
KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023



KubeCon | CloudNativeCon


OPEN SOURCE SUMMIT

China 2023

Non-intrusively Enable OpenKruise and Argo Workflow in a Multi-Cluster Federation

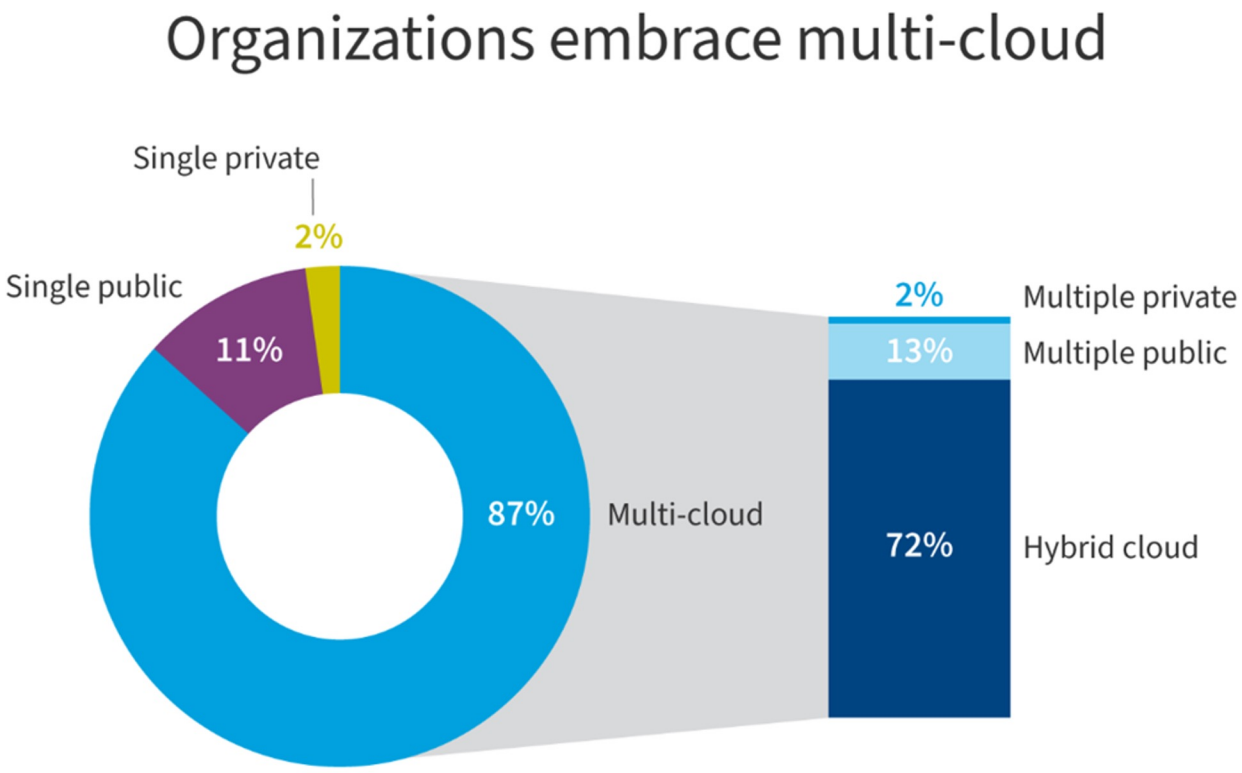
Tiecheng Shen, Volcano Engine
Rong Zhang, vivo

Agenda

- Multi-cluster Federation Background
- Challenges of Migration and Integration(take OpenKruise and Argo Workflow as examples)
- Resource Interpreter Framework
- Adoption in vivo

Multi-cluster Federation Background

Why Multi-cloud



N=750
Source: Flexera 2023 State of the Cloud Report

Typical stages

Different stages of multi-cloud multi-cluster

Stage 1: Isolated Clusters

- Consistent cluster operation and maintenance
- Consistent application delivery
- Resource islands

Stage 2: Cluster Fleet

- Unified application delivery
- Unified application access
- Unified scheduling

Stage 3: Unified resource pool

- Infinitely scalable cluster pools

Challenges

Challenges of managing multiple Kubernetes clusters

Problem 1: Too many clusters

- Cumbersome and Repetitive setup
- Incompatible Cluster Lifecycle API
- Fragmented API endpoints

Problem 2: Workload Fragmentation

- Per-cluster customization for Apps
- Multi-cluster service discovery for Apps
- Sync Apps between clusters

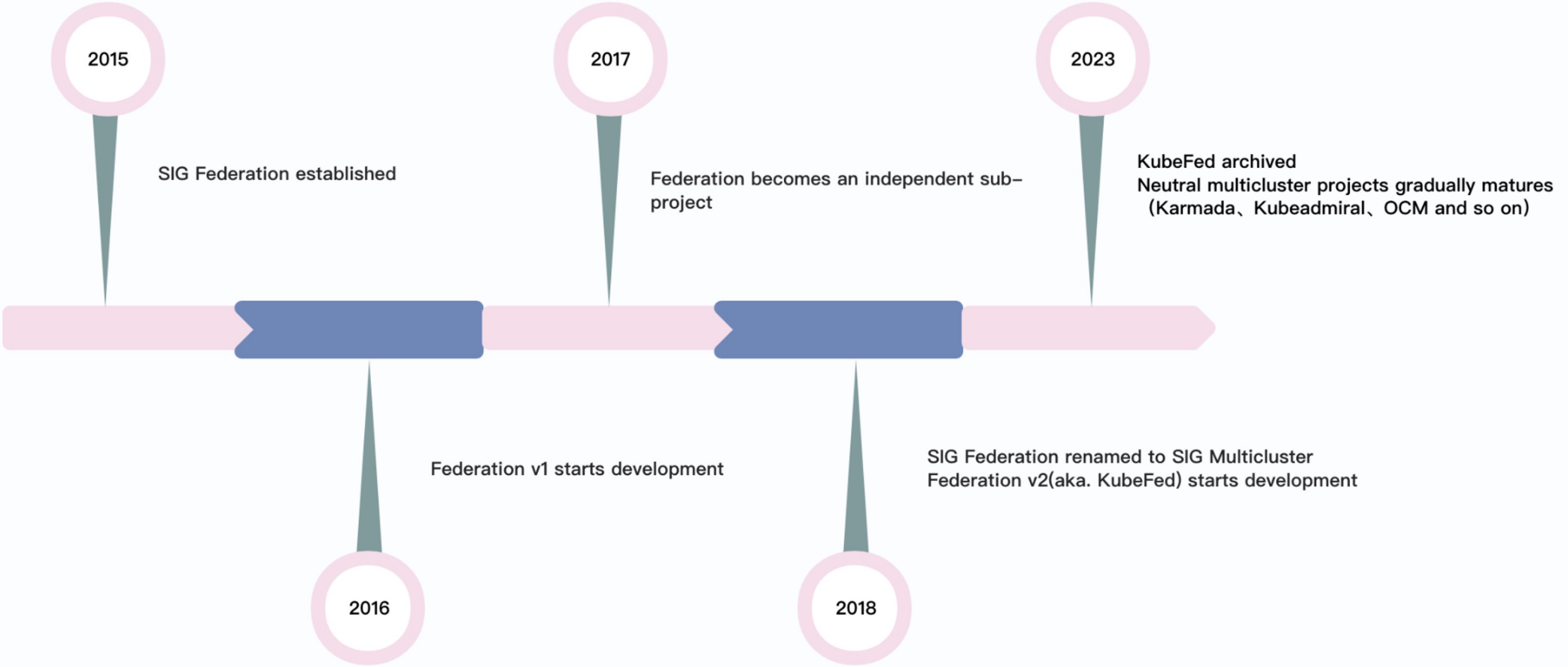
Problem 3: Boundary of Clusters

- Resource Scheduling
- Application Availability
- Horizontal Auto-scaling

Problem 4: Vendor lock-in

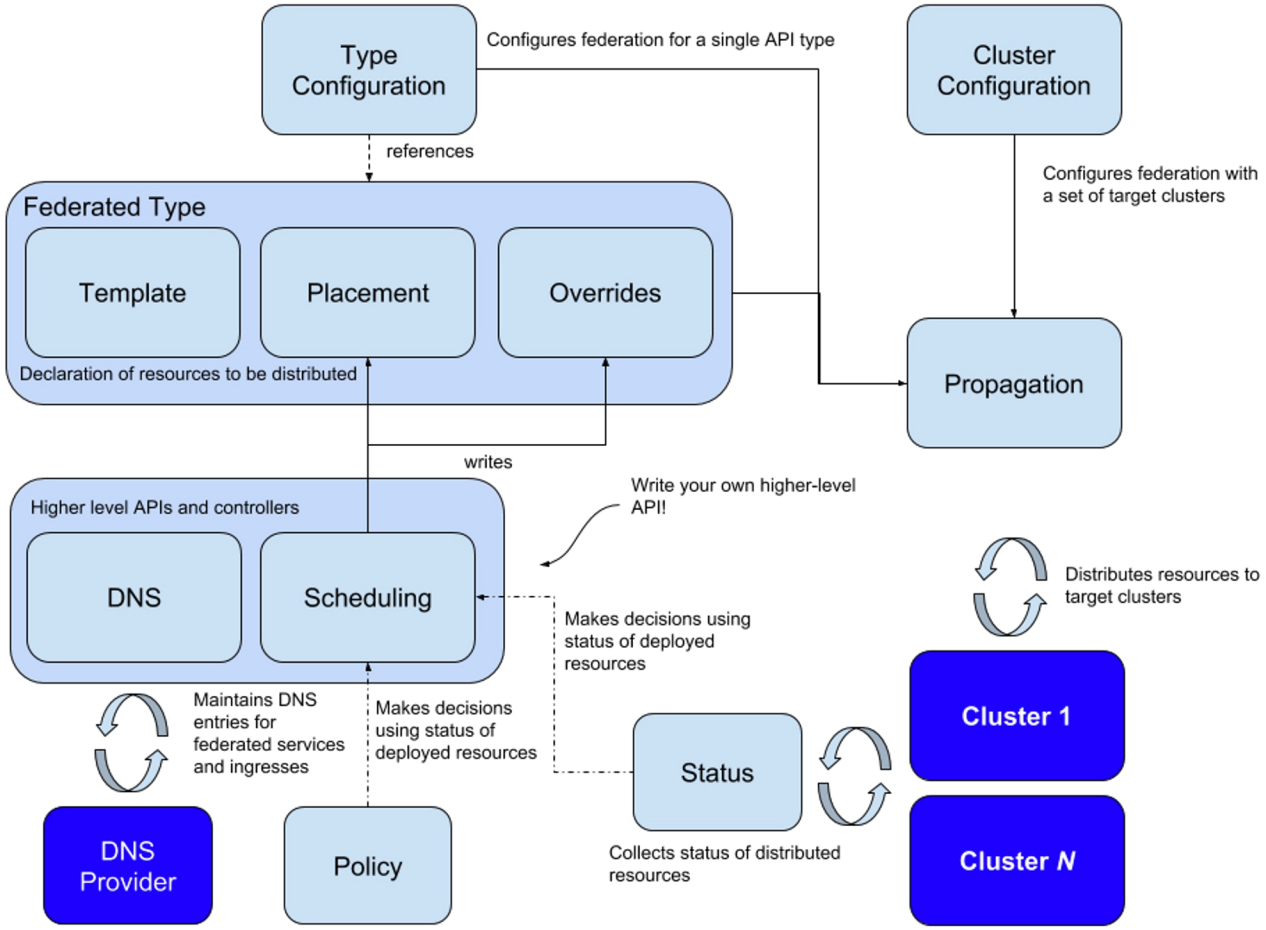
- Deployment Gravity
- Lack of Migration Automation
- Lack of independent projects in multi-cloud ecosystem

Community Timeline



Challenges of Migration and Integration

KubeFed's Dilemma



Advantage

- Modular design
- Placement and override API structure

Disadvantage

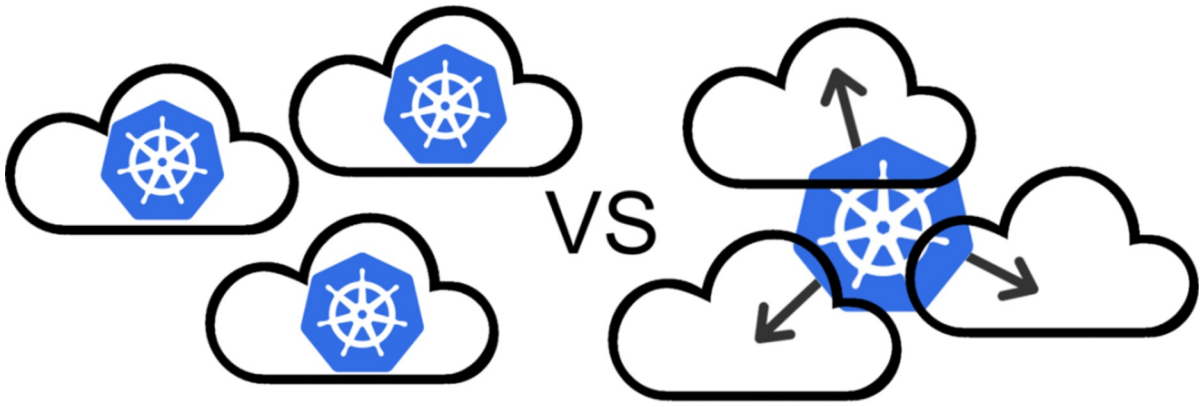
- Not compatible with K8s native APIs, especially the lack of expansion capabilities for CRDs, require extra learning and adoption efforts
- Lack of turnkey solution, customizations on building blocks result in no standard

Migration and Integration

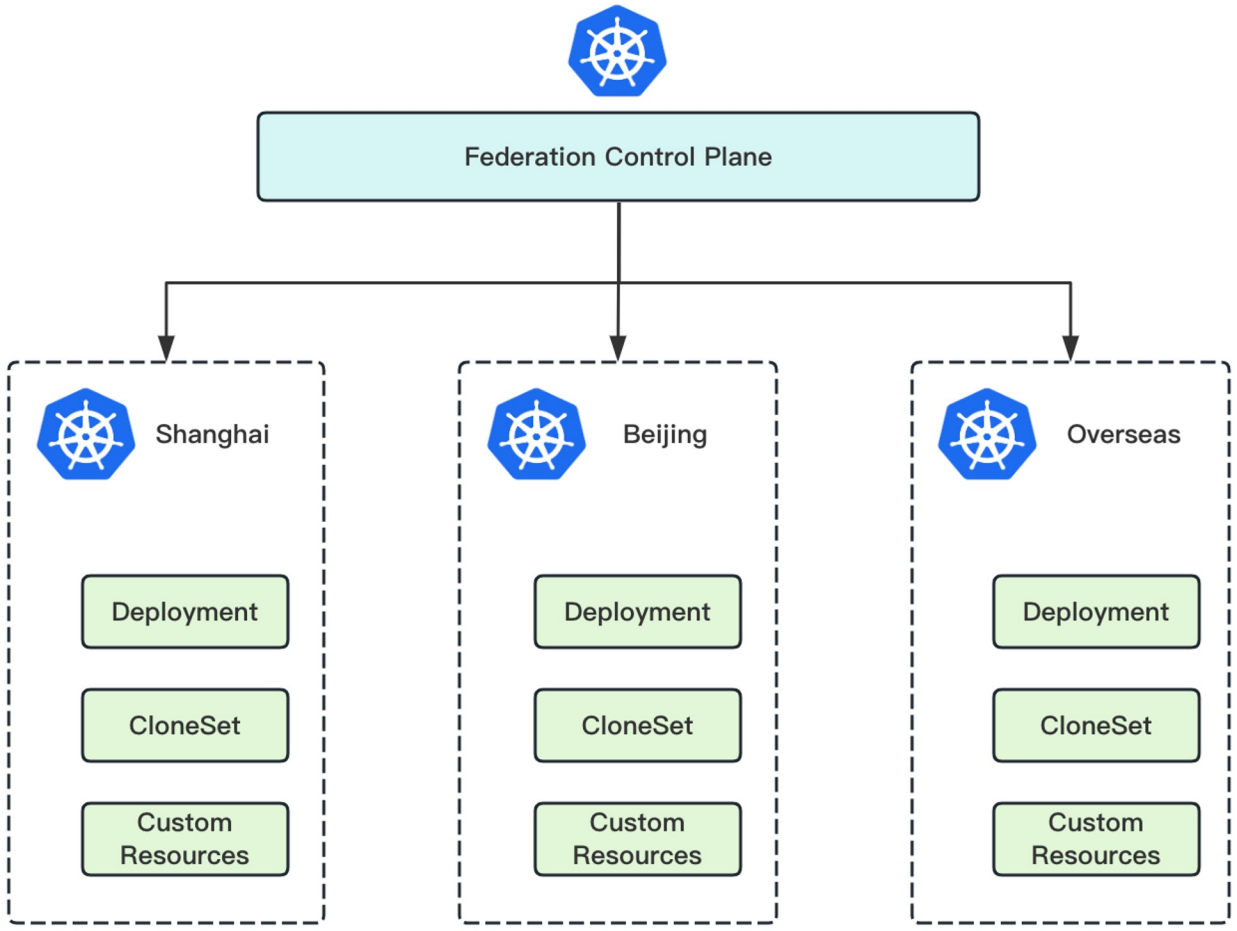
When users switch from a single-cluster architecture to a multi-cluster federation, the core demands of users are often

- **to reduce intrusive modifications as much as possible(migration)**
- **be as compatible with the wider Kubernetes ecosystem as possible(integration)**

However, KubeFed does not solve the above problem as well.



Typical Cases



Situation 1:

Assume we have several clusters and several kinds of workloads:

- Deployment
- CloneSet from OpenKruise
- Custom Resources controlled by operators

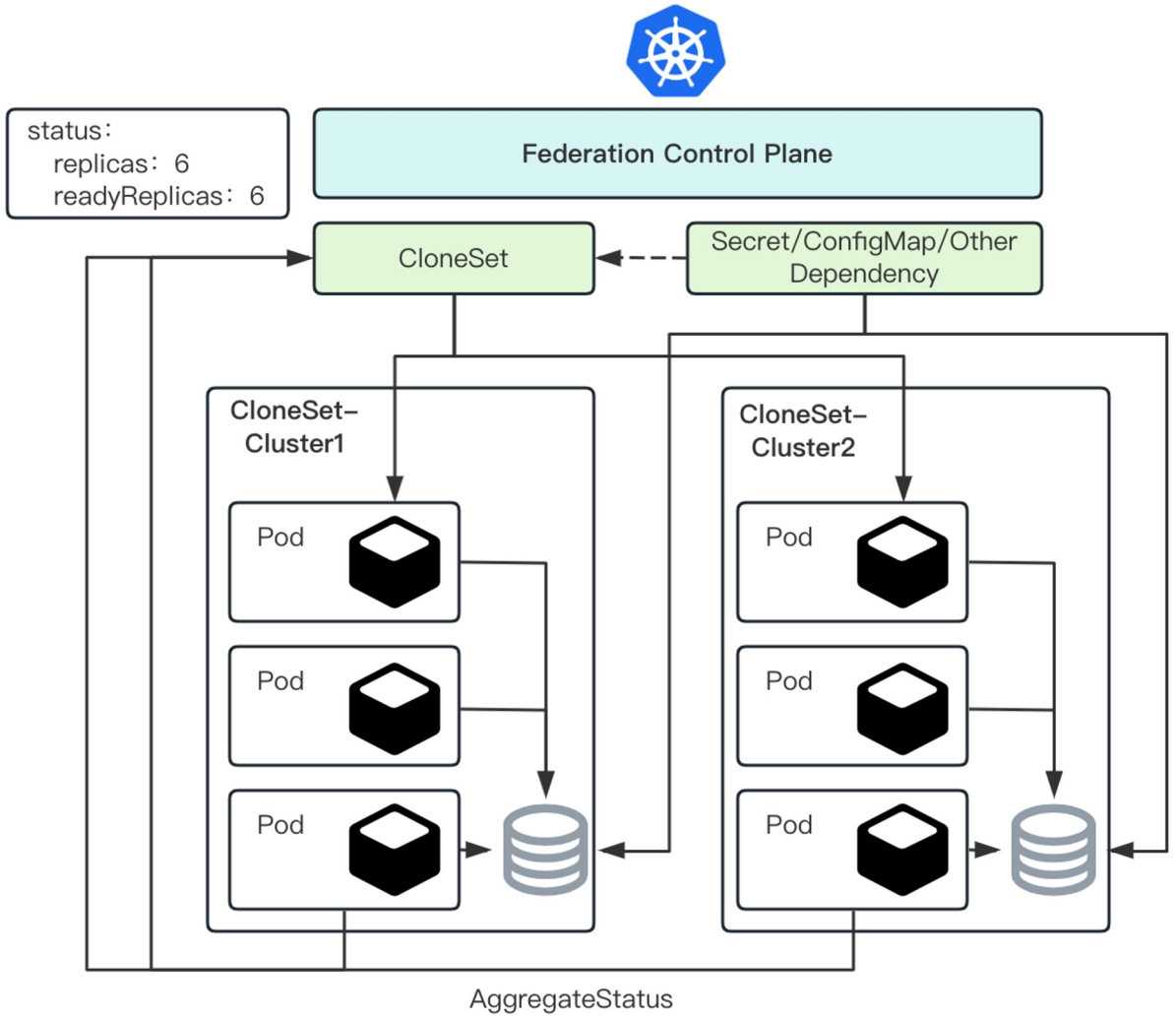
We hope to:

- Use a unified strategy to dynamically allocate replicas across multiple clusters

Which means we need to:

- Resolve the replicas number of different workloads and their required resources

Typical Cases



Situation 2:

Assume we have registered several clusters into a federation and start to propagate workloads:

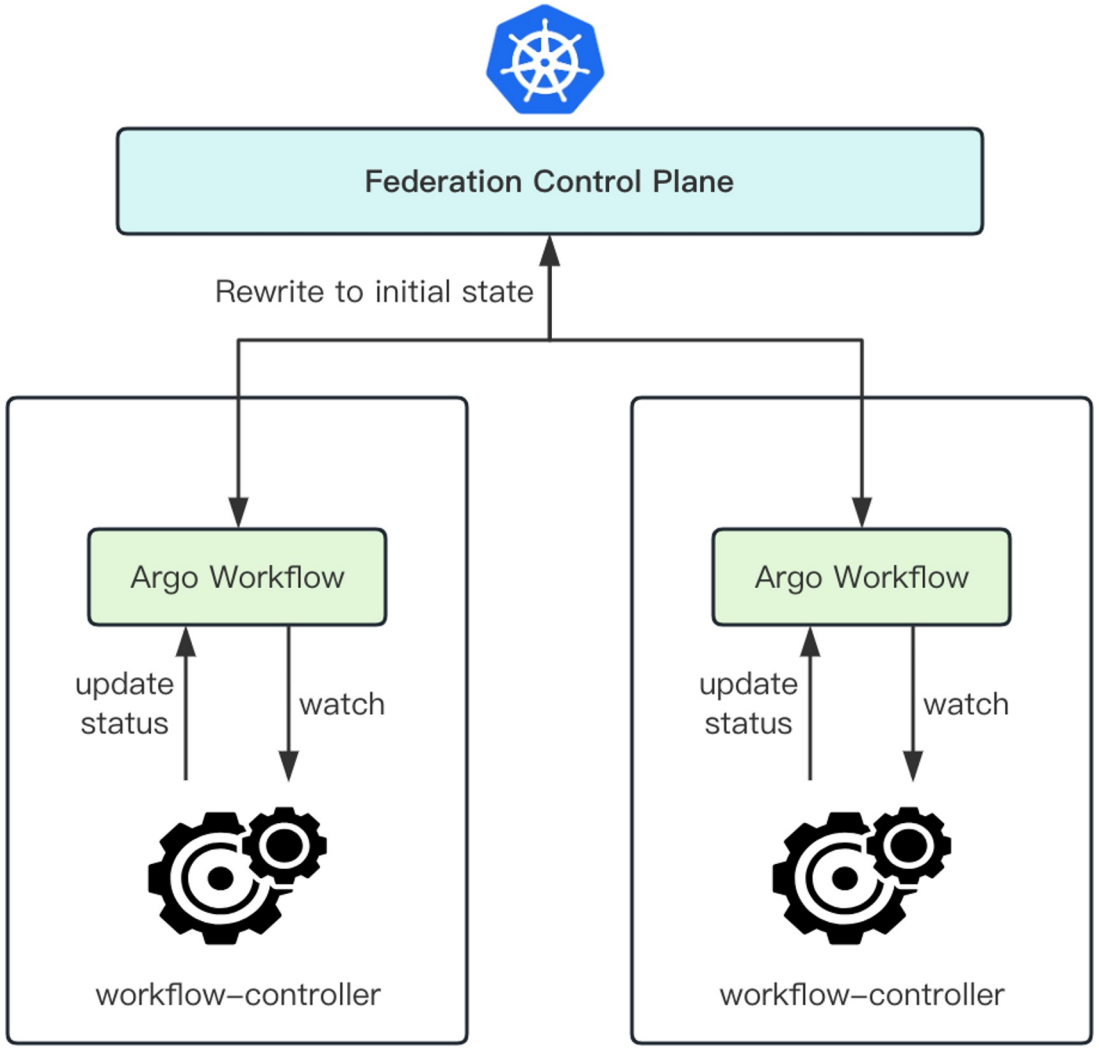
We hope to:

- Automatically identify dependent objects of different CRDs
- Aggregate the status of resource objects on different clusters

Which means we need to:

- Resolve the dependency of different workloads
- Resolve the status of different workloads and define how they are aggregated

Typical Cases



Situation 3:

Assume we propagate several workflows to different clusters using a multi-cluster federation.

Since the federated controller and the operators in member clusters will update resource objects at the same time, resource objects will change repeatedly and deadlock will occur.

Typical cases:

- workflow-controller updates the status
- hpa-controller updates the replicas

Which means we need to:

- Decide which value to use when a conflict occurs

Resource Interpreter Framework

Interpreter Operation

To solve the above problems, Karmada (a Multi-Cluster Kubernetes Orchestration) implements a common **Resource Interpreter Framework** to interpret resource structure. Users can customize resource interpreters based on their CRs.

Karmada defines several **interpreter operations**:

- **InterpretReplica**: indicates how to figure out the replica declaration of a specific object.
- **ReviseReplica**: indicates how to modify the replica by adopting the replica scheduling policy.
- **InterpretStatus**: indicates how to figure out how to get the status.
- **Retain**: indicates how to retain the desired resource template to resolve conflicts between federation and member clusters.
- **AggregateStatus**: indicates how to figure out how to aggregate status to resource template.
- **InterpretHealth**: indicates how to figure out the health status of a specific object.
- **InterpretDependency**: indicates how to figure out the dependencies of a specific object.

Custom Interpreters

Resource Interpreter Framework consists of built-in and customized interpreters:

- built-in interpreter: used for common Kubernetes native or well-known extended resources.
- customized interpreter: interprets custom resources or overrides the built-in interpreters.

    - Running a webhook at runtime.
    - Defining declarative configuration. Users can quickly customize resource interpreters for both Kubernetes resources and CR resources by the rules declared in the *ResourceInterpreterCustomization* API specification.

ResourceInterpreterCustomization is a CRD which defines the implementation of interpreter operations by Lua scripts.

Example Code For CloneSet

InterpretReplicas:

```
replicaResource:
  luaScript: >
    local kube = require("kube")
    function GetReplicas(obj)
      replica = obj.spec.replicas
      requirement = kube.accuratePodRequirements(obj.spec.template)
      return replica, requirement
    end
replicaRevision:
  luaScript: >
    function ReviseReplica(obj, desiredReplica)
      obj.spec.replicas = desiredReplica
      return obj
    end
```

AggregateStatus:

```
statusAggregation:
  luaScript: >
    function AggregateStatus(desiredObj, statusItems)
      replicas = 0
      updatedReplicas = 0
      readyReplicas = 0
      availableReplicas = 0
      updatedReadyReplicas = 0
      expectedUpdatedReplicas = 0
      for i = 1, #statusItems do
        if statusItems[i].status ~= nil and statusItems[i].status.replicas ~= nil then
          replicas = replicas + statusItems[i].status.replicas
        end
        if statusItems[i].status ~= nil and statusItems[i].status.updatedReplicas ~= nil then
          updatedReplicas = updatedReplicas + statusItems[i].status.updatedReplicas
        end
        if statusItems[i].status ~= nil and statusItems[i].status.readyReplicas ~= nil then
          readyReplicas = readyReplicas + statusItems[i].status.readyReplicas
        end
        if statusItems[i].status ~= nil and statusItems[i].status.availableReplicas ~= nil then
          availableReplicas = availableReplicas + statusItems[i].status.availableReplicas
        end
        if statusItems[i].status ~= nil and statusItems[i].status.updatedReadyReplicas ~= nil then
          updatedReadyReplicas = updatedReadyReplicas + statusItems[i].status.updatedReadyReplicas
        end
        if statusItems[i].status ~= nil and statusItems[i].status.expectedUpdatedReplicas ~= nil then
          expectedUpdatedReplicas = expectedUpdatedReplicas + statusItems[i].status.expectedUpdatedReplicas
        end
```

Example Code For Workflow

Retain:

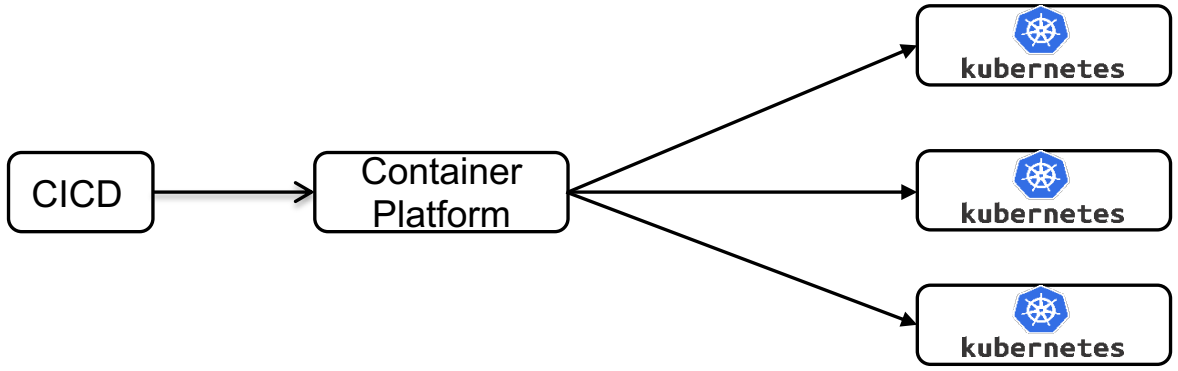
```
retention:
  luaScript: >
    function Retain(desiredObj, observedObj)
      if observedObj.spec.suspend ~= nil then
        desiredObj.spec.suspend = observedObj.spec.suspend
      end
      if observedObj.status ~= nil then
        desiredObj.status = observedObj.status
      end
      return desiredObj
    end
```

InterpretHealth:

```
healthInterpretation:
  luaScript: >
    function InterpretHealth(observedObj)
      if observedObj.status == nil then
        return false
      end
      if observedObj.status.phase == nil or observedObj.status.phase == '' or
          observedObj.status.phase == 'Failed' or observedObj.status.failed == 'Error' then
        return false
      end
      return true
    end
```

Adoption in vivo

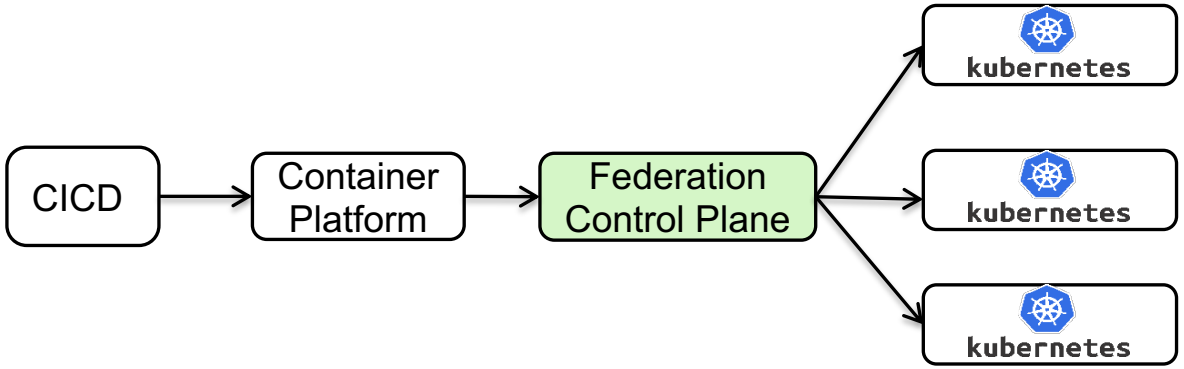
Current status of Container Platform



Multi-Cluster in vivo

Multi-Cluster in vivo

- Platform provides cluster admin permission import
- Each cluster runs independently
- Integrate functions such as continuous integration, continuous delivery, and monitoring and alerting
- The platform provides the ability to view resource usage across multiple clusters
- Aggregation of APIs across multiple clusters



Multi-Cluster Federation

Multi-Cluster Federation in vivo

- Federation of Kubernetes resources and other CRD
- Unified scheduling
- Unified application delivery
- Unified application access
- FedHpa and CronHpa
- Automated intervention for fault handling

Production Factors for Federation cluster

Operators

1. Federation cluster
- monitoring and alerting
- Log analysis
- Control Plane Operation

2. Cluster Addons
- OpenKruise
- Argo Workflow

Network/Storage

1. Multi-cluster networking
- Calico
- Cloud vendors

2. Multi-cluster Service Mesh
- Istio
- Linkerd
- Consul
- Envoy

3. Multi-cluster storage
- Distributed file system
- Data migration
- Storage Performance
- Storage Capacity

Container Platform

1. Multi-cluster Application
- Partition Upgrade
- Rolling Upgrade
- Rollback

2. Application Migration
- Online business migration
- Online business rollback

3. Multi-cluster search
- data query performance
- Share and collaborate on data

4. Multi-cluster event
- Troubleshooting
- Data analysis
- decision support

Control Plane

1. Multi-cluster scheduling
- Cross-cluster scheduling
- Rescheduling
- Simulation scheduling

2. Multi-cluster RBAC
- Unified permission management
- Flexible authorization
- Security audit and monitoring

3. Multi-cluster HPA
- FedHPA
- CronHpa
- Manual scaling

Cluster addons OpenKruise

Intro： OpenKruise is an extended component suite for Kubernetes, which mainly focuses on automated management of large-scale applications, such as *deployment, upgrade, ops and availability protection*.

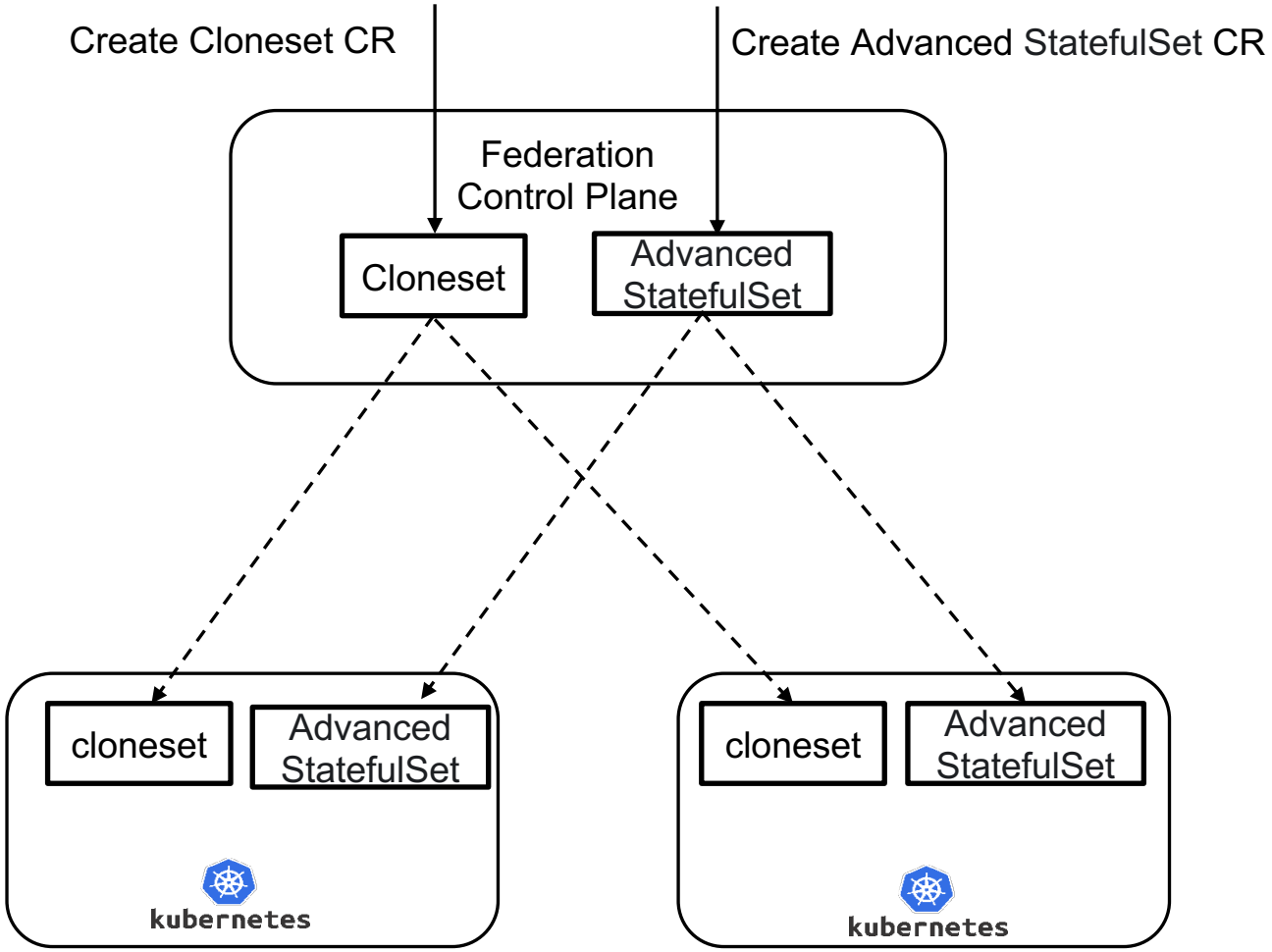
Openkruise in vivo

- CloneSet
- Advanced StatefulSet

Container platform function based on Openkruise

- Partition upgrade
- Rolling upgrade
- Rollback by partition
- In-place update
- MaxUnavailable and MaxSurge
- Application traffic is not interrupted
- Selective Pod deletion
- Scale up with rate limit

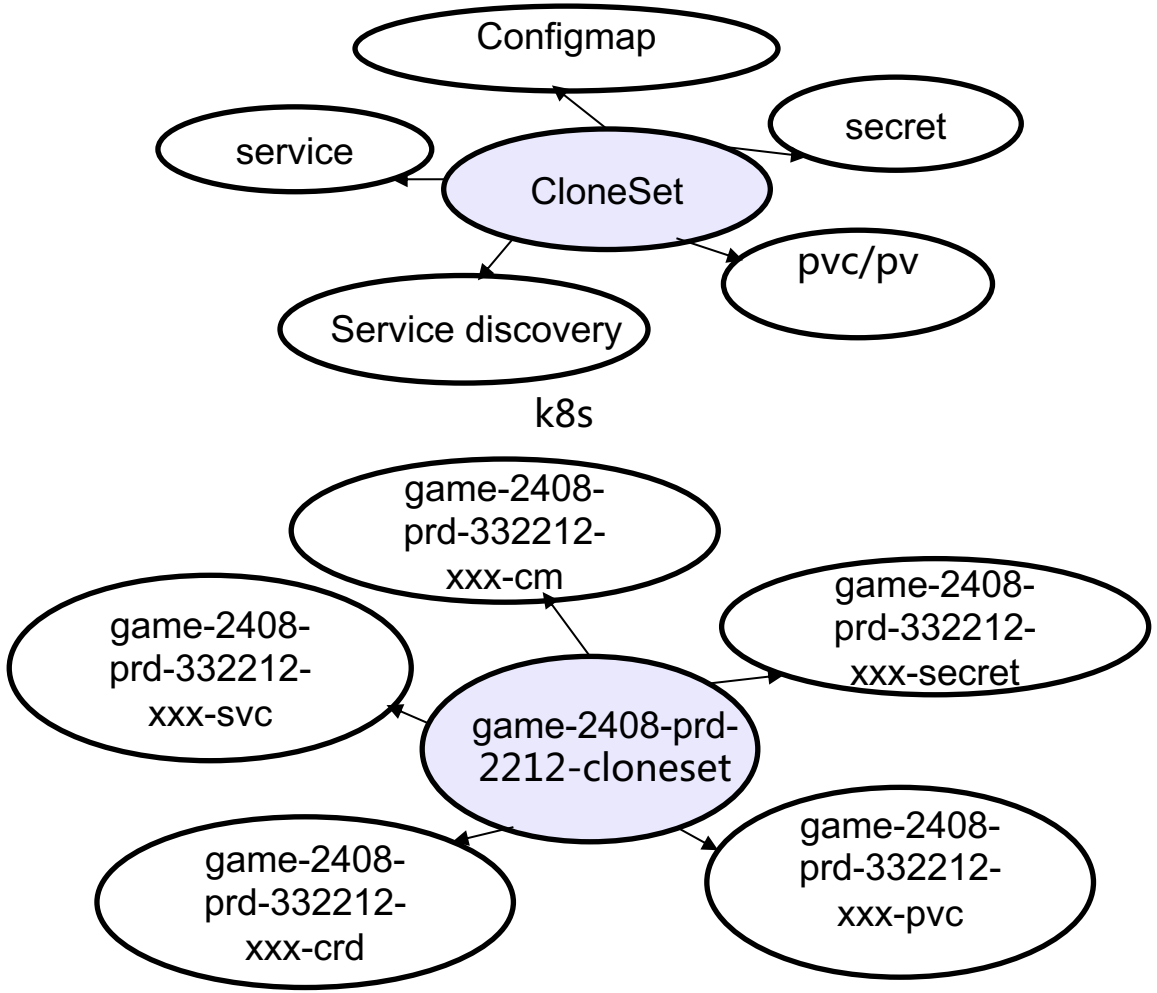
Integration OpenKruise



Setup OpenKruise

- Install Cloneset CRD and Advanced StatefulSet CRD on the Federation Control plane
- Deploy OpenKruise with chart in the member cluster
- Custom Cloneset and Advanced StatefulSet Interpreters by Lua scripts
- Deploy CloneSet and Advanced StatefulSet
- Get Cloneset and Advanced StatefulSet status on the Federation Control plane

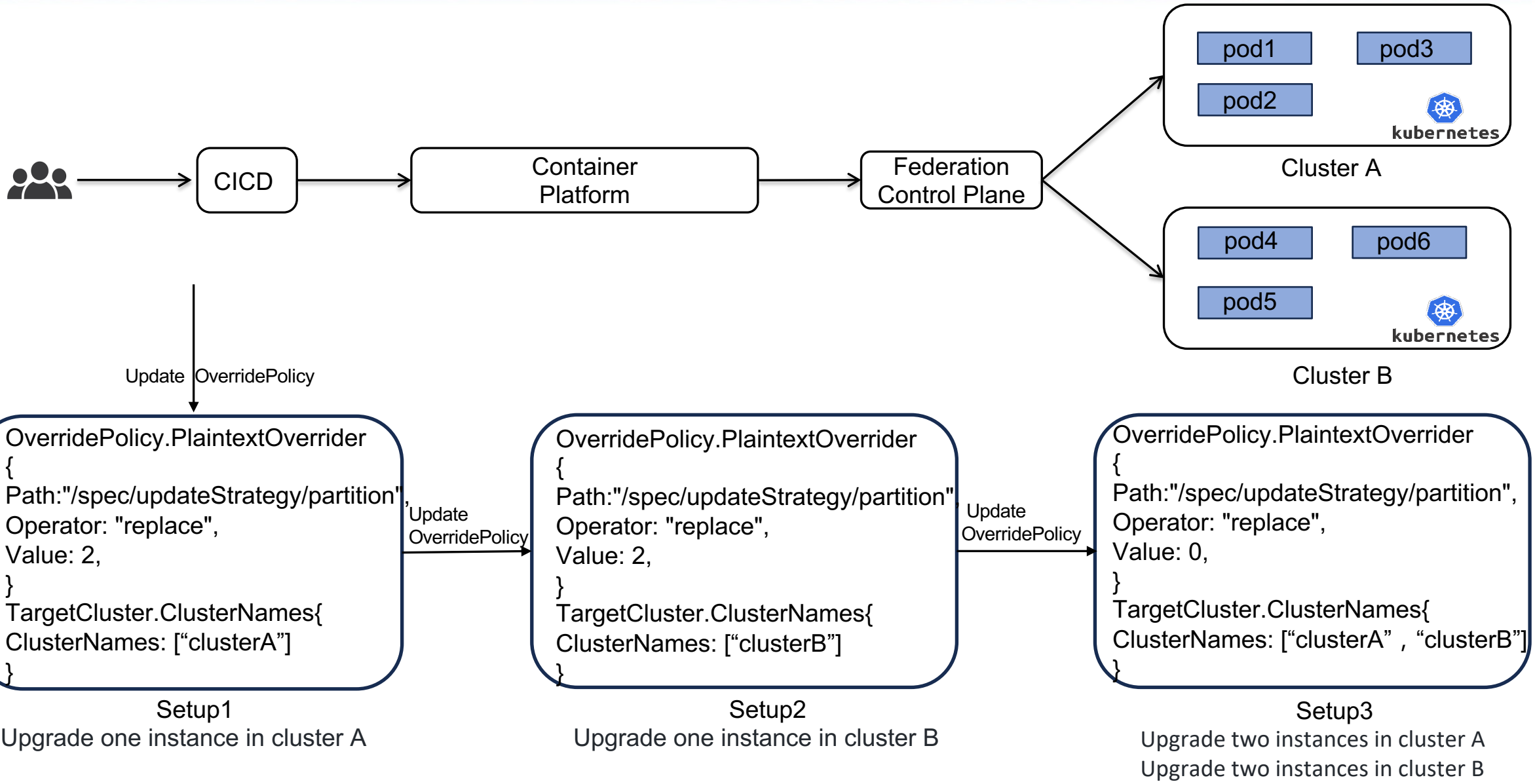
Overview of publishing your Application



update(OverridePolicy)

- ImageOverrider
  - in-place update
- CommandOverrider
- ArgsOverrider
- PlaintextOverrider
  - Rolling upgrade
  - Partition upgrade
  - Rollback by partition
  - MaxUnavailable and MaxSurge
  - Scale up with rate limit

Partition Upgrade Case

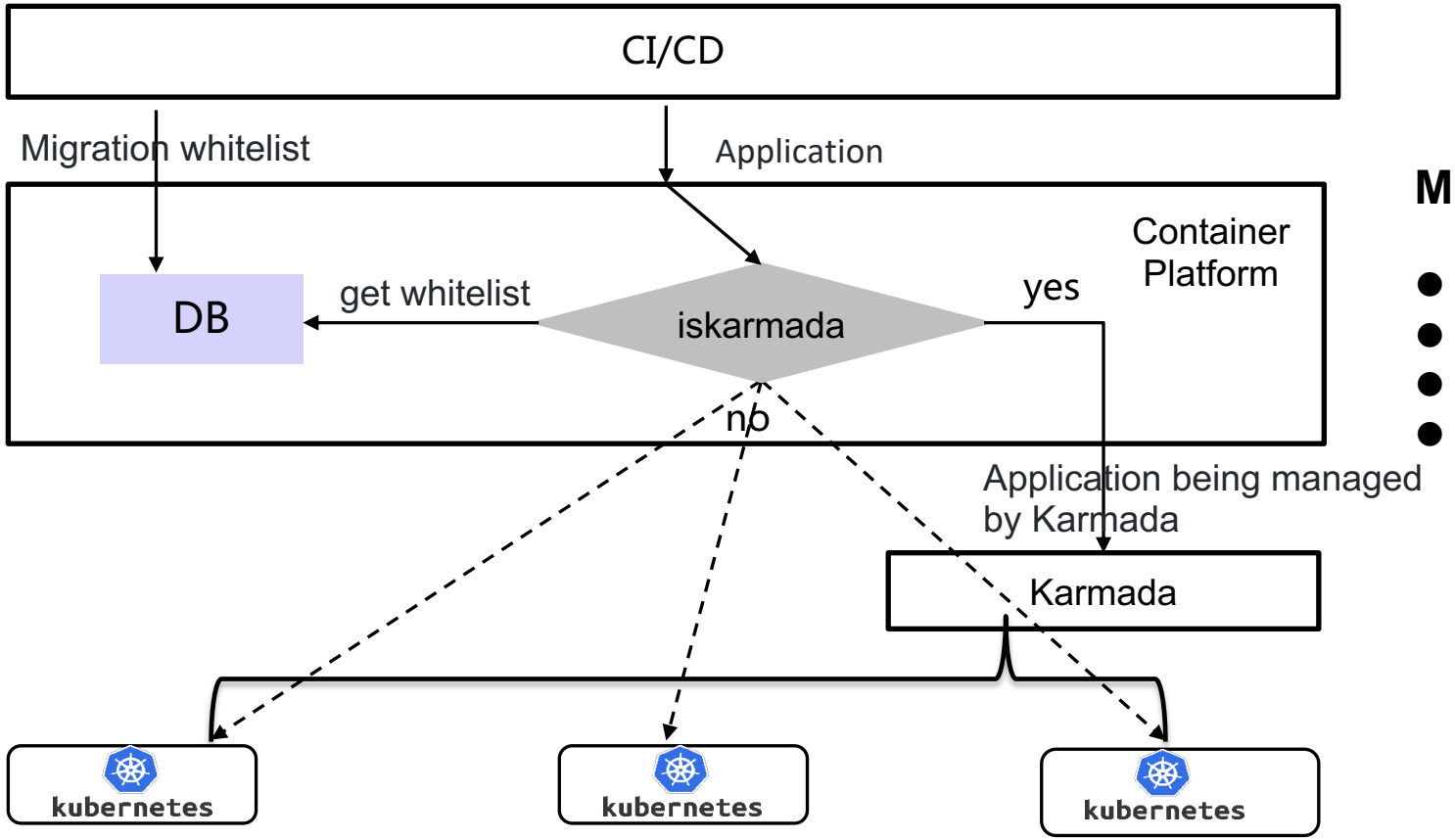


Setup1
Upgrade one instance in cluster A

Setup2
Upgrade one instance in cluster B

Setup3
Upgrade two instances in cluster A
Upgrade two instances in cluster B

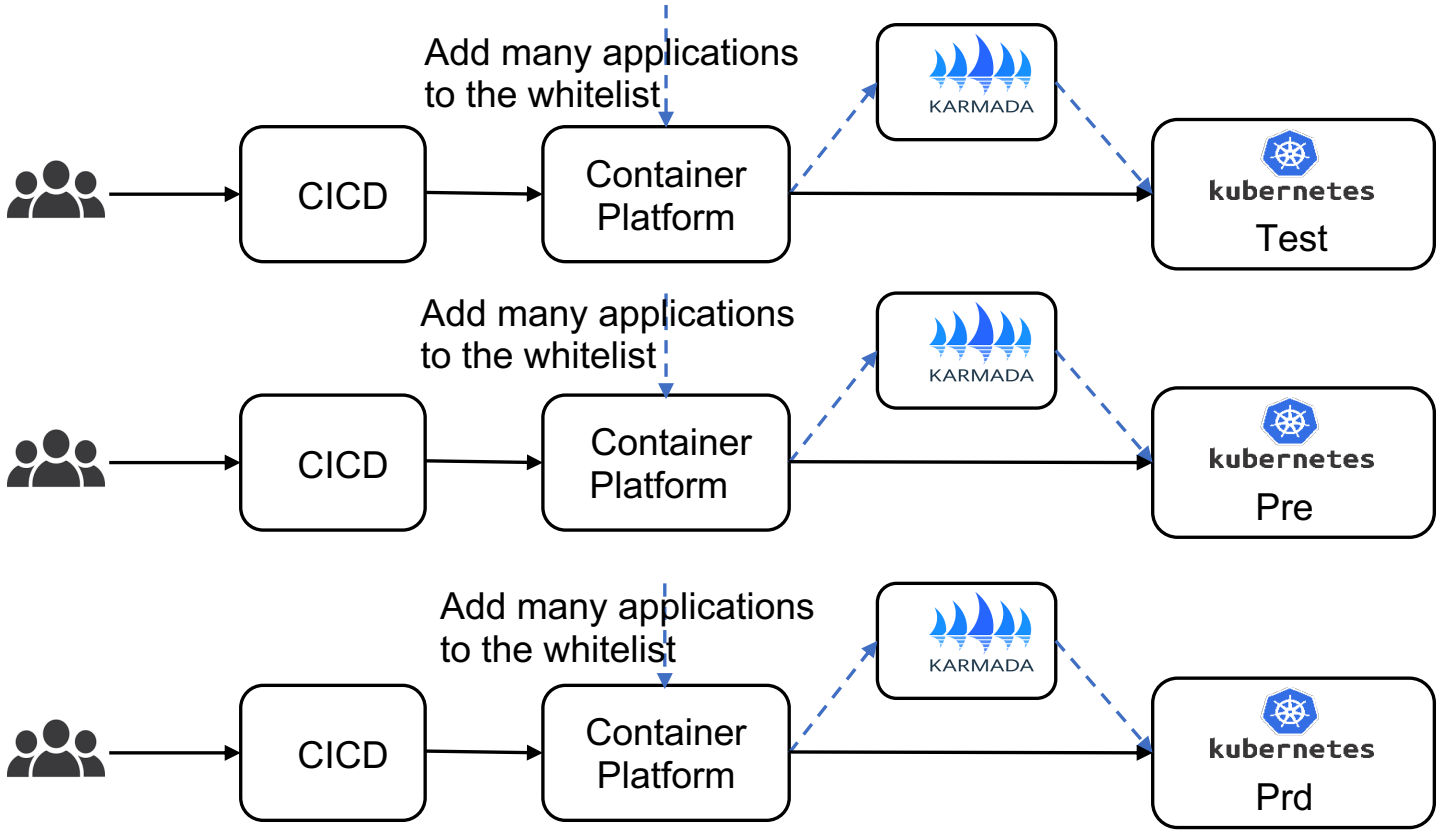
Online business migration



Migration process

- Add the application to the migration whitelist
- Deploy the application
- Manage the application through Karmada
- Finally, the application is deployed, and the user is completely unaware of the migration process. Both management methods are supported

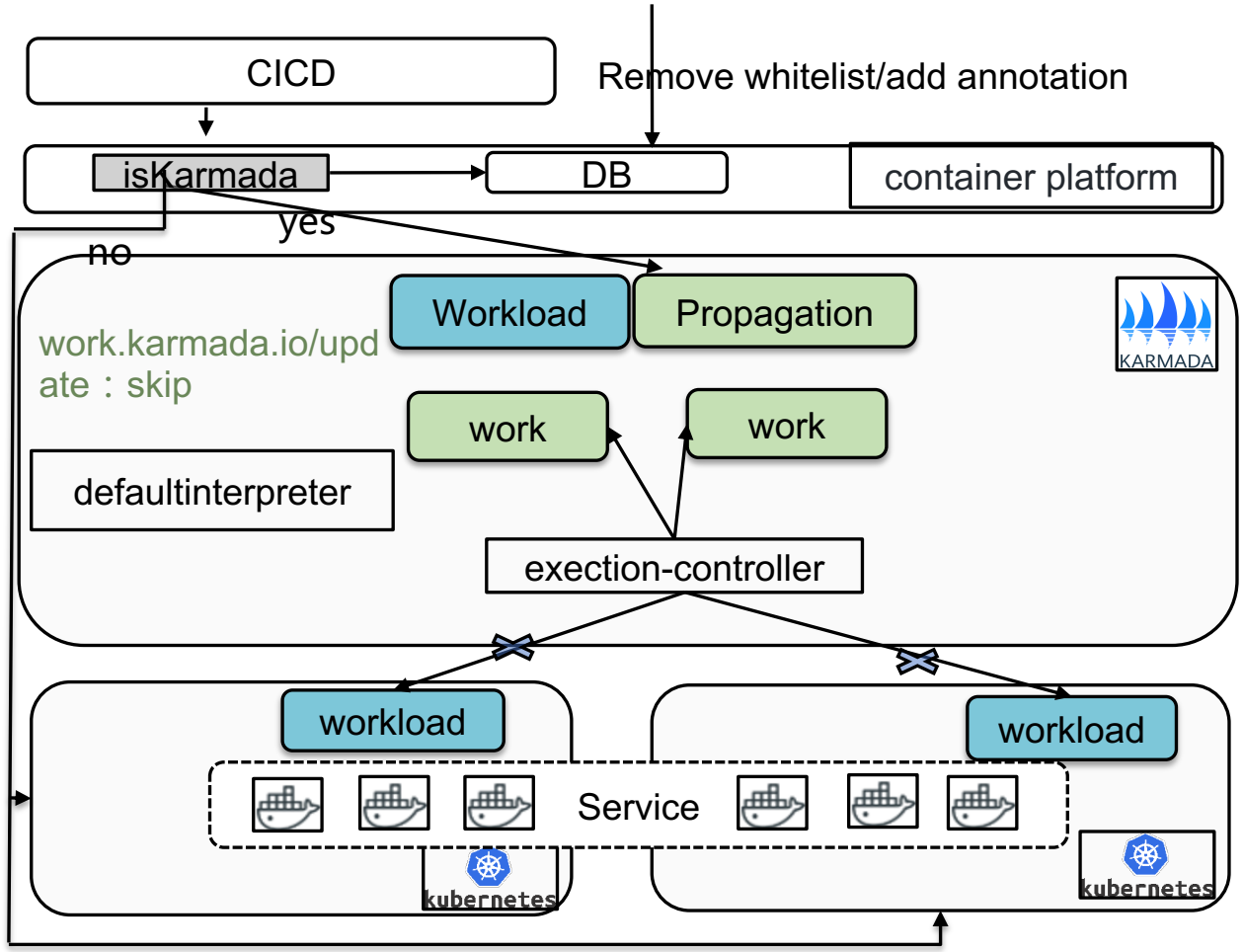
Strategies for online business migration



Principles of application migration

- Test first, then pre-release, and finally production.
- For major changes, perform gradual, staged, and proportional (1:2:7 ratio) gray release migration.
- Both parties responsible for the migration should perform point checks, verify, and observe monitoring for 5-10 minutes.
- Confirm whether there are any exceptions during the migration. If exceptions are found, follow the rollback process.

Online business Rollback



Why ?

- An error occurred during the migration
- An unknown error occurred on the control plane

exection-controller

- Observe the work object annotation
- Interrupting member cluster resource management


KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023
Q&A