KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA

Sailing Multi Cloud Traffic Management With Karmada

Zhonghu Xu, Huawei

About me

- Istio steering committee member
- Istio Core Maintainer & Contributor
- Open source enthusiastic
- Kubernetes member & core contributor
- Github：https://github.com/hzxuzhonghu

Zhonghu Xu

Open Source Engineer

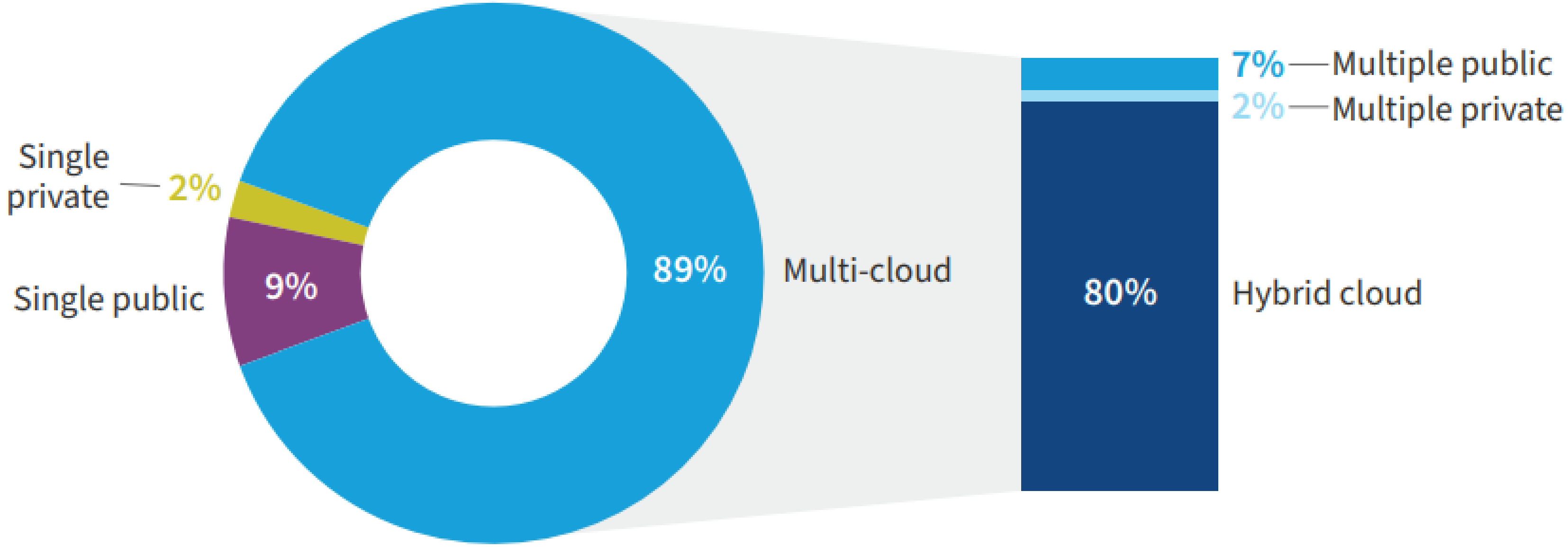
Huawei

Agenda

1. Background
2. Karmada introduction
3. Multi cloud traffic management
4. Conclusion

Background: Multi cloud, multi cluster strategy is widely adopted

Cloud strategy for all organizations



N=753
Source: Flexera 2022 State of the Cloud Report

Flexera

Background: Why Multi Cloud

- Avoid vendor locking
- Meeting compliance requirements
- Enhancing resilience
- Improving flexibility and scalability

Background： Multi Cloud Challenges

- Management complexity
- Security concerns.
- Communication
- Monitoring system concern

Background: Multi-Cloud Kubernetes Challenge

Challenges of managing multiple container clusters

Too Many Clusters

Cumbersome and Repetitive setup

Incompatible Cluster Lifecycle API

Fragmented API endpoints

Workload Fragmentation

Per-cluster customization for Apps

Multi-cluster service discovery for Apps

Sync Apps between clusters

Boundary of Clusters

Resource Scheduling

Application Availability

Horizontal Auto-scaling

Vendor lock-in

Deployment Gravity

Lack of Migration Automation

Lack of independent, neutral, open source multi-cluster management projects

Karmada: Open, Cloud-Native, Multi-Cloud Orchestration Engine

Easily build infinitely scalable cluster pools with Karmada

Use multi-cloud clusters just like a single K8s cluster

K8s Native API Compatible

Zero change upgrade: single-cluster ➔ multi-cluster
Seamless integration of existing K8s tool chain

Open and Neutral

Jointly initiated by Internet, finance, manufacturing, teleco, cloud providers, etc.
Target for open governance with CNCF

Avoid Vendor Lock-in

Integration with mainstream cloud providers
Automatic allocation, migration across clusters
Not tied to proprietary vendor orchestration

Out of the Box

Built-in policy sets for scenarios:
GDPS, Active-active, Remote Disaster Recovery

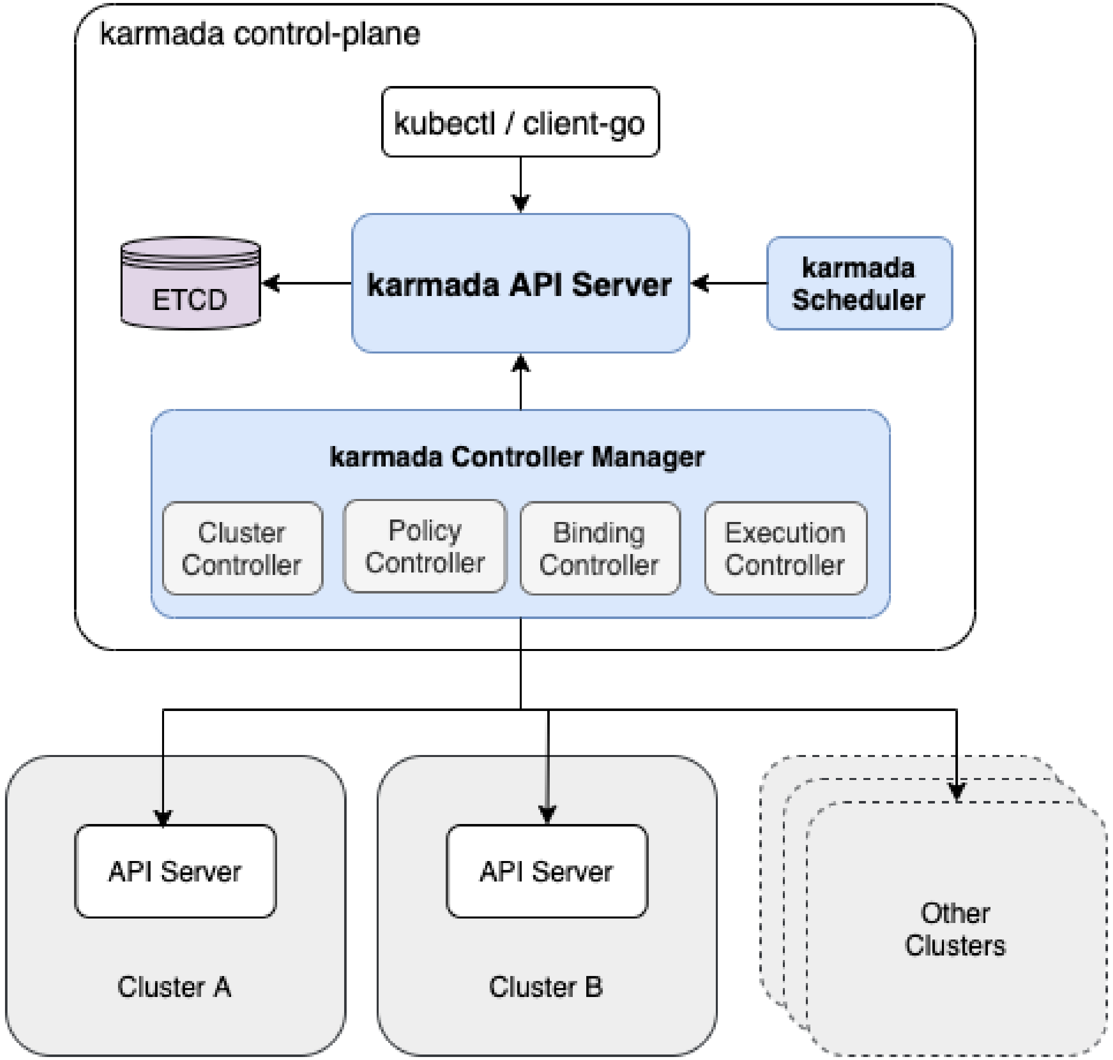
Fruitful Multi-Cluster Scheduling Policies

Cluster Affinity, Multi Cluster Splitting/Rebalancing,
Multi-Dimension HA: Region/AZ/Cluster/Provider

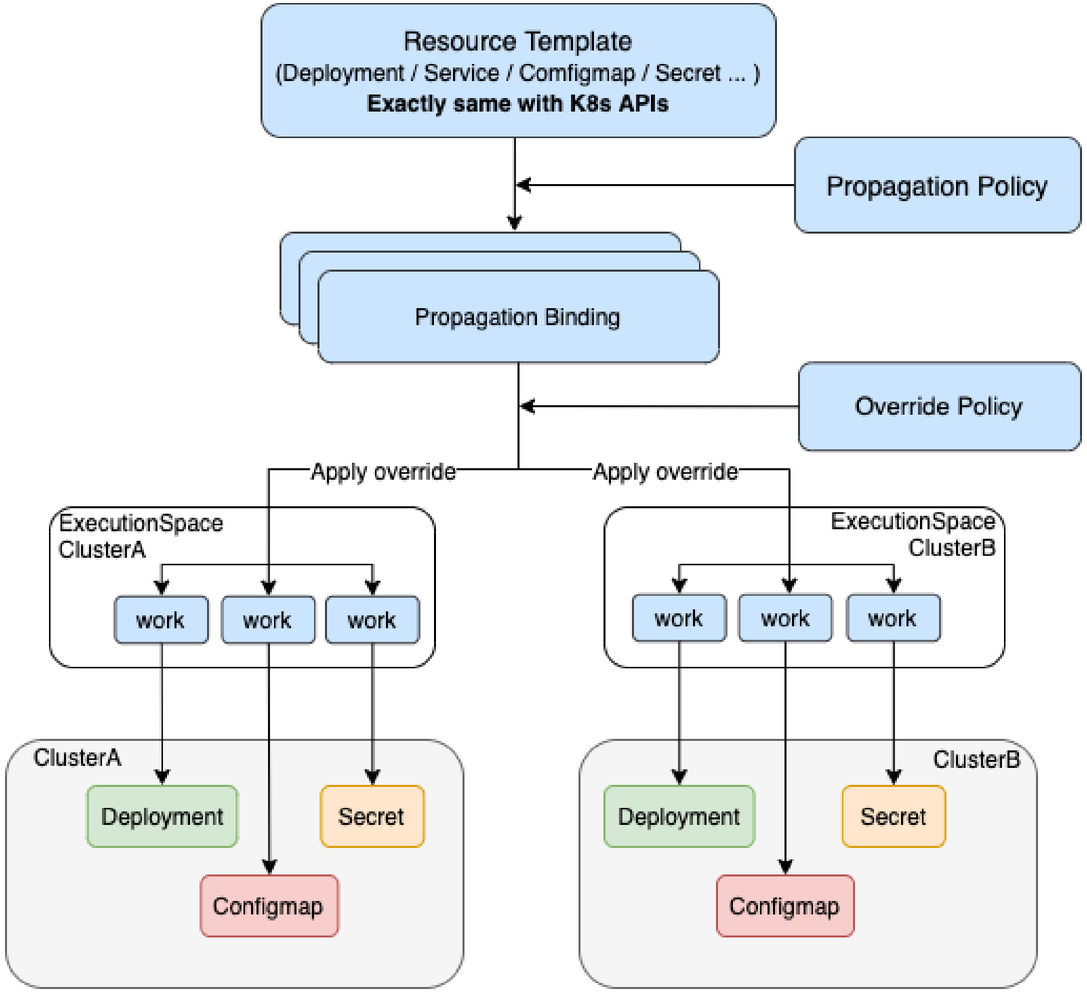
Centralized Management

Cluster location agnostic
Support clusters in Public cloud, on-prem or edge

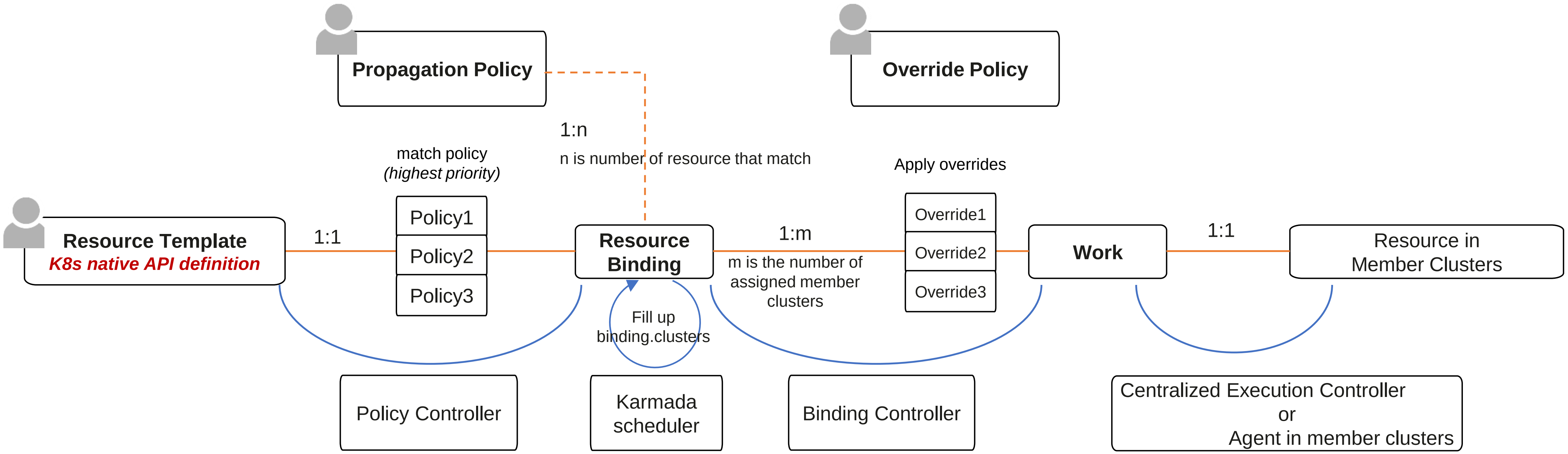
Karmada Architecture



Karmada Resources



Karmada API Workflow



Zero Change: Running Multi-Cluster Application with vanilla K8s API

Reusable propagation policy

```
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: 3
        minGroups: 3
```

Example: spread all deployments (that has label

ha-mode: multi-zone-replication) to 3 zones

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    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
```

Exactly vanilla Kubernetes API

Use exactly vanilla K8s API to deploy multi-cluster application

kubectl create -f nginx-deployment.yaml

Propagation Policy: multi-cluster scheduling policy

```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: example-policy
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: deployment-1
      labelSelector:          # standard labelSelector
  propagateDependensies: false
  placement:
    clusterAffinity:
      clusterNames:
        - cluster1
        - cluster3
    clusterTolerations:       # like pod tolerations
    spreadConstraints:
      - spreadByField: zone
        maximum: 3
        minimum: 3
  schedulerName: default
```

resourceSelector

- Match resources that the propagation policy apply to.
- *apiVersion + kind* for basic filtering
- *name* for exact match
- *labelSelector* for advanced matching

placement

- Represents preferences of propagating resources
- *clusterAffinity*:
  - Preferred clusters to go
  - Exact match by names or match by labelselector
- *clusterTolerations*:
  - similar idea to pod tolerations and node taints, member clusters have taints to mark reservation for special usage,
    and only federated resources spread by propagation policies with corresponding tolerations can go there.
- *spreadConstraints*:
  - constraints of spreading federated resources among member clusters.
  - Users can specify dynamic grouping clusters by labels or by fields,
    and maximum/minimum groups they want to use.

Override Policy: resource customization among clusters

```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: OverridePolicy
metadata:
  name: example-override
  namespace: default
spec:
  # restrict resource types that this override policy applies to
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name:          # user can either select resource by name or by labelselector
      labelSelector:
        matchLabels:
          image: nginx
  # this override policy will only apply to resources propagated to the matching clusters
  targetCluster:
    clusterNames:      # user can either select cluster by names or by labelselector
      - dc-1-cluster-1
      - dc-1-cluster-2
    labelSelector:
      matchLabels:
        failuredomain.kubernetes.io/region: dc1
  # all matching targetClusters would share the same set of overrides below
  overriders:
    plaintext:
    - path: "/spec/template/spec/containers/0/image"
      value: "dc-1.registry.io/nginx:1.17.0-alpine"
    - path: "/metadata/annotations"
      op: "add"
      value:
        foo: bar
    - path: "/metadata/annotations/foo"
      op: "remove"
```

Example usage of override policy API

There are 3 clusters:

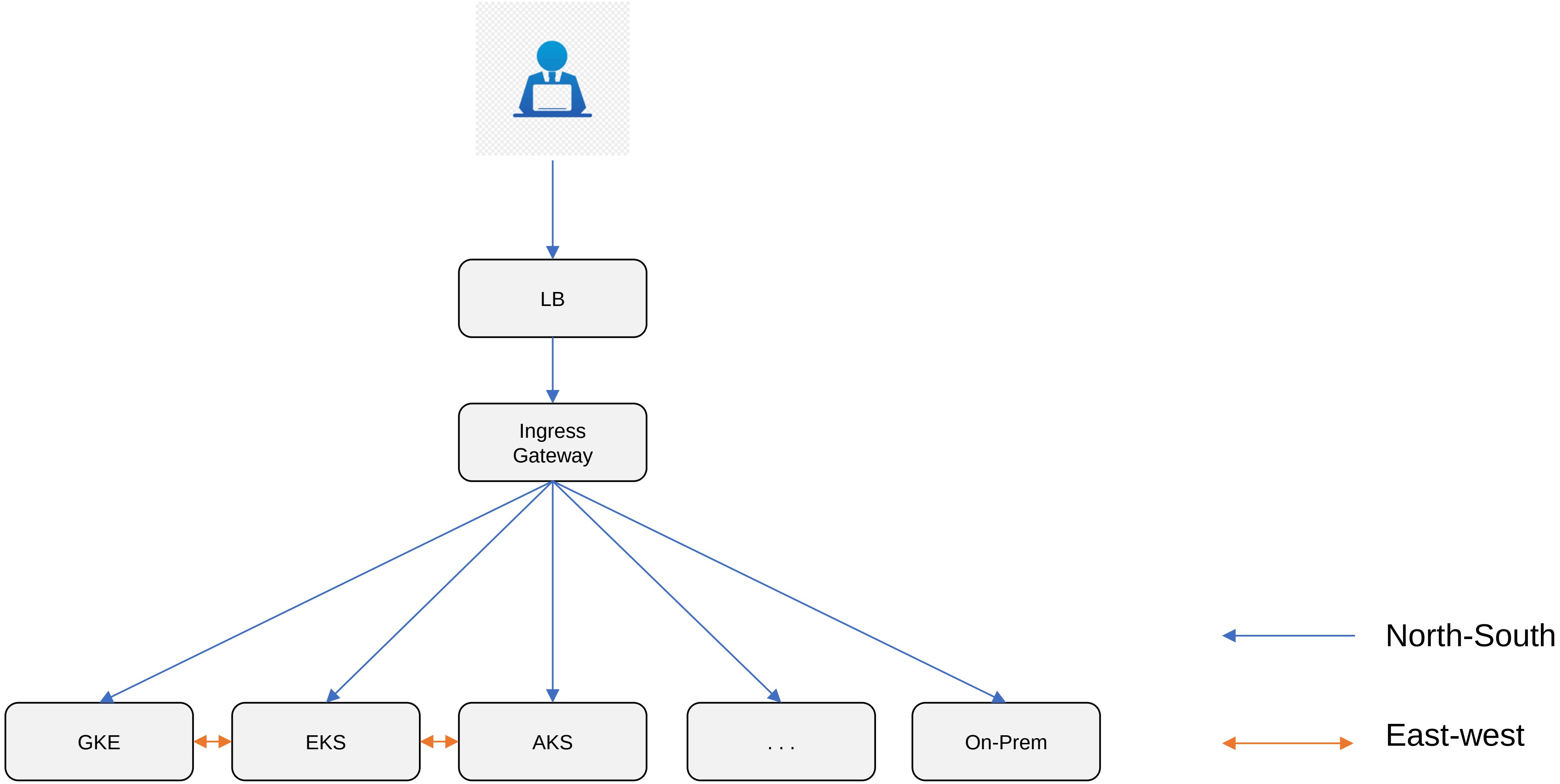
- *cluster-1* and *cluster-2* locate in *dc-1* which is an on-prem environment
- *cluster-3* is a managed Kubernetes cluster on public cloud.

To save image downloading bandwidth, latency:

- For deployments in *cluster-1* and *cluster-2*, download image from local registry *dc-1.registry.io*
- For deployments in *cluster-3*, download image from the registry managed by cloud provider

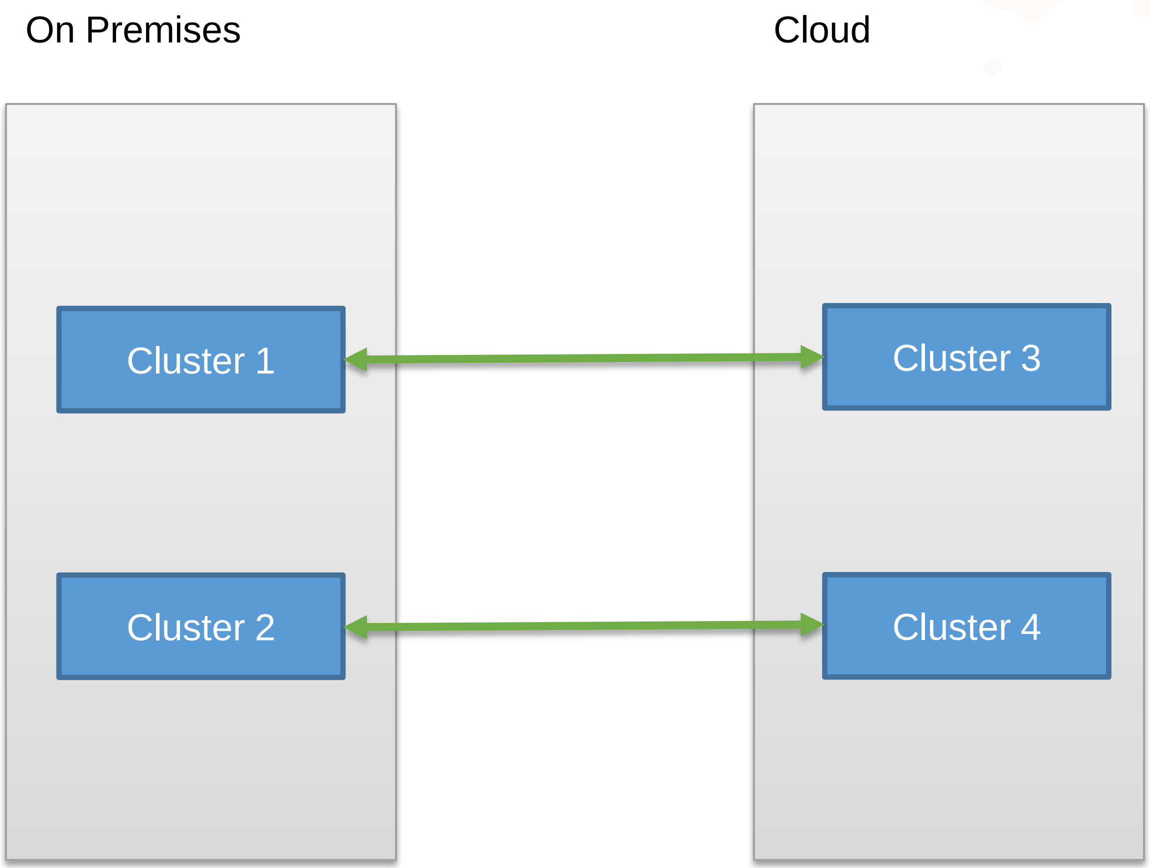
Override rules for *cluster-1* and *cluster-2* shown as in the left

Multi cloud Traffic Management



Multi cloud Traffic Management Challenges

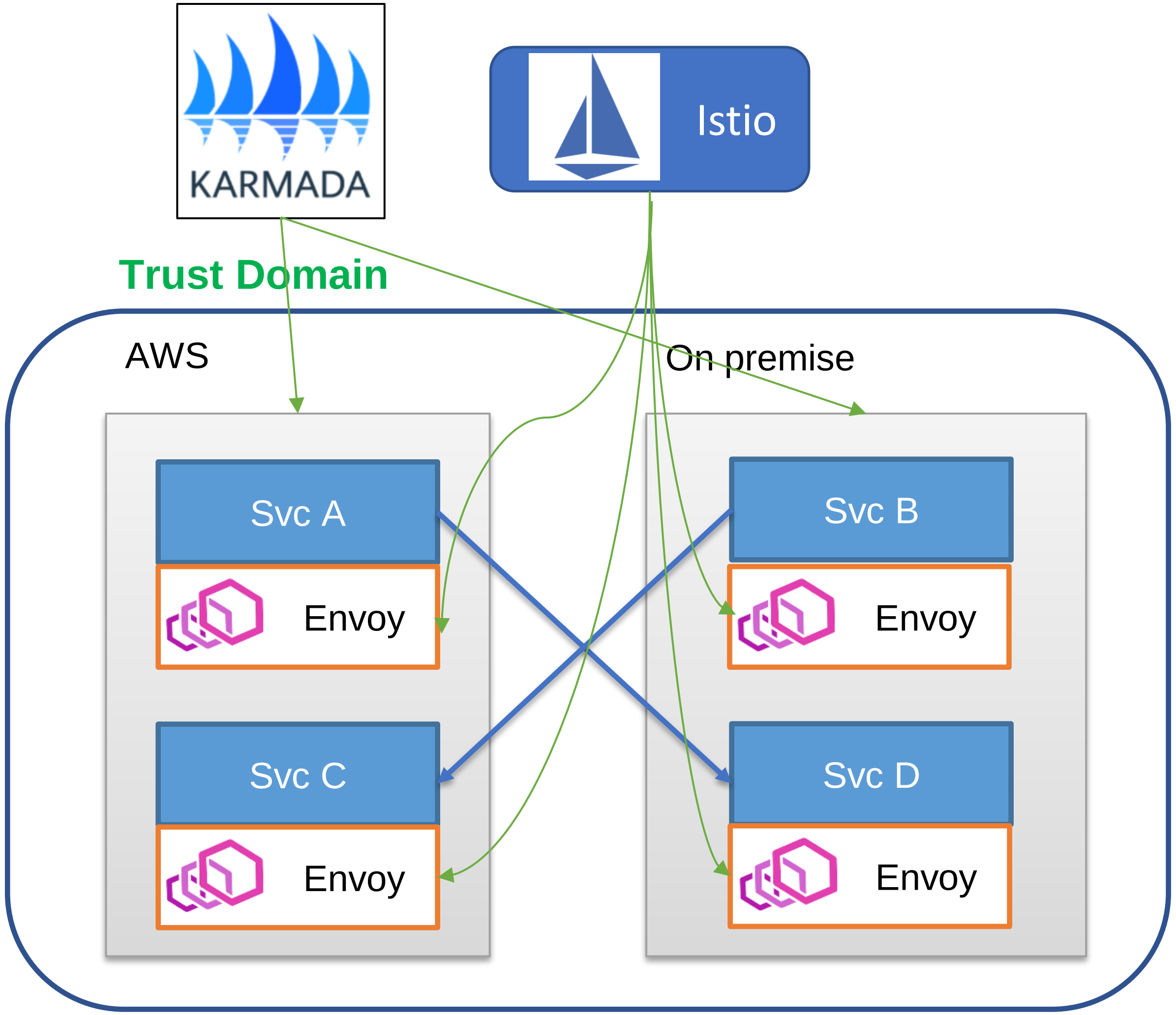
- Network reachability
- Service discovery
- DNS resolve
- Load balancer Policy
- Security for cross cluster traffic



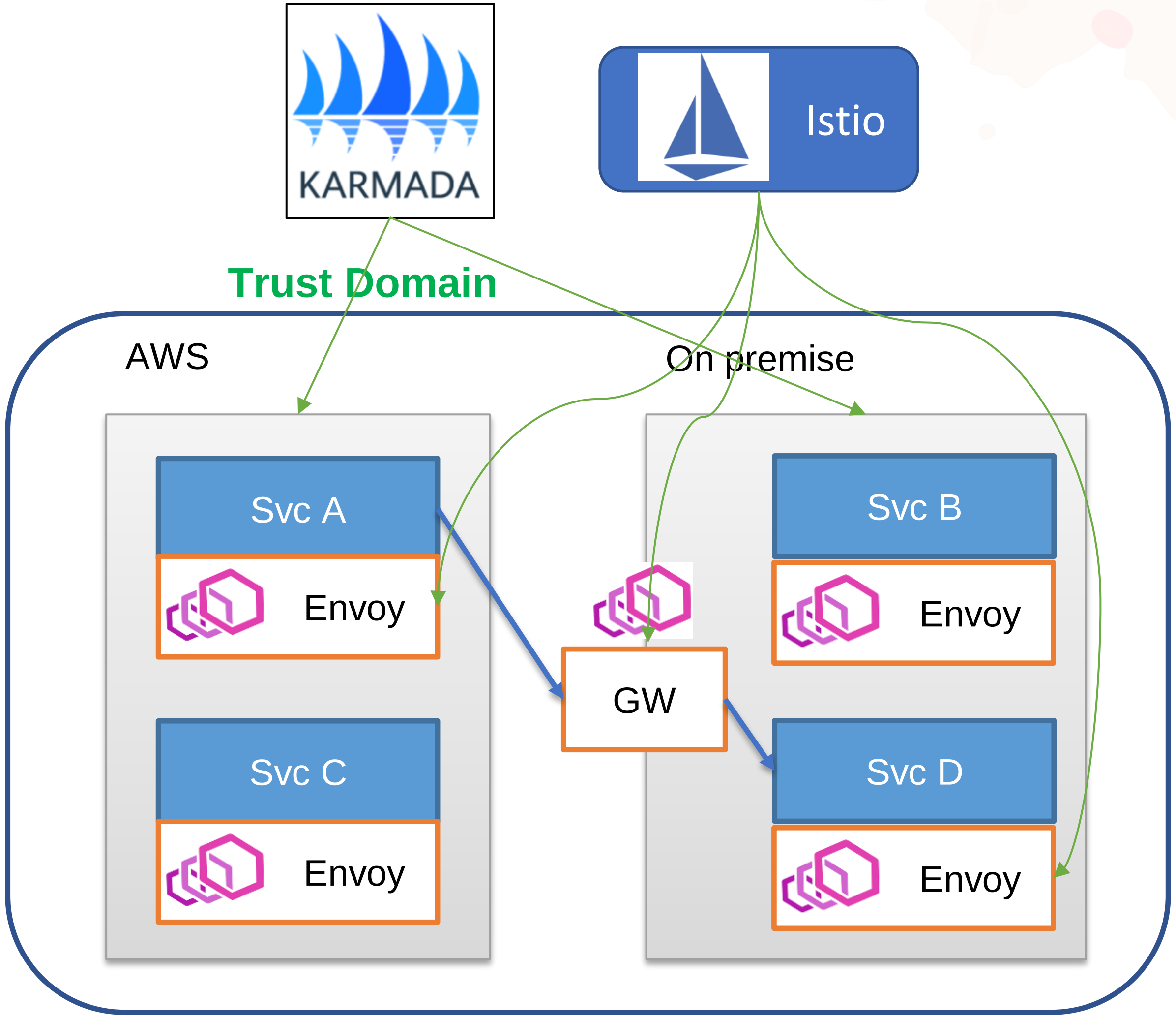
Multi cloud Traffic Management: Karmada Way

- Submariner to build connect overlay networks of different cloud clusters
- Export and import services between clusters with Multi-cluster Service APIs.
- Integrate with mature service mesh Istio

Multi cloud Traffic Management: Karmada + Istio

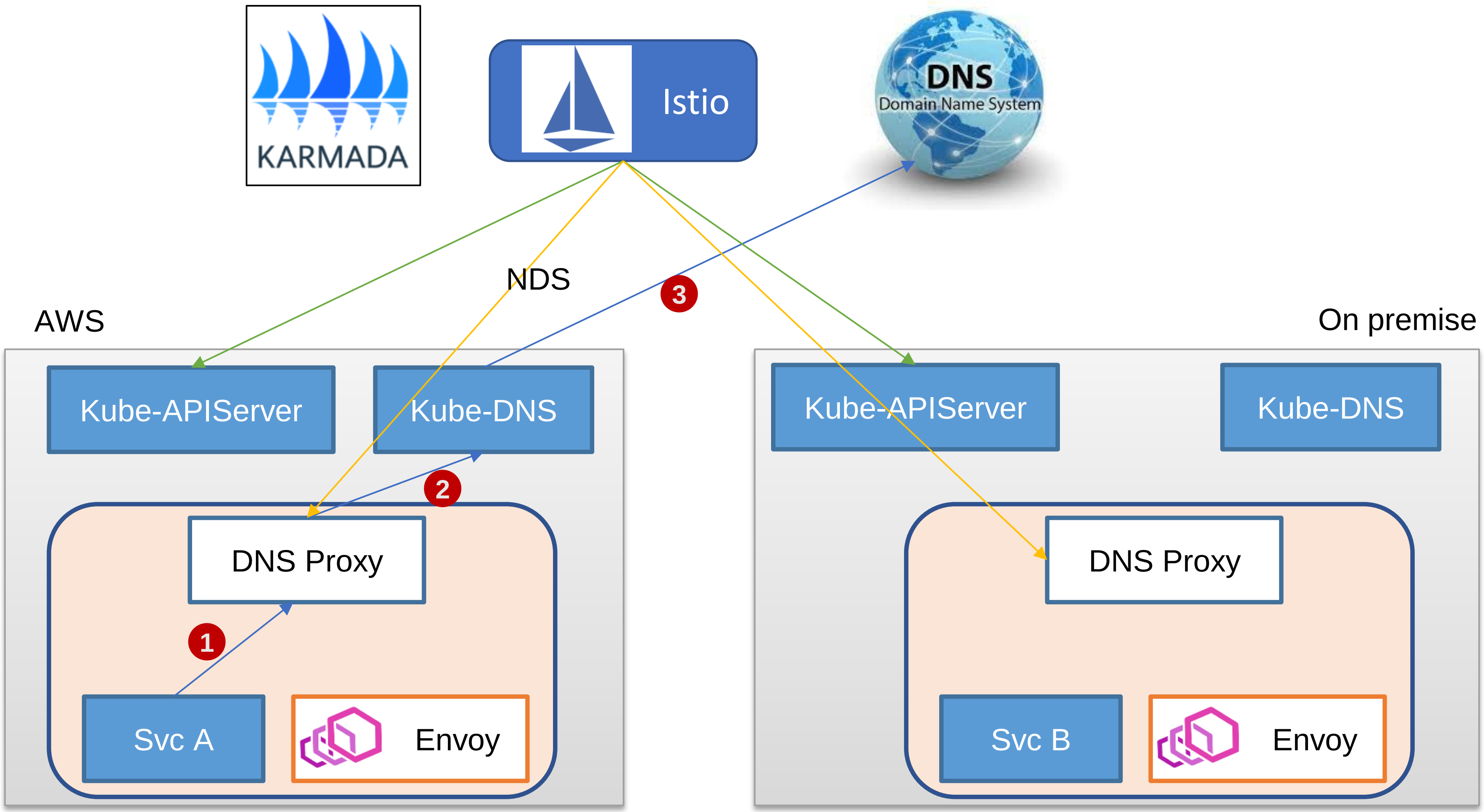


Flat network



Non flat network

Multi cloud Traffic Management: DNS Resolution



Multi cloud Traffic Management: Flat network

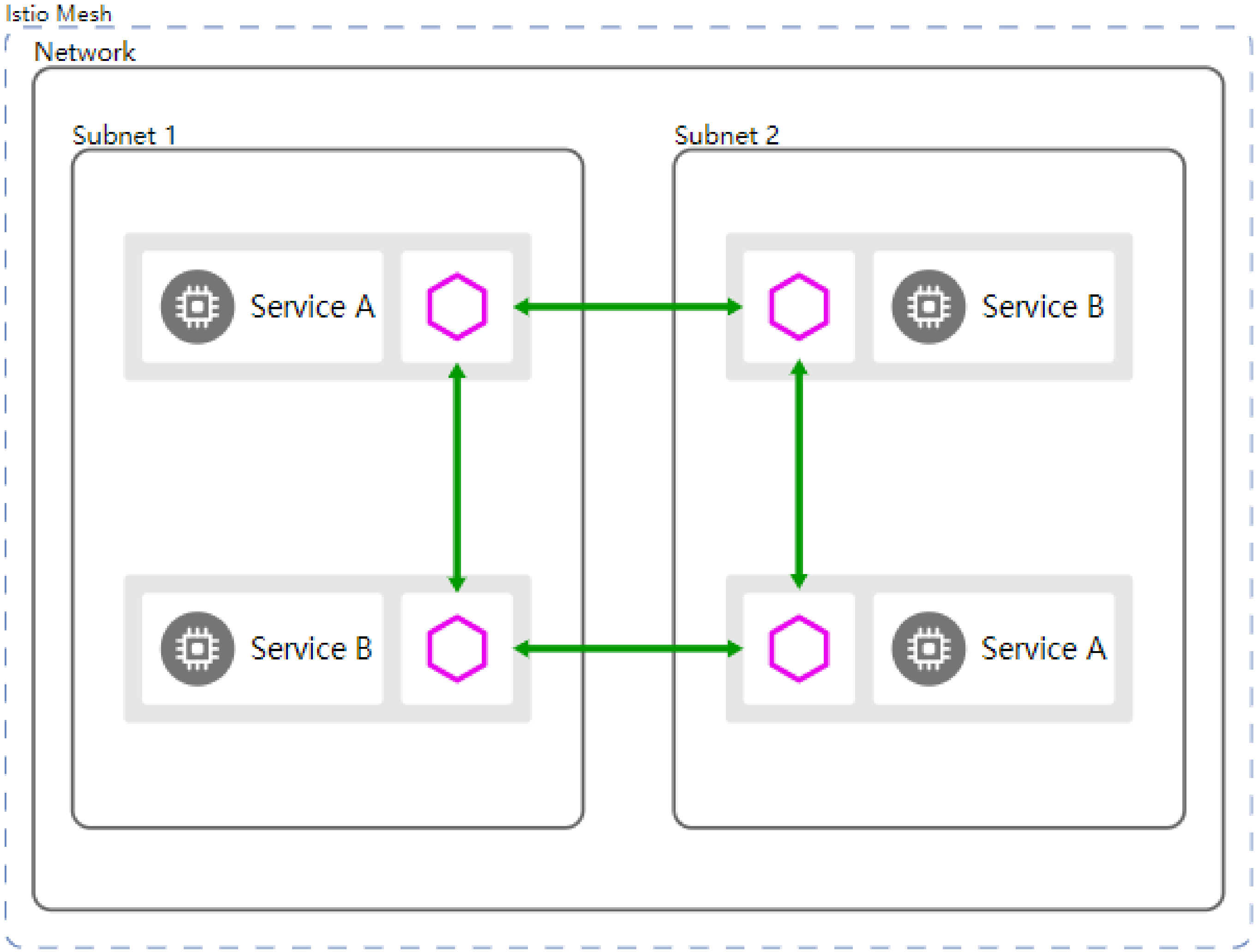
Flat Network

Pros

① Low latency east-west traffic, as gateway is not needed
② Inherit all capabilities from single cluster

Cons

① Complexity: need additional tool to build flat network
② Security: not secure as all workloads are within a single network.
③ No overlapping service ip ranges



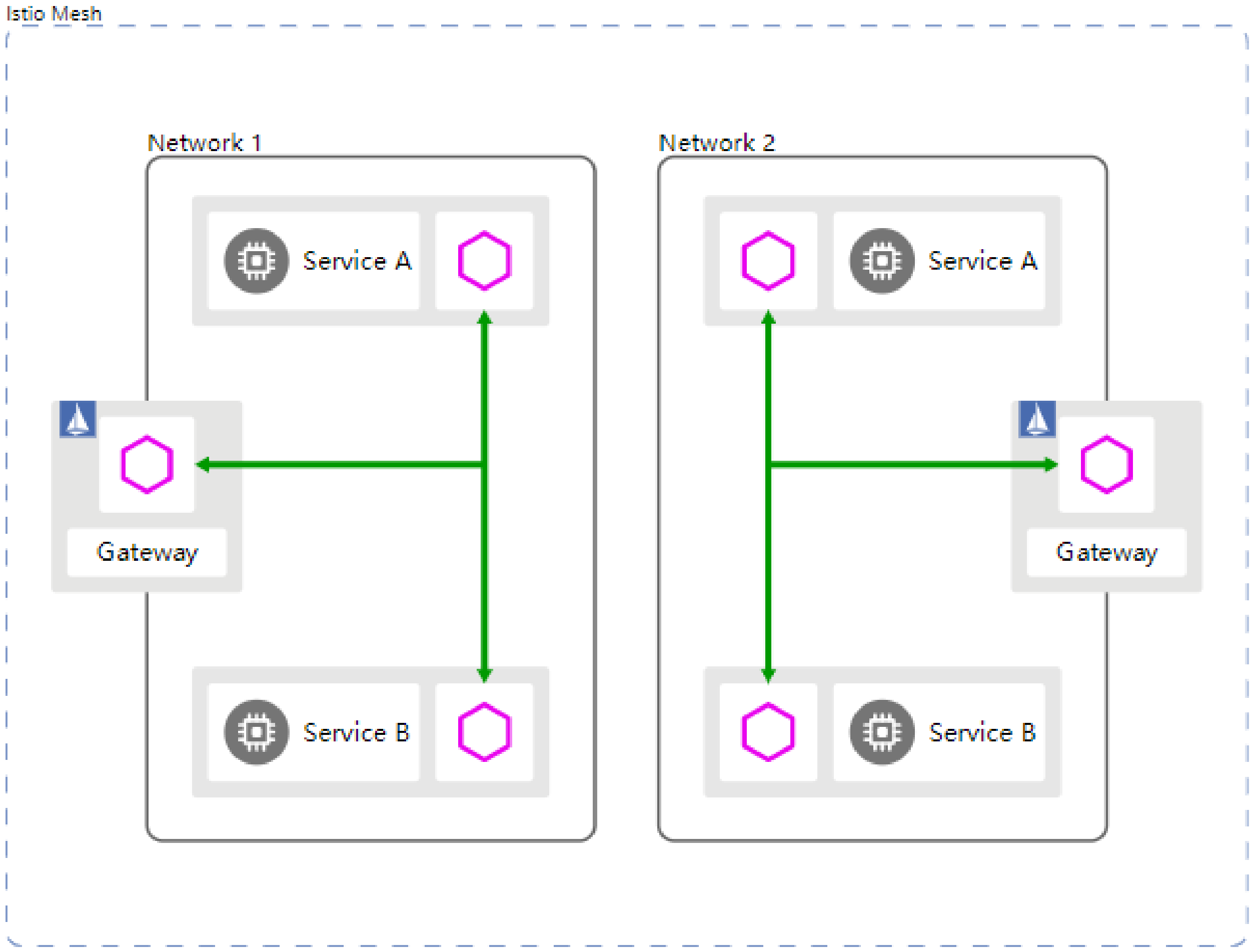
Multi cloud Traffic Management: Different networks

Pros

① Isolation: network segmentation
② Security: east-west traffic is encrypted and through gateway
③ Scaling of network addresses
④ Cost saving

Cons

① Cross network service communication requires east-west gateway
② Gateway works in TLS AUTO_PASSTHROUGH mode, lack of routing capabilities as flat network
③ Additional hops.



Conclusion

- Multi cloud evolution and what’s challenged it bring about
- What Karmada can do for multi cloud
- Inter-Cloud communication with Istio

QA