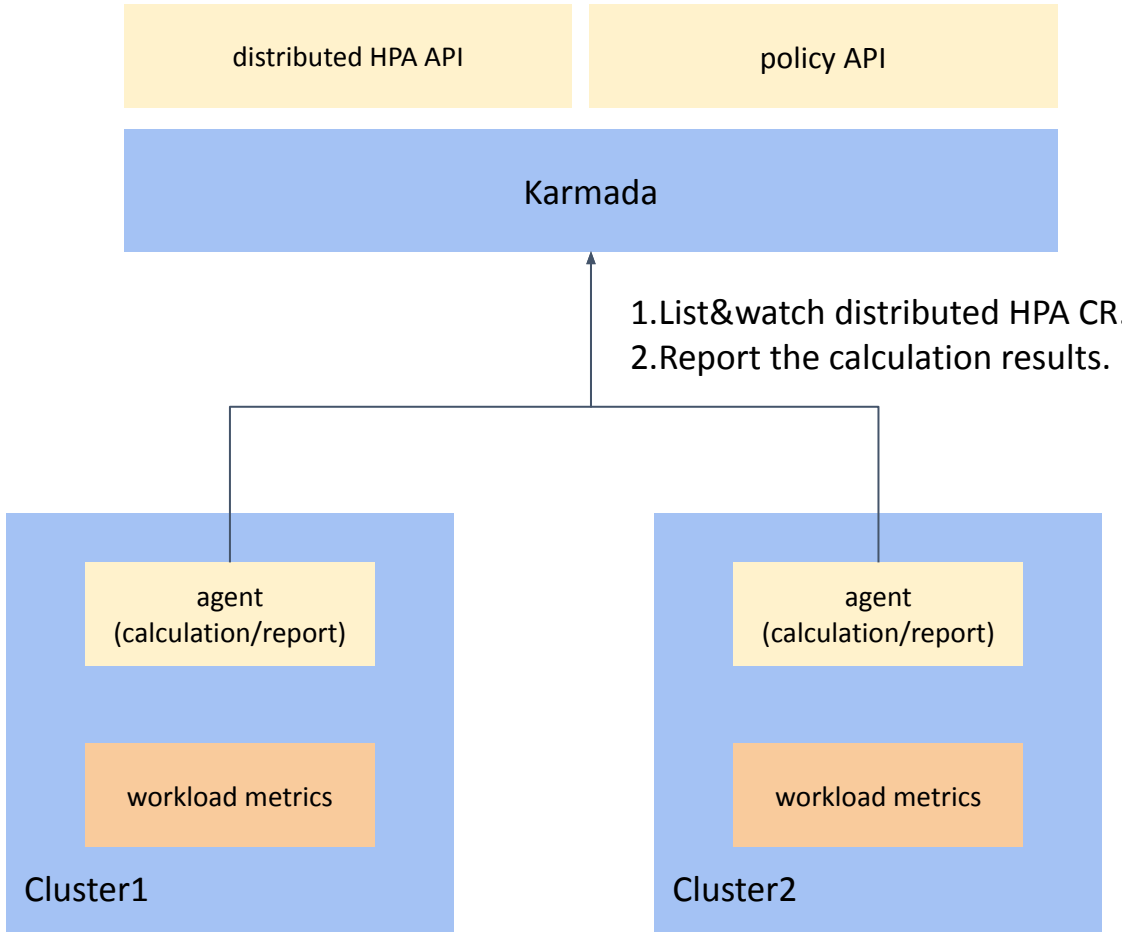
Explore distributed autoscaling

Design 2:

- Define the metrics for scaling in distributed HPA API.
- Member cluster agents will list&watch the distributed HPA CR and calculate the intermediate calculation state.
- Update the calculated intermediate state to the "status" of distributed HPA CR.
- Karmada processes abnormal data based on reports from multiple 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.

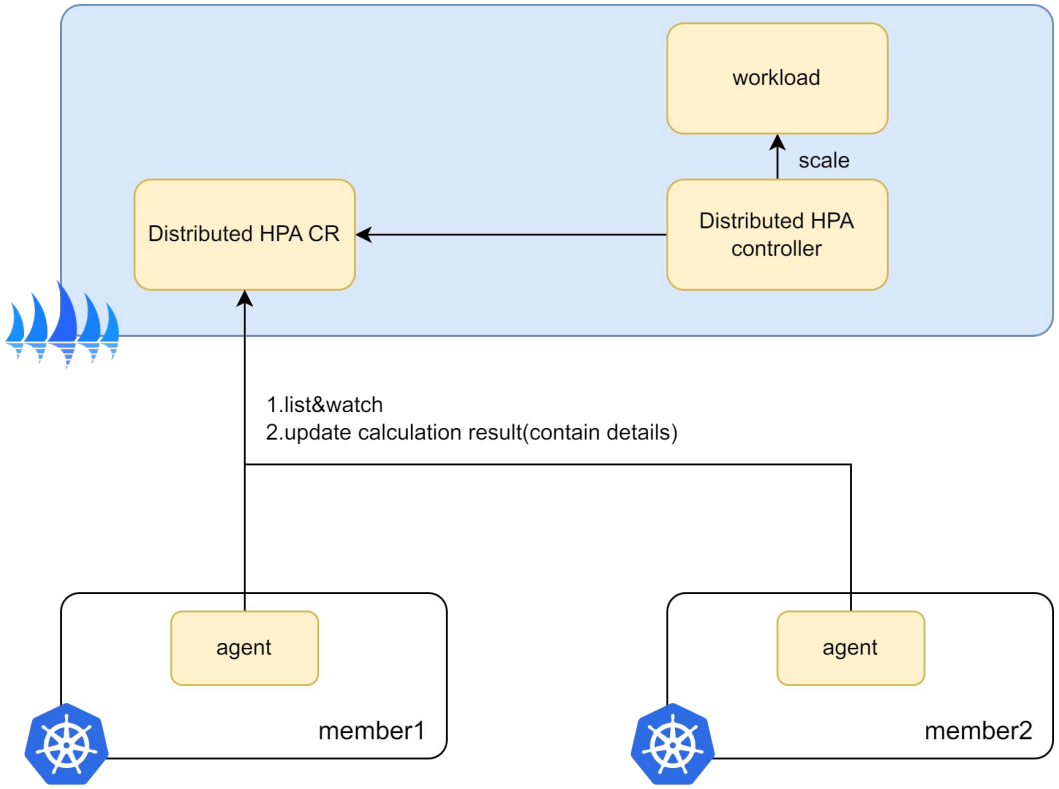
Shortcomings:

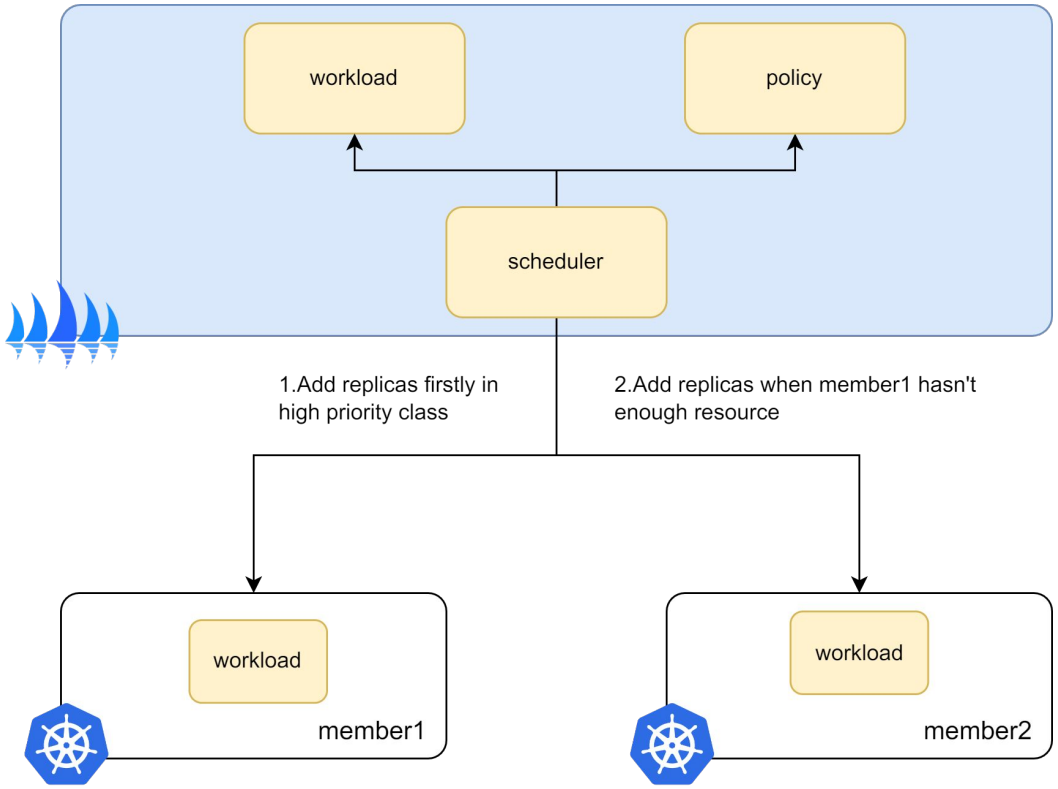
- The agent component has a high maintenance cost and requires upgrades and version compatibility.



Explore distributed autoscaling

Scenario examples (priority,scale-up):





Explore distributed autoscaling

Design 1

- Smaller maintenance costs (without agent components)
- Rougher cross-cluster autoscaling (member clusters first, without a god's-eye view)
- Mixed duties for scaling and scaling replicas placement

Design 2

- Higher maintenance costs (due to the existence of agent components)
- More precise cross-cluster autoscaling (member clusters report data, Karmada overall cross-cluster elasticity)
- Separation of duties for scaling and scaling replicas placement

Explore distributed autoscaling

FederatedHPA(Centralized)

- Concurrent autoscaling needs to match the bandwidth size.
- Scaling capacity should match component resource allocation.
- Lower maintenance costs.

Distributed HPA

- Large scale, but limited bandwidth or control plane component resources.
- Lower bandwidth consumption means more cost savings.

Join us

https://karmada.io

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

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

Q&A

Thanks