KubeCon CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

Unlocking Scalability and Simplifying Multi-Cloud Management with Karmada and PipeCD

Hongcai Ren, Huawei, Karmada Community
Khanh Tran, CyberAgent, PipeCD Community

Agenda

- Why multi-cluster
- Introduction of Karmada
- Introduction of PipeCD
- Karmada x PipeCD Integration demo

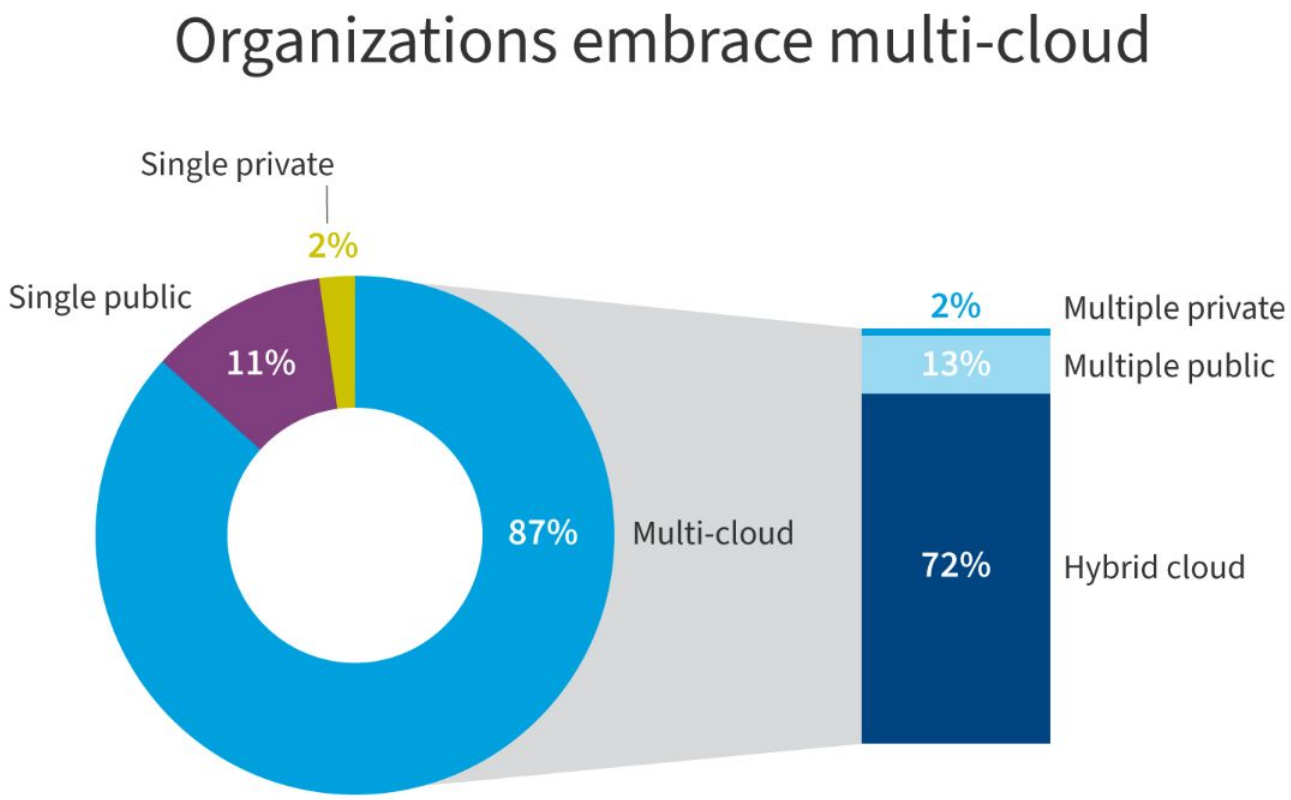
Agenda

- **Why multi-cluster**
- Introduction of Karmada
- Introduction of PipeCD
- Karmada x PipeCD Integration demo

Why multi-cluster

- Location
  - Latency: Deploy apps as close to the customers as possible
  - Data locality: Keep user data in-country
- Isolation
  - Environment (e.g. dev, test, prod)
  - Security isolation: sensitive data must be isolated
  - Organization isolation: Teams have different domains
- Reliability
  - Blast radius: Apps incident in one cluster must not impact the whole system
  - Scale: The apps is too big to fit in a single cluster
- And so on ...

Multi-Cloud and Multi-Cluster Deployment Has Become a Common Practice



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

- More than 87% of enterprise respondents are using the services of multiple cloud vendors at the same time.
- Cloud native technologies and the cloud market are maturing, and an era of programmable multi-cloud management services is coming.

Challenges of Being Cloud-Native Multi-Cloud

Challenges to multi-cloud container cluster management

Numerous clusters

Complex and repeated cluster configurations

Cluster management varies from vendor to vendor

Fragmented API access entries

Scattered services

Application configuration differs across clusters

Cross-cloud service access

Application synchronization between clusters

Restrictions from clusters

Restricted resource scheduling

Restricted application availability

Restricted auto scaling

Vendor lock-in

Service deployment dependency

Lack of automatic failover

Lack of neutral open-source,

multi-cluster orchestration projects

Agenda

- Why multi-cluster
- **Introduction of Karmada**
- Introduction of PipeCD
- Karmada x PipeCD Integration demo

Karmada: Open-Source, Cloud-Native Platform for Multi-Cloud Container Orchestration

Compatible with Kubernetes native APIs

1

- Upgrade from single-cluster to multi-cluster deployments without code refactoring
- Seamlessly integrated with the Kubernetes single-cluster tool chain

2

Out-of-the-box

- Built-in policy sets for multiple scenarios,such as geo-redundancy,
- intra-city active-active, and remote DR

No vendor lock-in & **Centralized management**

3

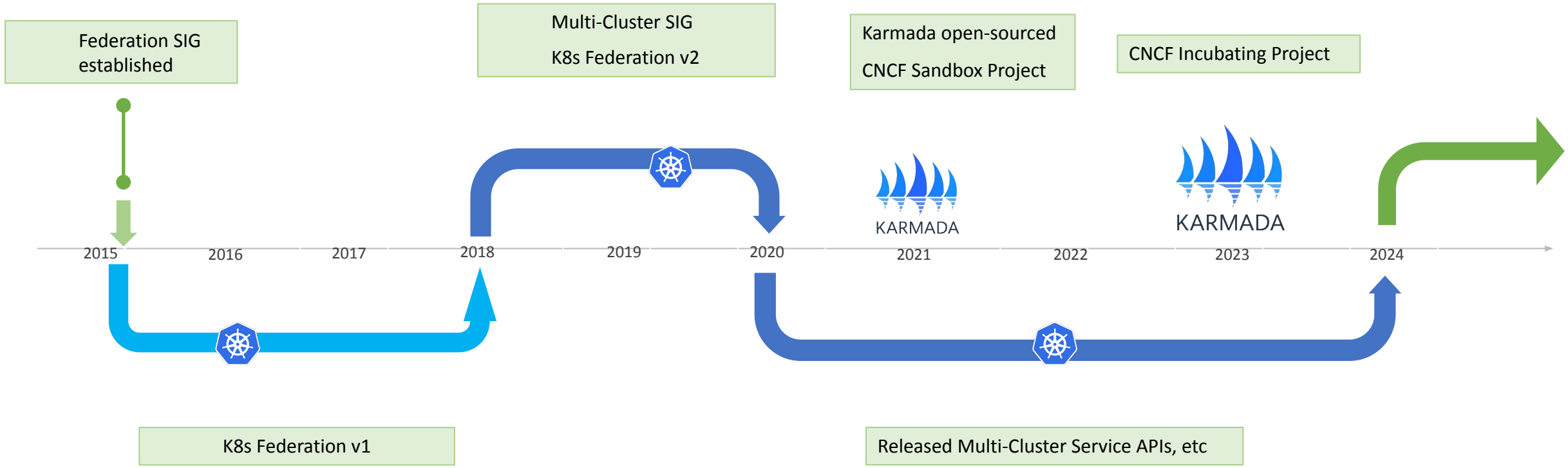
- Support for multi-cloud platforms, auto resource allocation, and free migration
- Not bound to any commercial products from cloud vendors
- Support public clouds, private clouds, and edge clouds

4

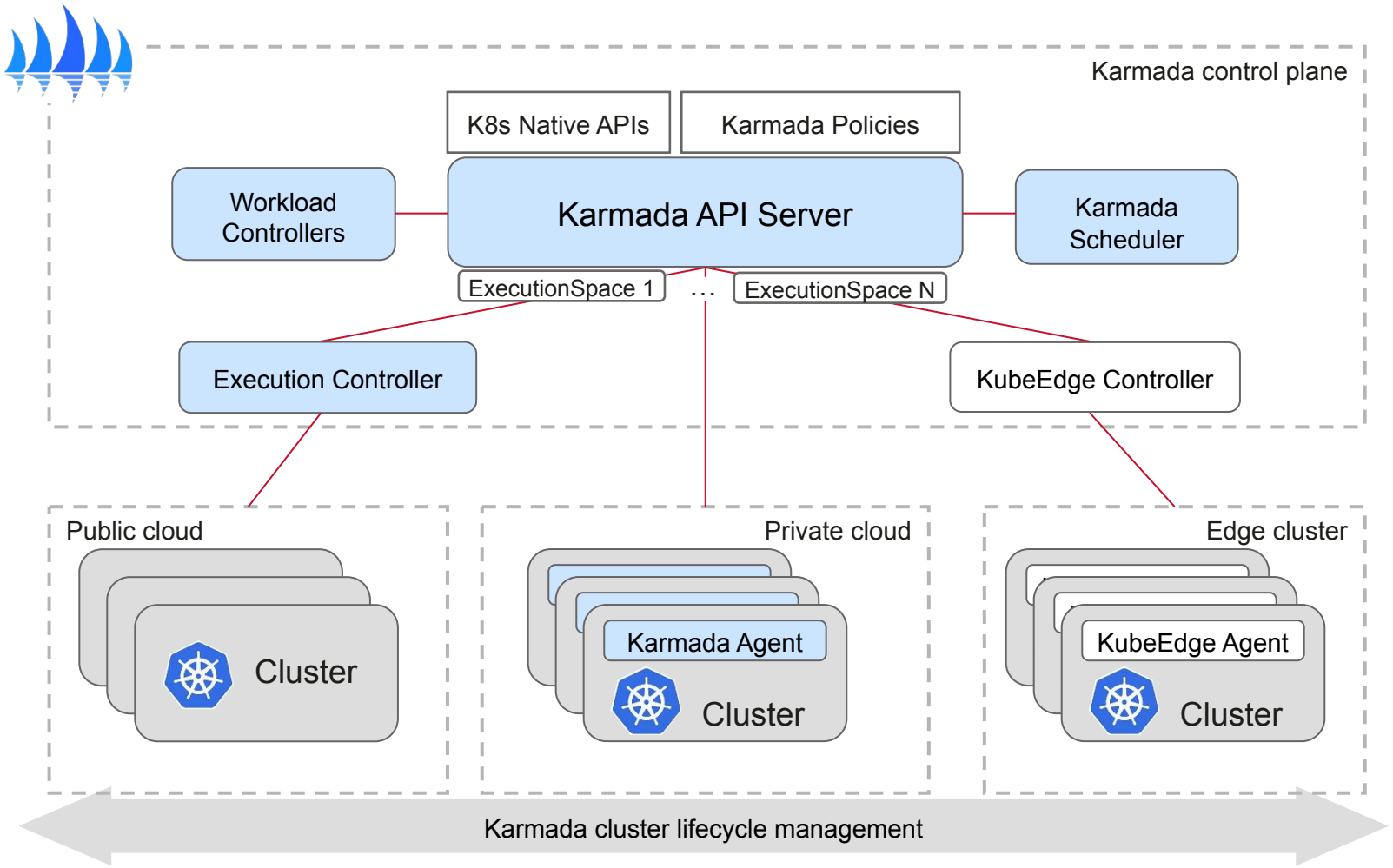
Various multi-cluster scheduling policies

- Cluster scheduling based on affinity and multi-cluster
- HA deployment across regions, AZs, clusters, and vendors

Development of Multi-Cluster Container Orchestration



Karmada Overview



Core Features

Multi-cluster management

1. Support cluster in Public cloud,on-prem or edge
2. Unified management
3. Multi-Vendor support including Huawei Cloud CCE, AWS EKS, and Other K8s clusters.
4. Push and Pull mode support
5. X86 & ARM64 support

Cross-cloud deployment

1. Use Kubernetes native API, including CRD
2. Facilitating cross-cloud application deployment, release, and operations capabilities for users.
3. Use built-in policy sets for scenarios, including: Active-active, Remote Disaster Recovery, etc

Cross-cloud App failover

Provide the ability for failover and address disaster recovery concerns in multi-cloud applications.

Centralized Management

Access cluster's resource from Karmada control plane (get/describe/logs/exec, etc)

Multi-cluster service and LB

Deploy service across cluster by leveraging MCS-API

Zero Refactoring: Using Kubernetes Native APIs to Deploy a Multi-Cluster Application

Widely applicable propagation policy

```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: multi-zone-replication
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      labelSelector:
        matchLabels:
          ha-mode: multi-zone-replication
  placement:
    spreadConstraints:
      - spreadByField: zone
        maxGroups: 2
        minGroups: 2
      - spreadByField: cluster
        maxGroups: 3
        minGroups: 3
```

Example policy: Configure a multi-AZ HA deployment scheme for all Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  app: nginx
  ha-mode: multi-zone-replication
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
        ports:
        - containerPort: 80
```

Deploy apps with standard Kubernetes API definitions

kubectl create -f nginx-deployment.yaml

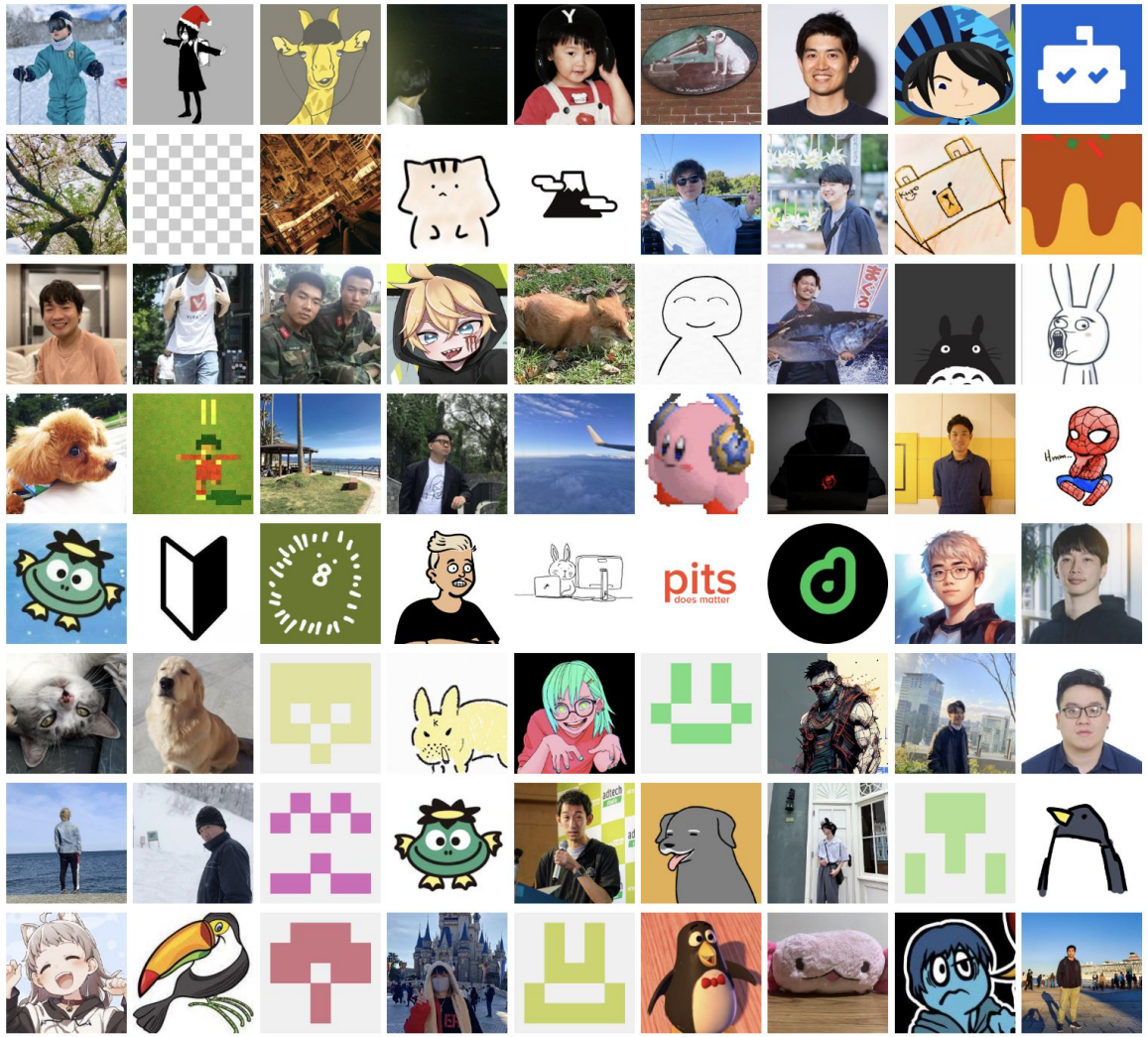
Agenda

- Why multi-cluster
- Introduction of Karmada
- **Introduction of PipeCD**
- Karmada x PipeCD Integration demo

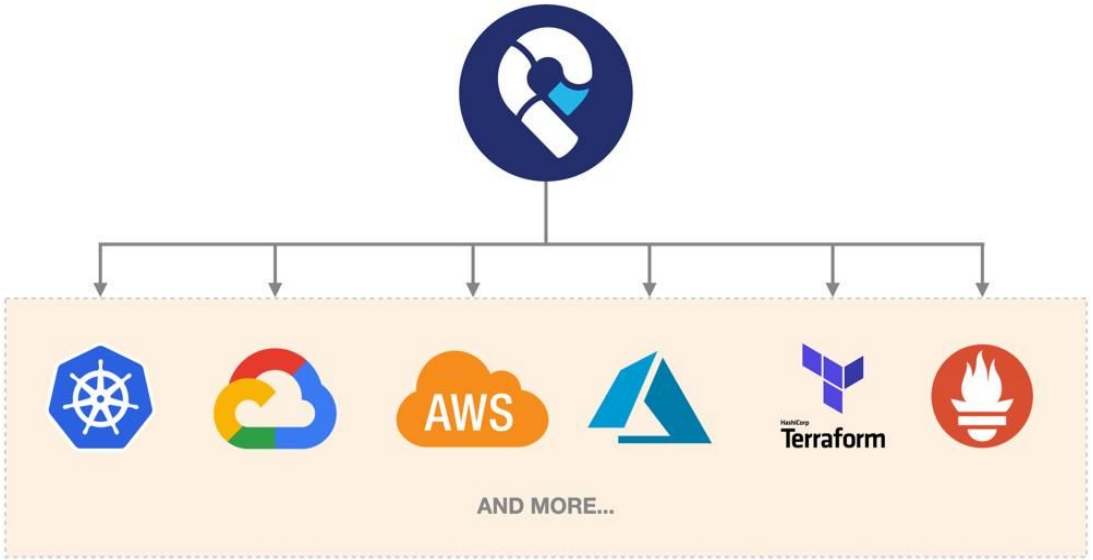
Introduction of PipeCD

PipeCD

- 4000s commits / 90 contributors
- CNCF Sandbox from 2023



What is PipeCD

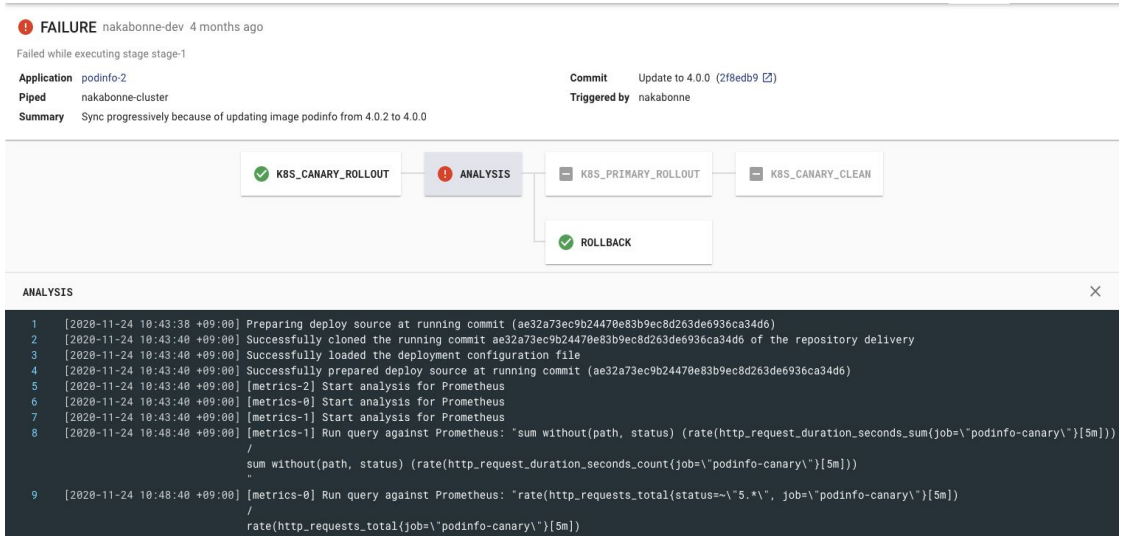


The One CD for All {applications, platforms, operations}

- A GitOps style continuous delivery platform that provides consistent deployment and operations experience for any applications
- Support multi-clouds providers, multi-analysis providers
- Built-in secret management/attachment for manifests manipulating
- Event-based communication mechanism to work with whatever CI
- And it works for not just Kubernetes workloads but other as well

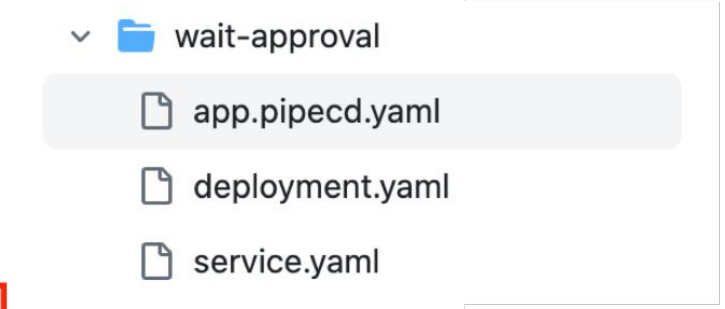
Build your own Progressive Delivery Pipeline

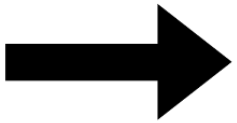
- Built In support for Progressive Delivery (Canary, Blue/Green…)
- Automatically analyze, rollout with metrics

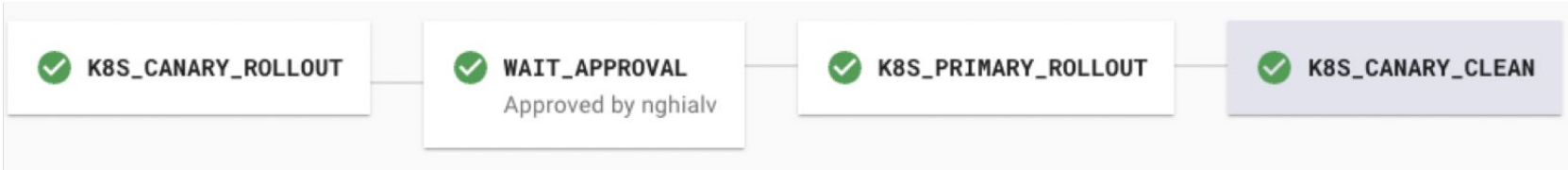


Zero Refactoring: Using your Kubernetes manifests to perform Progressive Delivery


```yaml
apiVersion: pipecd.dev/v1beta1
kind: KubernetesApp
spec:
  name: wait-approval
  labels:
    env: example
    team: product
  pipeline:
    stages:
      - name: K8S_CANARY_ROLLOUT
        with:
          replicas: 10%
      - name: WAIT_APPROVAL
        with:
          approvers:
            - khanhtc1202
      - name: K8S_PRIMARY_ROLLOUT
      - name: K8S_CANARY_CLEAN
```

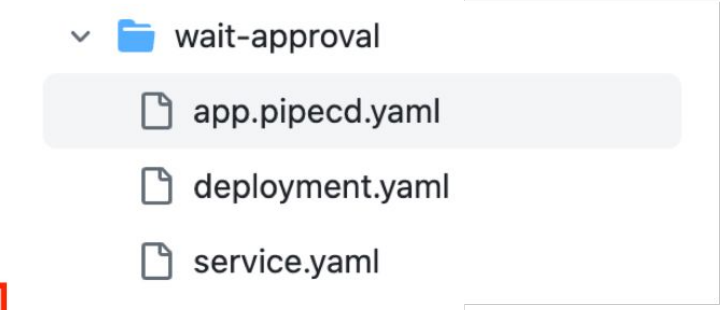






Zero Refactoring: Using your Kubernetes manifests to perform Progressive Delivery

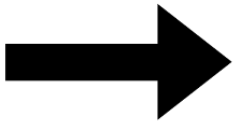

```yaml
apiVersion: pipecd.dev/v1beta1
kind: KubernetesApp
spec:
  name: wait-approval
  labels:
    env: example
    team: product
  pipeline:
    stages:
      - name: K8S_CANARY_ROLLOUT
        with:
          replicas: 10%
      - name: WAIT_APPROVAL
        with:
          approvers:
            - khanhtc1202
      - name: K8S_PRIMARY_ROLLOUT
      - name: K8S_CANARY_CLEAN
```

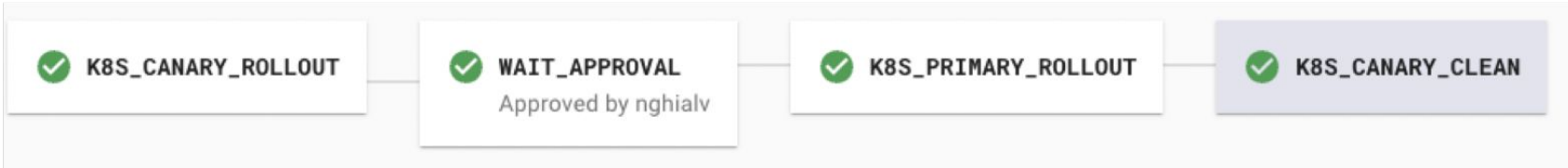


HELM

Kustomize.io

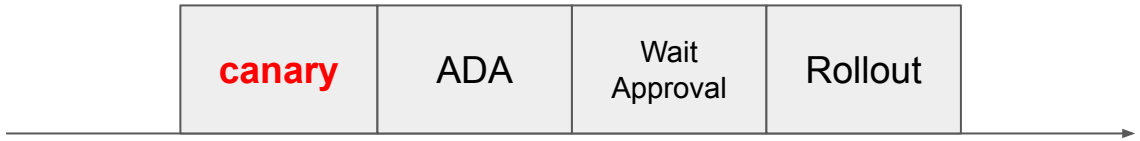
Istio





Less effort to Progressive Delivery

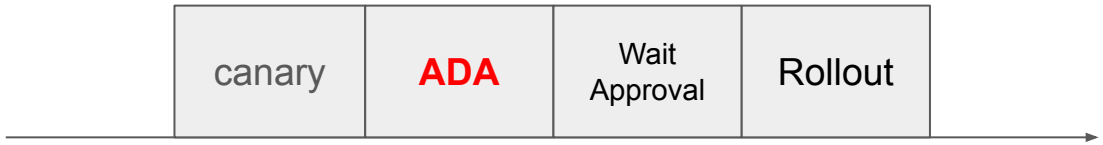
pipecd: PIPELINE_SYNC




```yaml
pipeline:
  stages:
    - name: K8S_CANARY_ROLLOUT
      with:
        replicas: 10%
    - name: K8S_BASELINE_ROLLOUT
      with:
        replicas: 10%
    - name: ANALYSIS
      with:
        duration: 15m
        metrics:
          - template:
              name: http_error_rate_canary_baseline
              appArgs:
                podNamePrefix:
          # - template:
          #     name: http_4xx_error_rate_threshold
          #     appArgs:
          #       podNamePrefix:
          #       threshold: "0.025"
    - name: K8S_BASELINE_CLEAN
    - name: K8S_CANARY_CLEAN
    - name: WAIT_APPROVAL
      with:
        timeout: 1h
    - name: K8S_PRIMARY_ROLLOUT
```

Less effort to Progressive Delivery

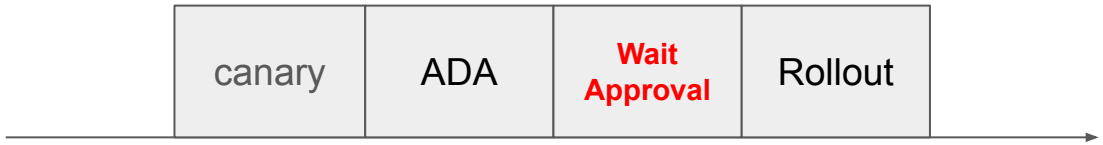
pipecd: PIPELINE_SYNC




```
pipeline:
  stages:
    - name: K8S_CANARY_ROLLOUT
      with:
        replicas: 10%
    - name: K8S_BASELINE_ROLLOUT
      with:
        replicas: 10%
    - name: ANALYSIS
      with:
        duration: 15m
        metrics:
          - template:
              name: http_error_rate_canary_baseline
              appArgs:
                podNamePrefix:
          # - template:
          #     name: http_4xx_error_rate_threshold
          #     appArgs:
          #       podNamePrefix:
          #       threshold: "0.025"
    - name: K8S_BASELINE_CLEAN
    - name: K8S_CANARY_CLEAN
    - name: WAIT_APPROVAL
      with:
        timeout: 1h
    - name: K8S_PRIMARY_ROLLOUT
```

Less effort to Progressive Delivery

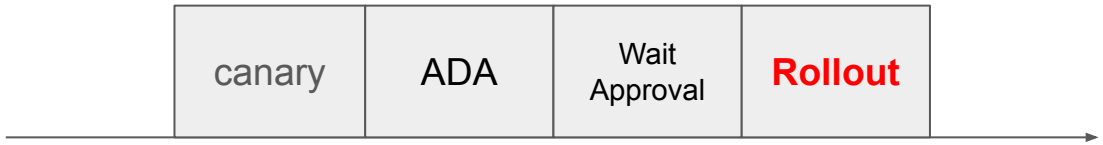
pipecd: PIPELINE_SYNC




```
pipeline:
  stages:
    - name: K8S_CANARY_ROLLOUT
      with:
        replicas: 10%
    - name: K8S_BASELINE_ROLLOUT
      with:
        replicas: 10%
    - name: ANALYSIS
      with:
        duration: 15m
        metrics:
          - template:
              name: http_error_rate_canary_baseline
              appArgs:
                podNamePrefix:
          # - template:
          #     name: http_4xx_error_rate_threshold
          #     appArgs:
          #       podNamePrefix:
          #       threshold: "0.025"
    - name: K8S_BASELINE_CLEAN
    - name: K8S_CANARY_CLEAN
    - name: WAIT_APPROVAL
      with:
        timeout: 1h
    - name: K8S_PRIMARY_ROLLOUT
```

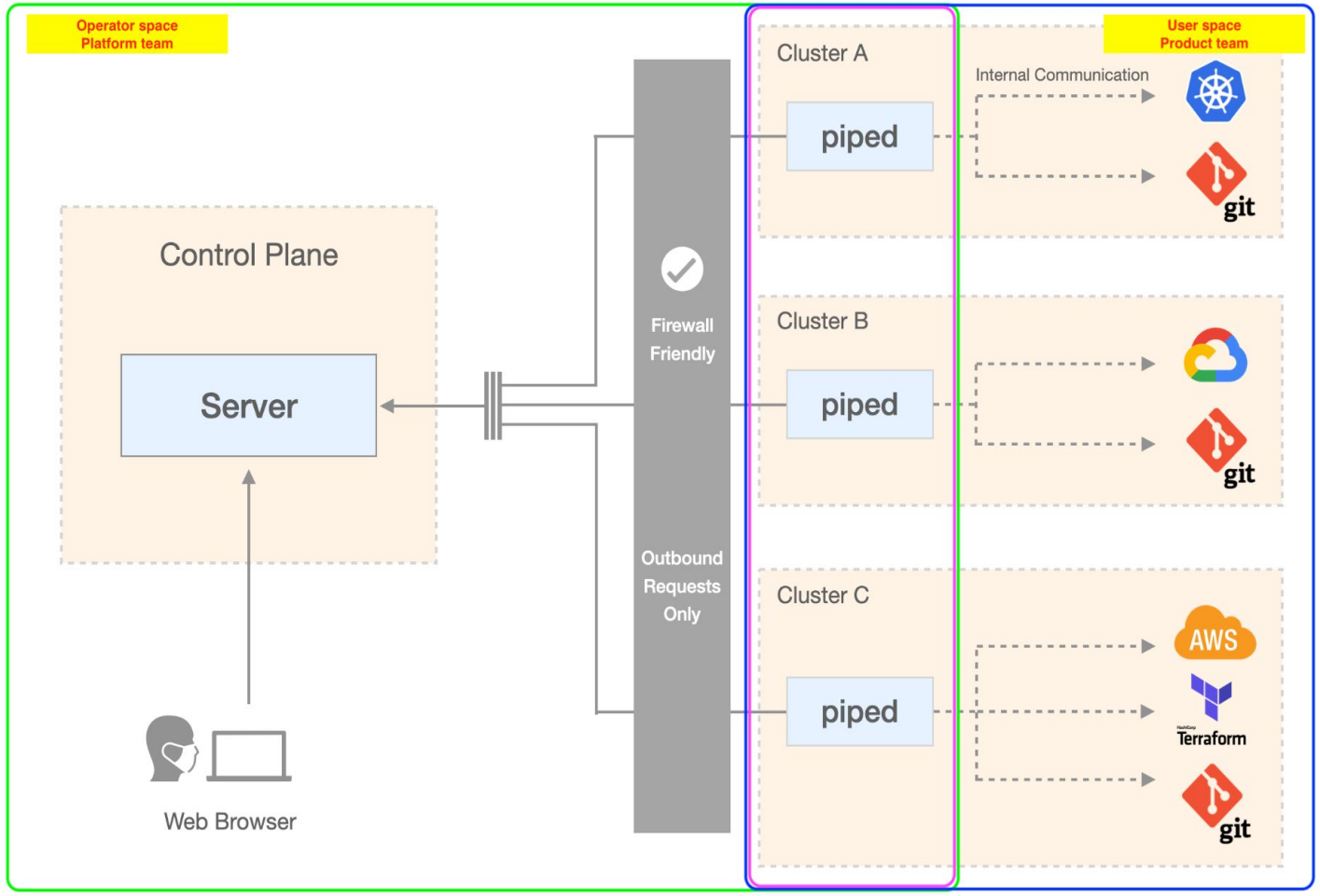
Less effort to Progressive Delivery

pipecd: PIPELINE_SYNC




```
pipeline:
  stages:
    - name: K8S_CANARY_ROLLOUT
      with:
        replicas: 10%
    - name: K8S_BASELINE_ROLLOUT
      with:
        replicas: 10%
    - name: ANALYSIS
      with:
        duration: 15m
        metrics:
          - template:
              name: http_error_rate_canary_baseline
              appArgs:
                podNamePrefix:
          # - template:
          #     name: http_4xx_error_rate_threshold
          #     appArgs:
          #       podNamePrefix:
          #       threshold: "0.025"
    - name: K8S_BASELINE_CLEAN
    - name: K8S_CANARY_CLEAN
    - name: WAIT_APPROVAL
      with:
        timeout: 1h
    - name: K8S_PRIMARY_ROLLOUT
```

Why PipeCD for Multi-cluster/cloud

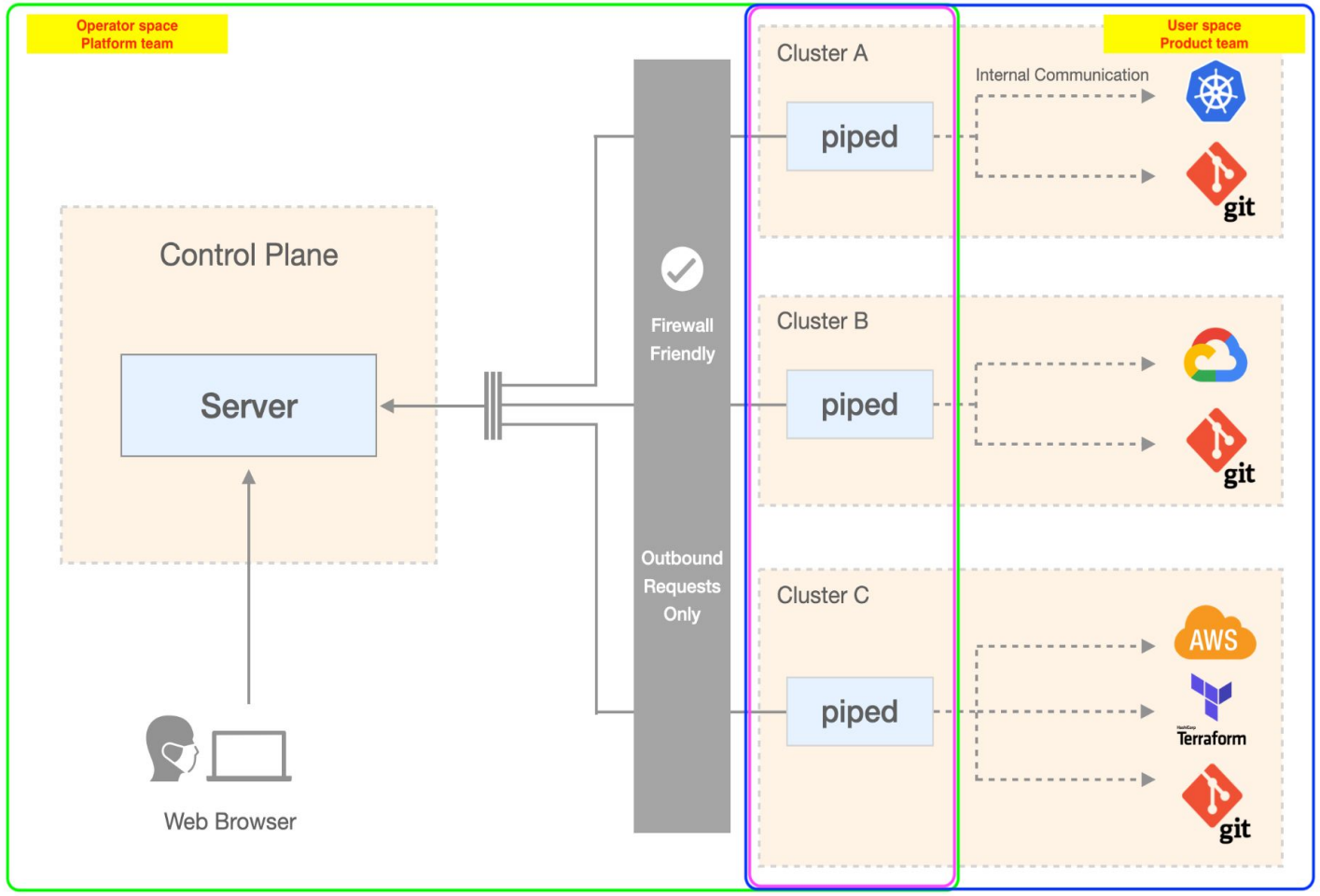


Support Multi-clusters/clouds at mind

Easy to operate multi-cluster, multi-tenancy by separating control-plane and piped

- Single Control-Plane for centric state management
- Agents installed freely next to your applications in clusters (or not)

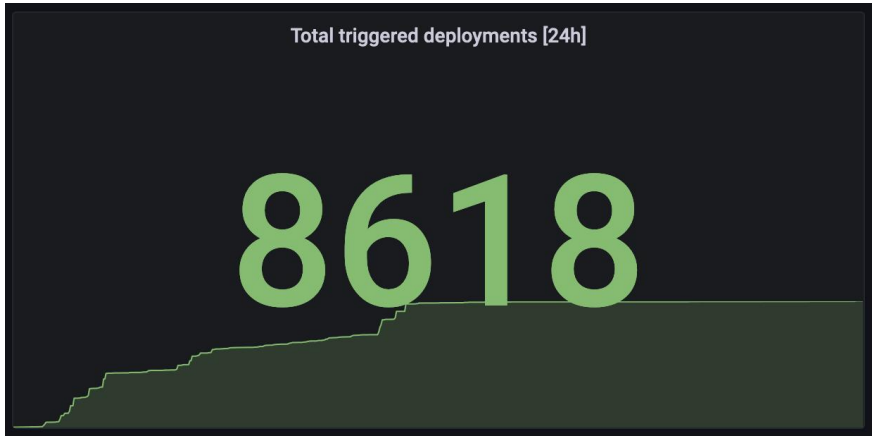
Why PipeCD for Multi-cluster/cloud



Support Multi-clusters/clouds at mind

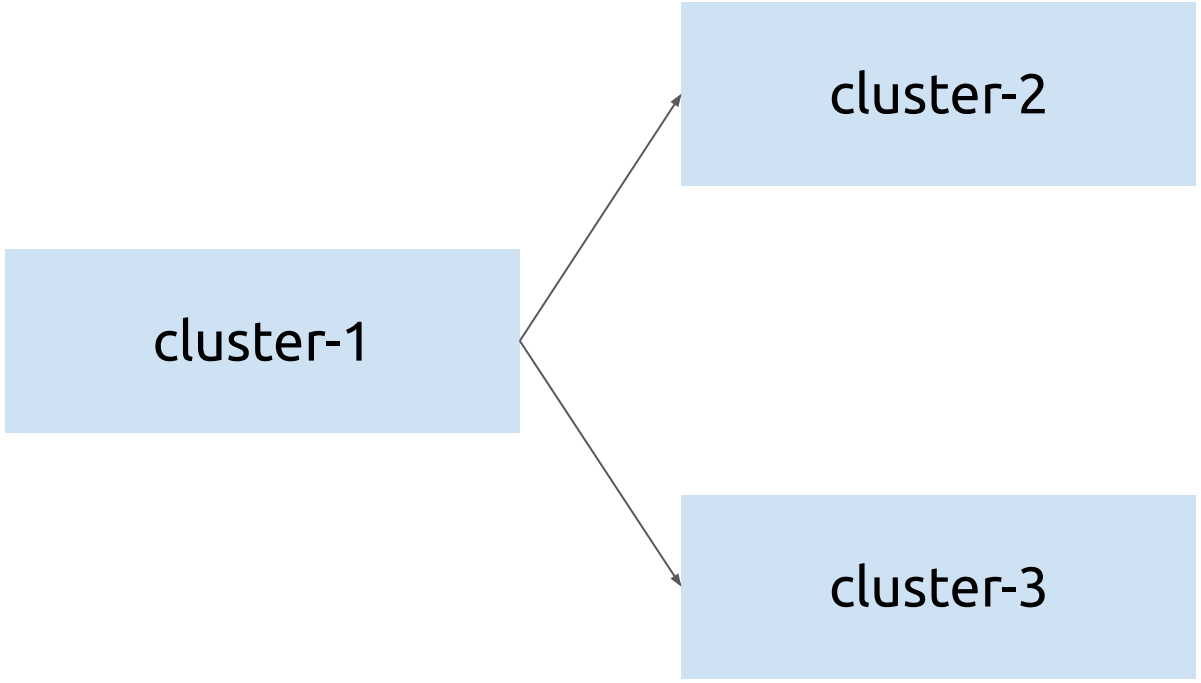
Easy to operate multi-cluster, multi-tenancy by separating control-plane and piped

- Single Control-Plane for centric state management
- Agents installed freely next to your applications in clusters (or not)



Why PipeCD for Multi-cluster/cloud

pipecd: Deployment Chain



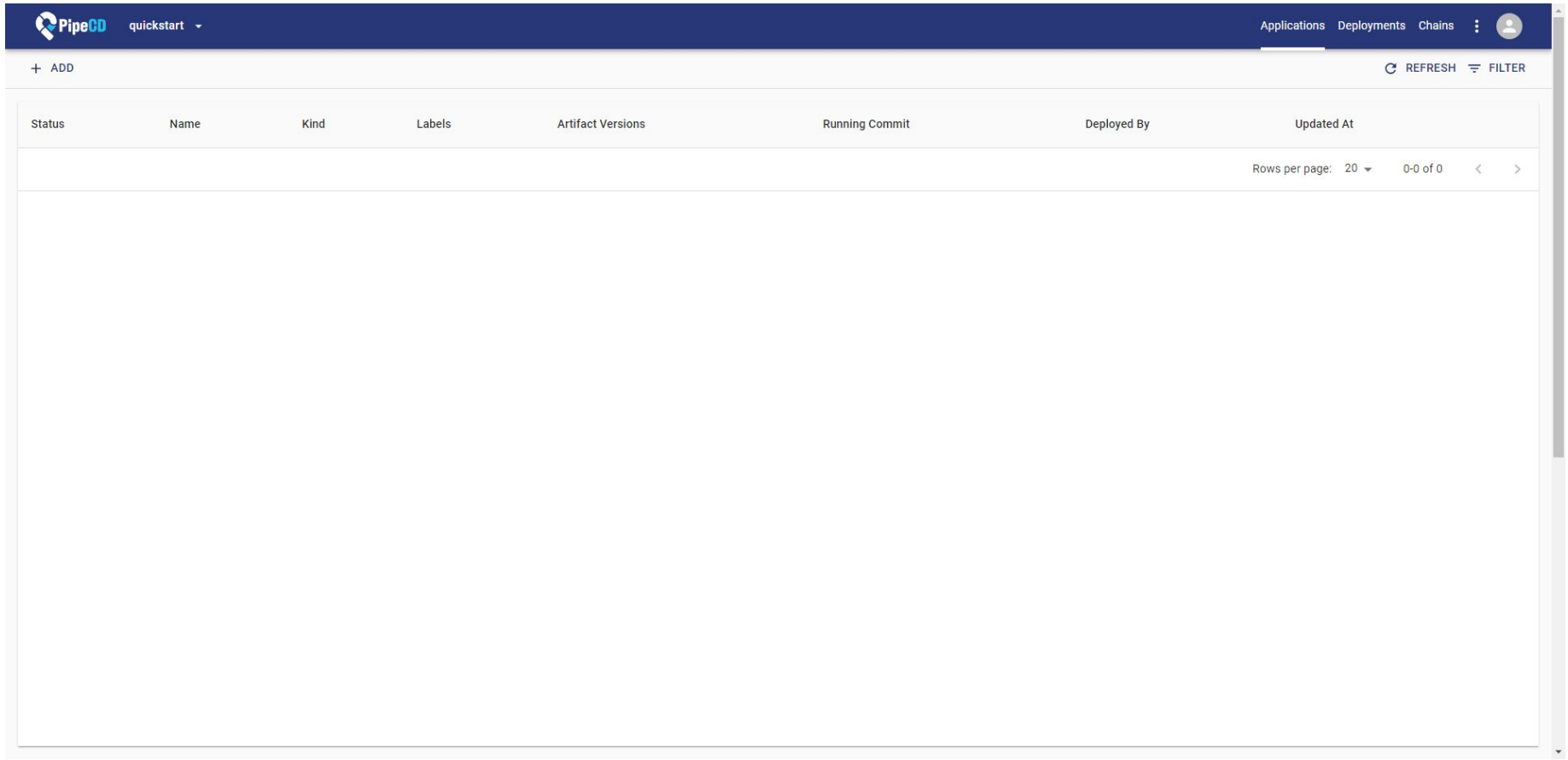

```yaml
apiVersion: pipecd.dev/v1beta1
kind: KubernetesApp
spec:
  name: app-1
  labels:
    env: cluster-1
  pipeline:
    stages:
      ...
  postSync:
    chain:
      applications:
        - name: app-1
          kind: KUBERNETES
          labels:
            env: cluster-2
        - name: app-1
          kind: KUBERNETES
          labels:
            env: cluster-3

```

Agenda

- Why multi-cluster
- Introduction of Karmada
- Introduction of PipeCD
- **Karmada x PipeCD Integration demo**

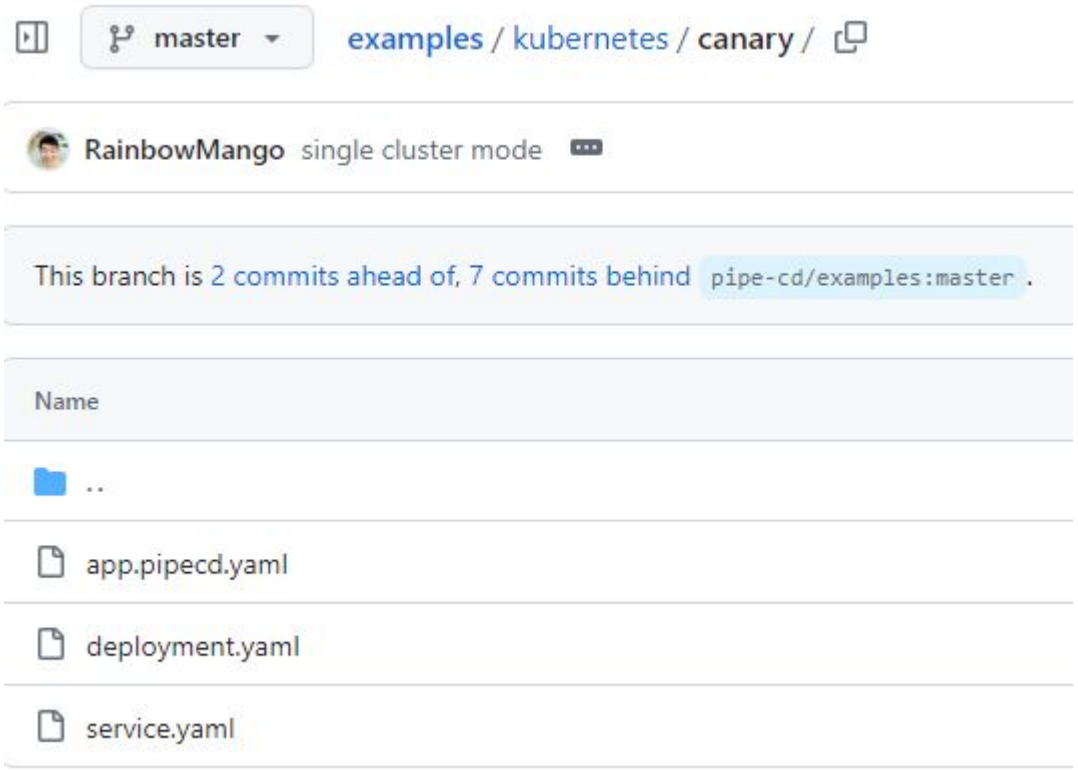
Demo of PipeCD

Install and launch the PipeCD



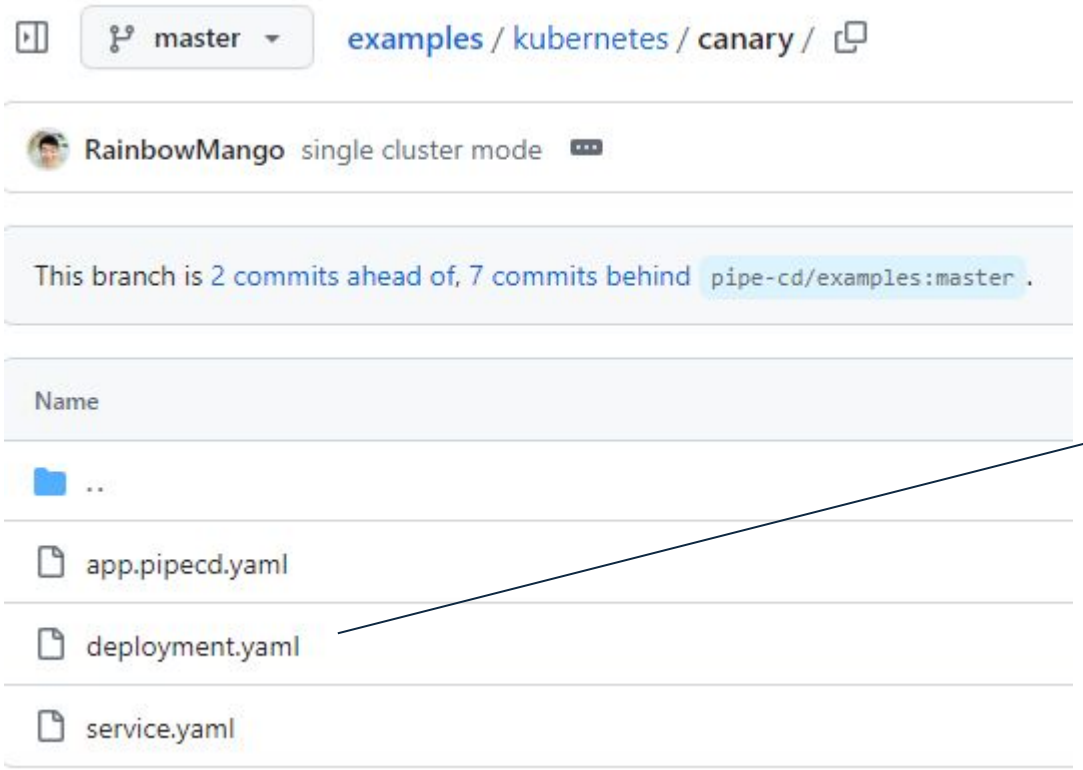
Demo of PipeCD

Prepare application as well as PipeCD configuration



Demo of PipeCD

Prepare application as well as PipeCD configuration




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: canary
  labels:
    app: canary
spec:
  replicas: 2
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: canary
      pipecd.dev/variant: primary
  template:
    metadata:
      labels:
        app: canary
        pipecd.dev/variant: primary
    spec:
      containers:
      - name: helloworld
        image: ghcr.io/pipe-cd/helloworld:v0.32.0
        args:
          - server
        ports:
        - containerPort: 9085
```

Demo of PipeCD

Prepare application as well as PipeCD configuration

master examples / kubernetes / canary /

RainbowMango single cluster mode

This branch is 2 commits ahead of, 7 commits behind `pipe-cd/examples:master`.

Name

..

app.pipecd.yaml

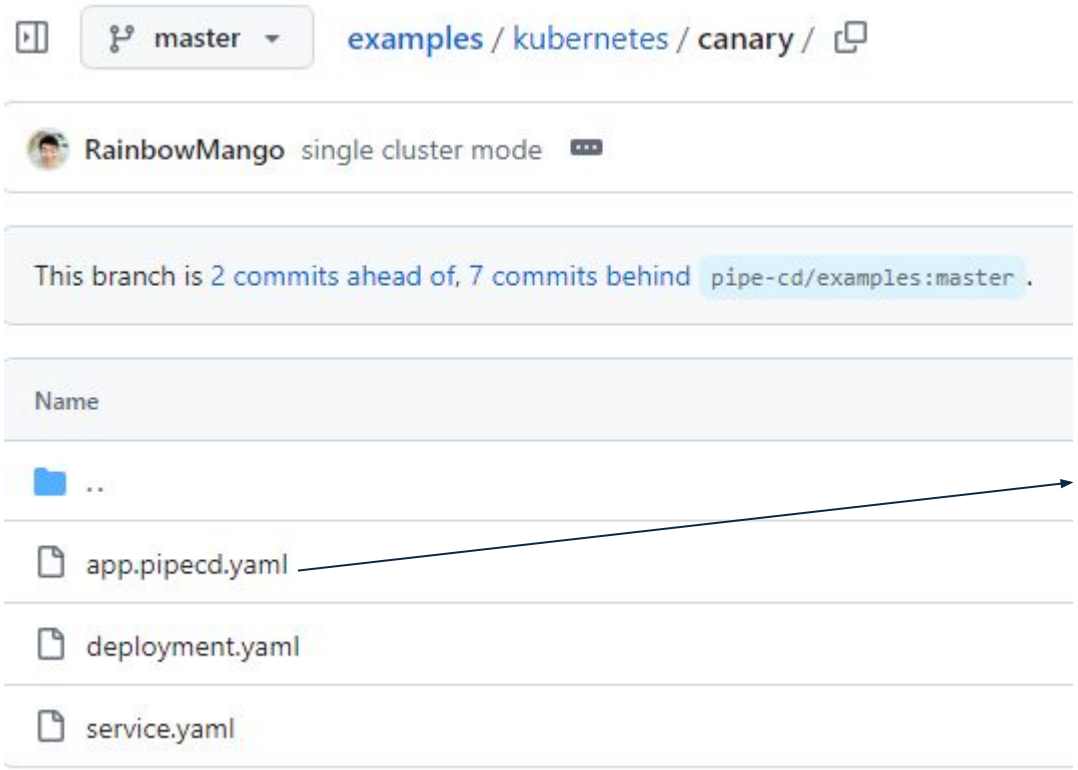
deployment.yaml

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: canary
spec:
  selector:
    app: canary
  ports:
    - protocol: TCP
      port: 9085
      targetPort: 9085
```

Demo of PipeCD

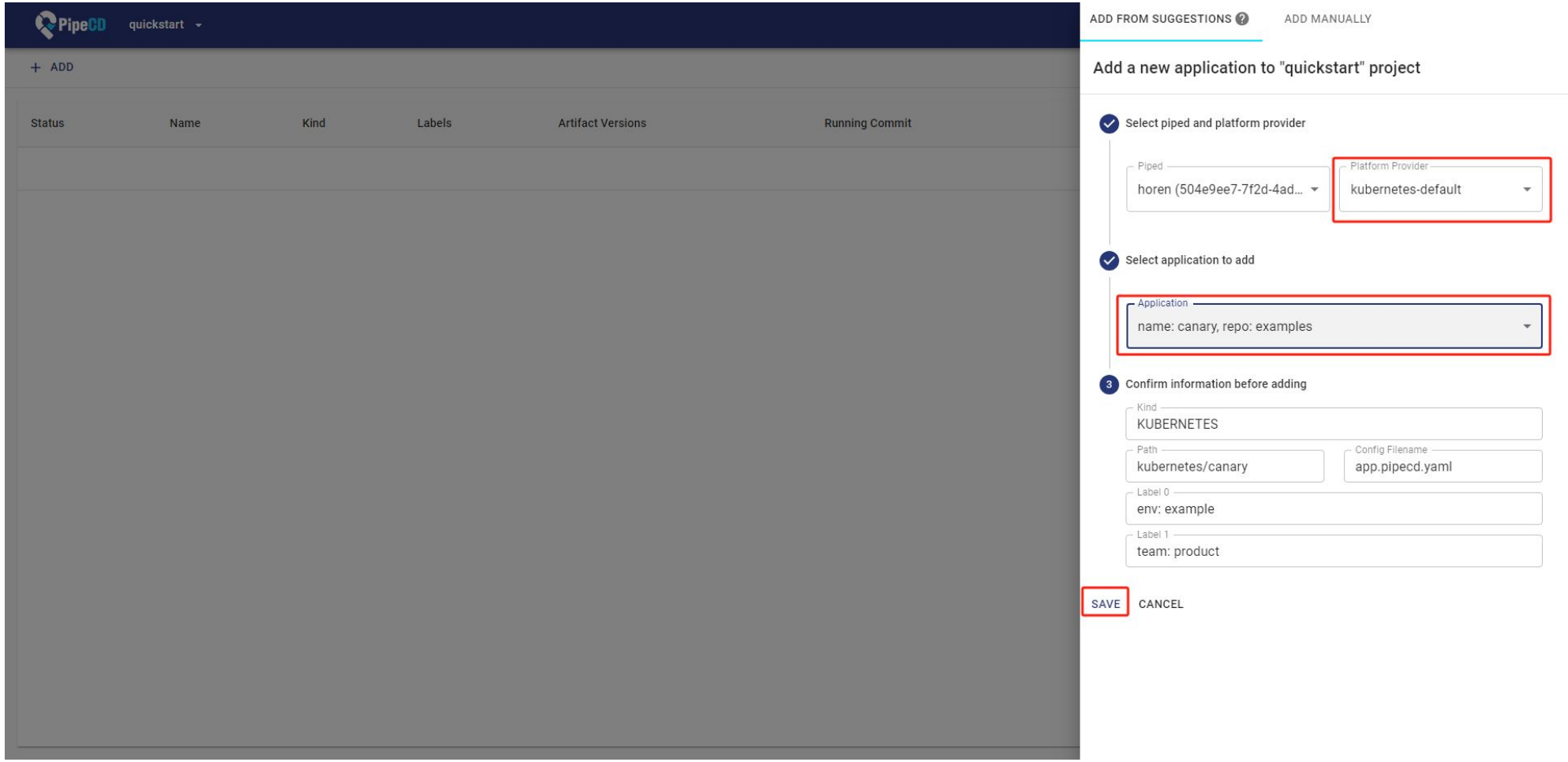
Prepare application as well as PipeCD configuration




```yaml
apiVersion: pipecd.dev/v1beta1
kind: KubernetesApp
spec:
  name: canary
  labels:
    env: example
    team: product
  pipeline:
    stages:
      # Deploy the workloads of CANARY variant. In this case, the number of
      # workload replicas of CANARY variant is 10% of the replicas number of PRIMARY variant.
      - name: K8S_CANARY_ROLLOUT
        with:
          replicas: 10%
      # Wait 10 seconds before going to the next stage.
      - name: WAIT
        with:
          duration: 10s
      # Update the workload of PRIMARY variant to the new version.
      - name: K8S_PRIMARY_ROLLOUT
      # Destroy all workloads of CANARY variant.
      - name: K8S_CANARY_CLEAN
```

Demo of PipeCD

Deploy application to a single cluster



Demo of PipeCD + Karmada

Add Karmada configuration

master | examples / kubernetes / canary /

RainbowMango add Karmada propagation policy

This branch is 1 commit ahead of, 7 commits behind `pipe-cd/examples:master`.

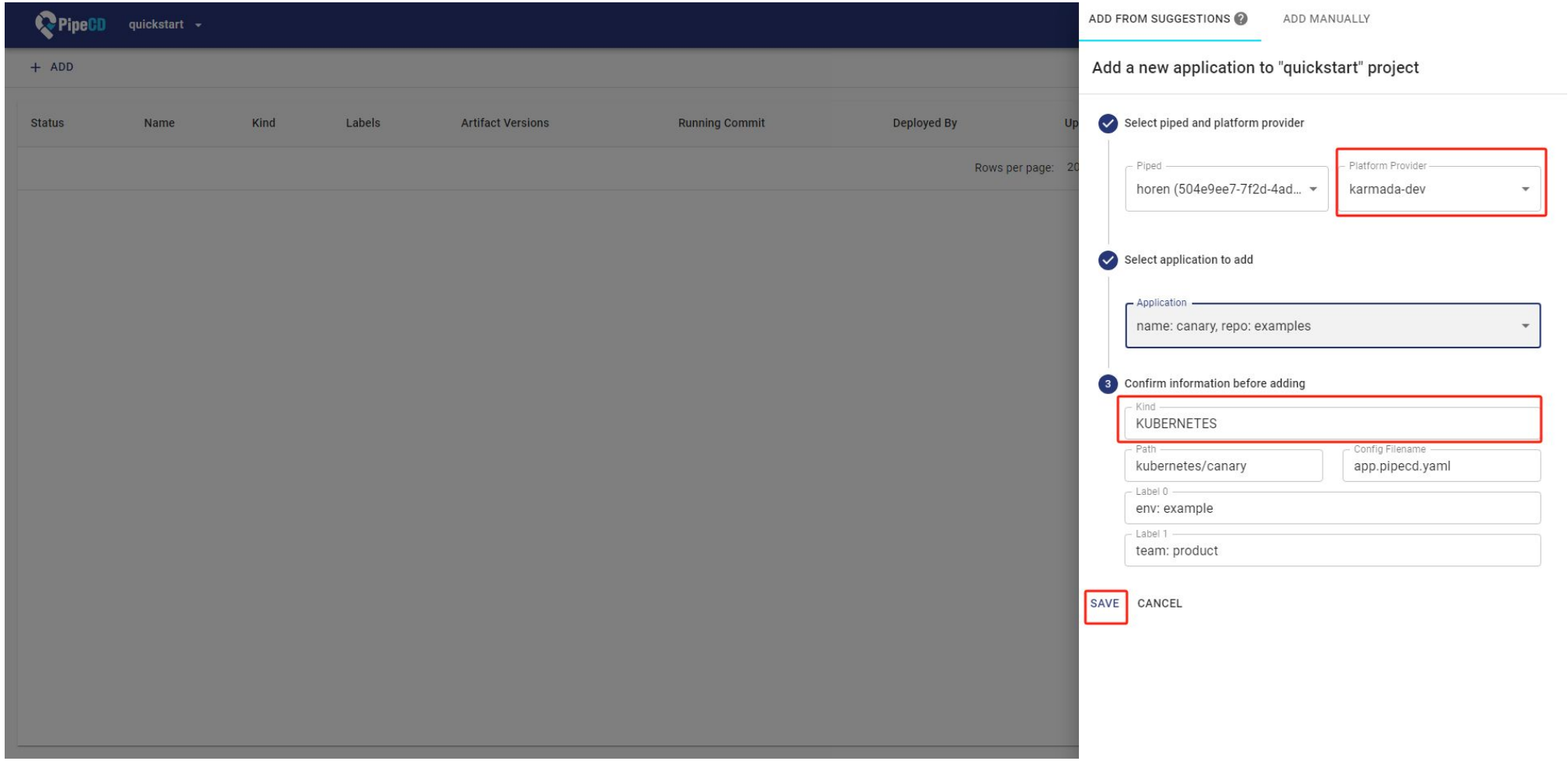
Name

- ..
- app.pipecd.yaml
- deployment.yaml
- propagationpolicy.yaml
- service.yaml

```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: canary
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: canary
    - apiVersion: v1
      kind: Service
      name: canary
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
    replicaScheduling:
      replicaSchedulingType: Divided
      replicaDivisionPreference: Weighted
      weightPreference:
        staticWeightList:
          - targetCluster:
              clusterNames:
                - member1
            weight: 1
          - targetCluster:
              clusterNames:
                - member2
            weight: 1
```

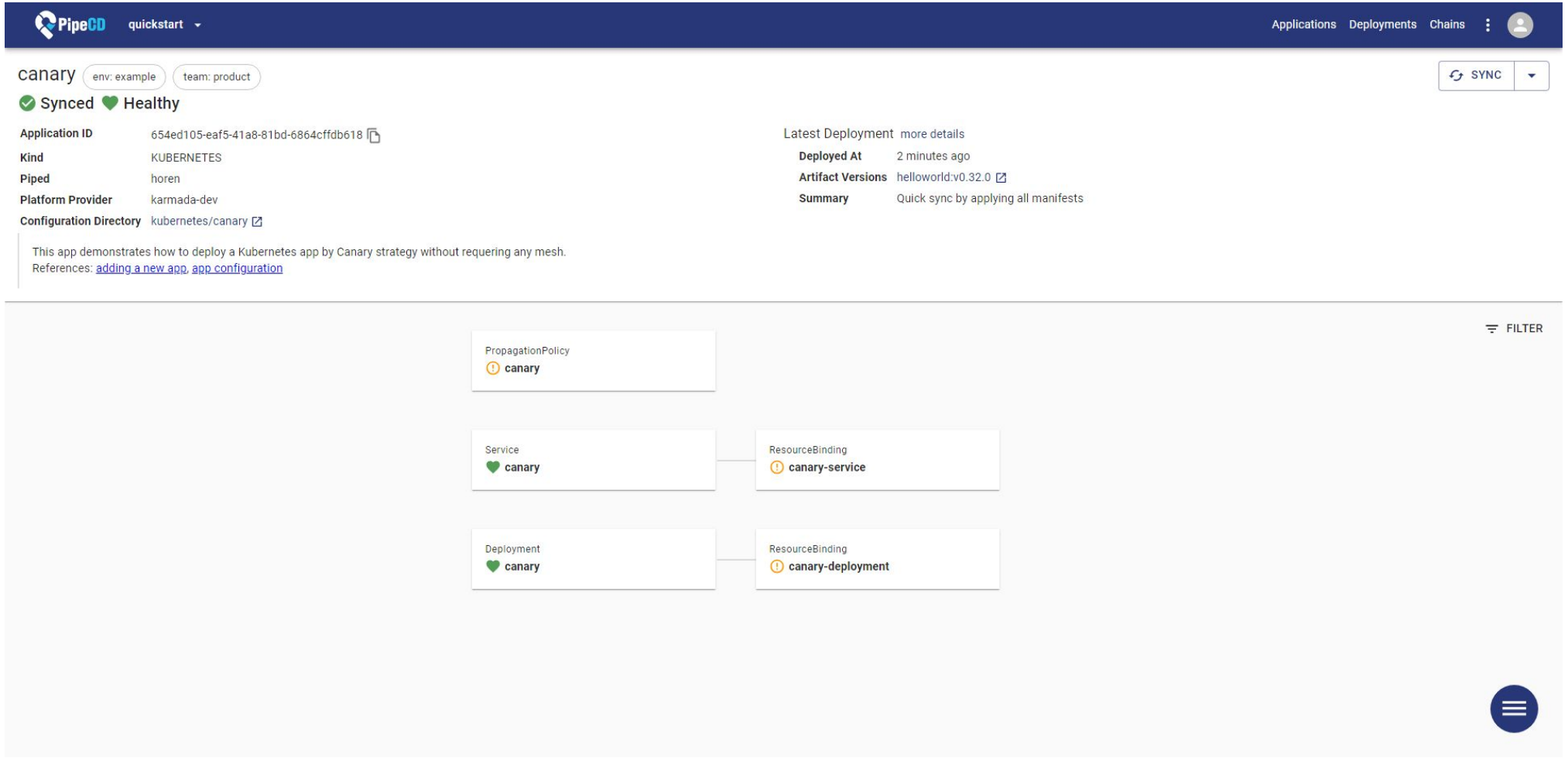
Demo of PipeCD + Karmada

Deploy application to Karmada

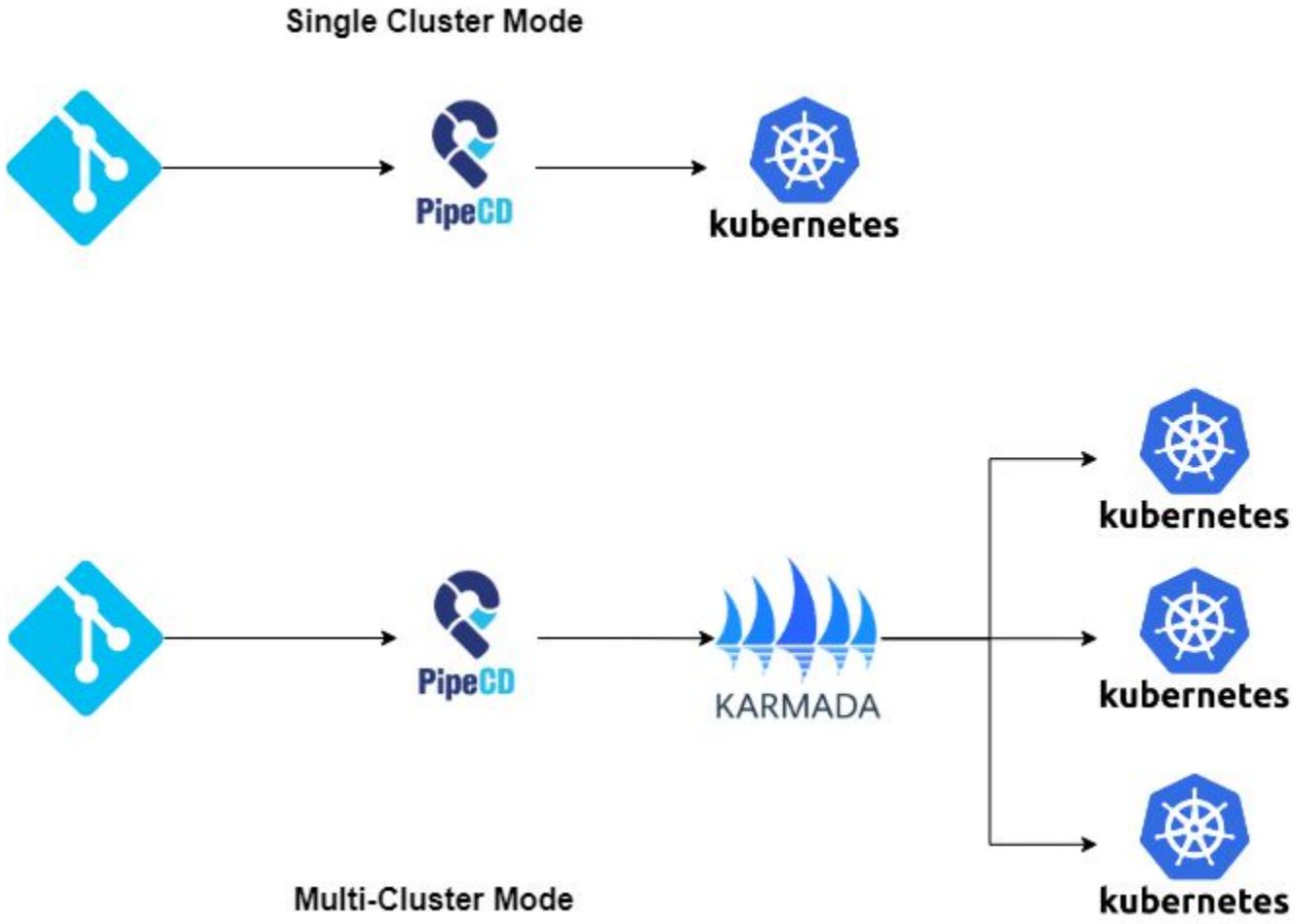


Demo of PipeCD + Karmada

Show status of applications



Demo of PipeCD + Karmada



Take Away

- PipeCD, a GitOps tool that enables deploy application by pull request on Git
  - Easy to manage / operate for both dev & operator
  - Manage multi applications across multiple clusters
  - Same simple but powerful interface for many kinds of applications (not just Kubernetes)

- Karmada, multi-cluster container orchestration platform
  - Speaks Kubernetes API
  - Easy to integrate with Kubernetes ecosystem tool-chain
  - Manage application across multiple clusters

Join Us

https://karmada.io

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

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

https://pipecd.dev

https://github.com/pipe-cd/pipecd

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


KubeCon
CloudNativeCon
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
AI_dev
Open Source GenAI & ML Summit
China 2024
Thanks!