Red Hat

OpenShift ACM

Open Cluster Manager

TRAINING

CHOI GOOK HYUN
Freelancer Instructor

DAY 1

- About the OpenShift Advanced Cluster

Red Hat

THE FEDERATION

THE FEDERATION

with kubernetes

"The Federation" API는 올해 2022년도 10월에 더 이상 개발을 하지 않기로 결정하였음. 다만, API사양은 계속 개발하며, 프로그램 부분은 OCM, KARMADA와 같은 커뮤니티에 개발을 맡기기로 결정.

공식적으로 Federation API는 v1, v2기반으로 되어 있으며, 쿠버네티스는 API정의 애플리케이션 개발은 외부 그룹에서 맡기로 하였음.

현재는 다음과 같이 두 그룹이 주요 리더로 끌고 가고 있음.

- Open Cluster Manager
- Karmada

THE FEDERATION

with kubernetes

Follow-up: discussion on archiving Kubefed 조회수 177회

pmo...@gmail.com

받는사람 kubernetes-sig-multicluster

Everyone-

In recent Kubecon SIG updates we have previously communicated that SIG leadership has been considering archiving the Kubefed[1] repository. This topic was given dis discuss offline. This communication is a follow-up on that action item.

TLDR: Our current favored option is to archive the kubefed project but we are soliciting community input before making a final decision. Continue reading for rationale.

In our discussions, Jeremy and I considered the following questions:

KARMADA

KARMADA

with kubernetes

KARMADA는 쿠버네티스 federation v1, v2프로토콜 기반으로 작성이 되어 있음. 쿠버네티스 기반으로 동작하며, 동작 방식은 OCM과 매우 흡사하나, 중간에 "Karmada API Server"중심으로 "etcd", "scheduler"와 통신하며 중간에 "Karmada controller"를 통해서 정책 배포 및 관리를 한다.

현재 KARMADA는 "CLOUD NATIVE"의 샌드박스(Sandbox) 프로젝트로 선언이 되어 있다.
CNCF에서는 좀 더 KARMADA를 지원하고 있는 분위기.

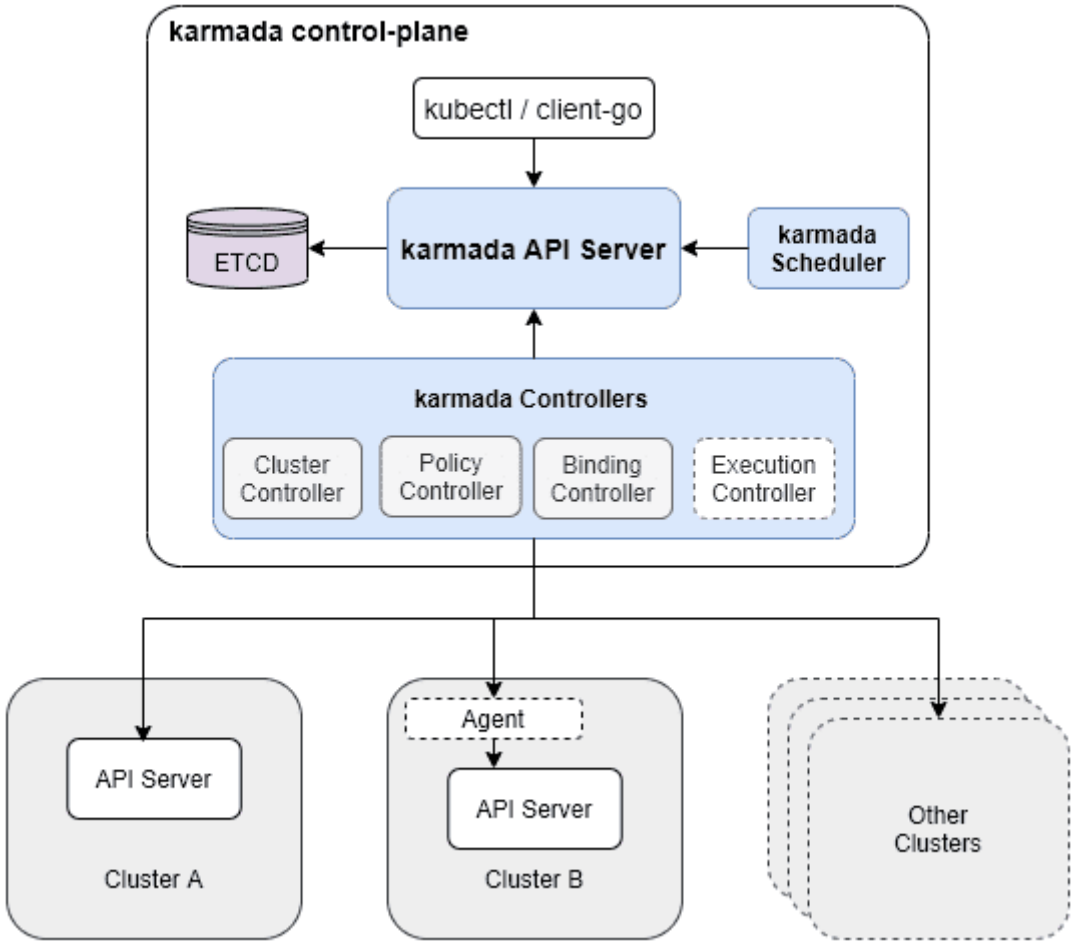
KARMADA

with kubernetes

주요 목적은 다음과 같다.

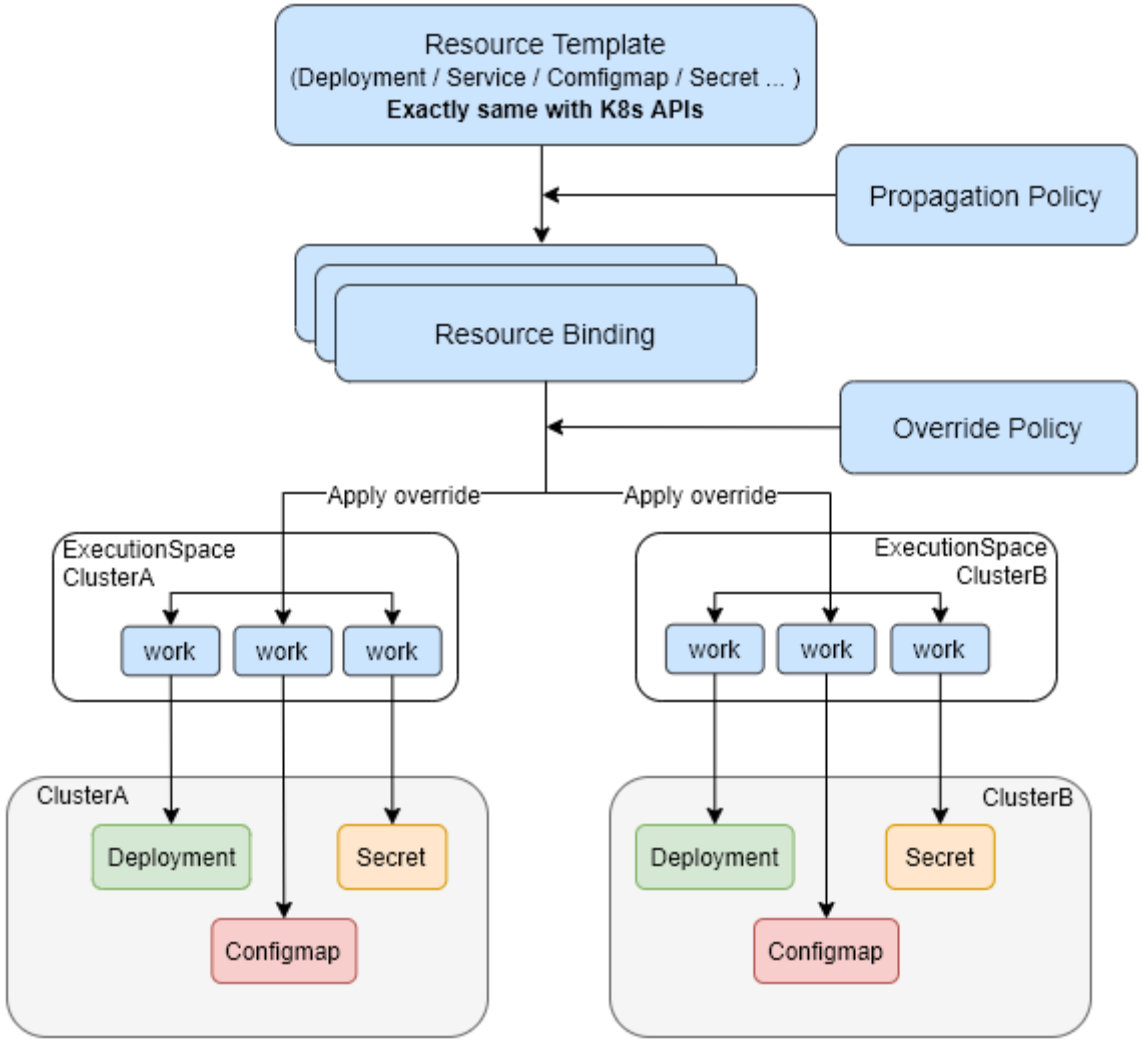
- KIS native API compatible
- Out of the Box
- Avoid Vendor Lock-in
- Centralized Management
- Fruitful Multi-Cluster Scheduling Policiese
- Open and Neutral

KARMADA



Red Hat

Karmada Concepts



Red Hat

OCM

OCM

with OpenShift

OpenShift는 Kubernetes기반으로 작성이 되어 있음. API는 두 가지 API로 구별이 됨.

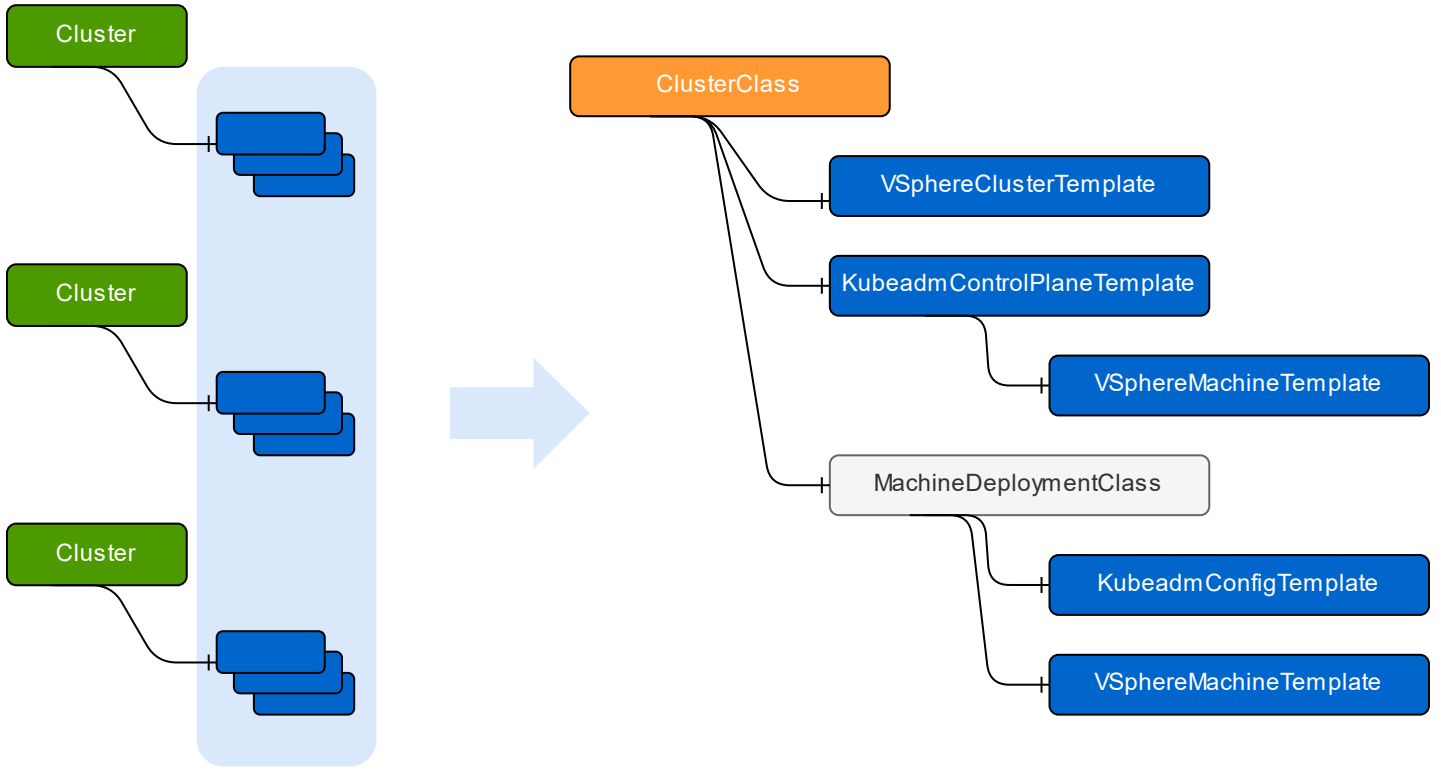
- Cluster API
- Federation API

Cluster API는 Kubernetes SIG에서 맨 처음 릴리즈 한 API. Federation API는 다중 클러스터 운영을 위해서 SIG에서 개발을 시작. OCM에는 현재 레드햇이 개발을 참여하고 있으며, 이 프로젝트도 "CLOUD NATIVE COMPUTING FOUNDATION"의 일부분.

프로젝트 주소

https://github.com/stolostron

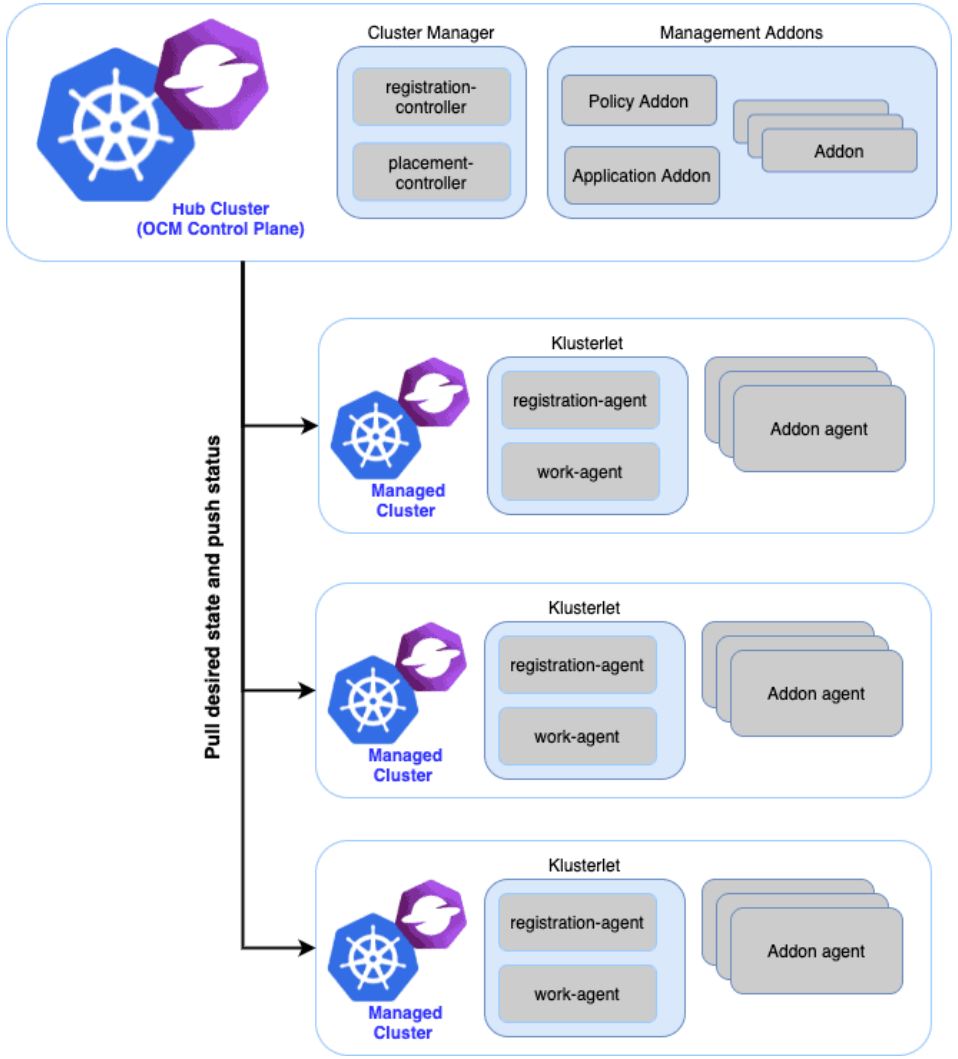
Federation



Viewer does not support full SVG 1.1

Red Hat

OCM Arch



Red Hat

OCM

MultiCluster API

클러스터 API는 다음과 같은 기능을 기본적으로 다루고 있다.

- 자원을 라이프 사이클(LifeCycle) 관리
- 설정 및 정책(Deployment, Policy) 적용 및 배포
- Pod 및 컨테이너 복제
- 네임스페이스 관리

Red Hat

OCM

Federation V1

Fede v1는 SIG에서 멀티 클러스터를 관리하기 위해서 프로젝트를 시작하였다. 쿠버네티스 SIG에서 제일 큰 프로젝트 중 하나 이었으며, 앞서 말한 것처럼 멀티 클러스터 관리를 위해서 v1를 설계를 하였다. 주요 목적은 다음과 같다.

- Difficulties in re-implementing the Kubernetes API at the cluster level, as federation-specific extensions were stored in annotations.
- Limited flexibility in federated types, placement and reconciliation, due to 1:1 emulation of the Kubernetes API.
- No settled path to GA, and general confusion on API maturity; for example, Deployments are GA in Kubernetes but not even Beta in Federation v1.

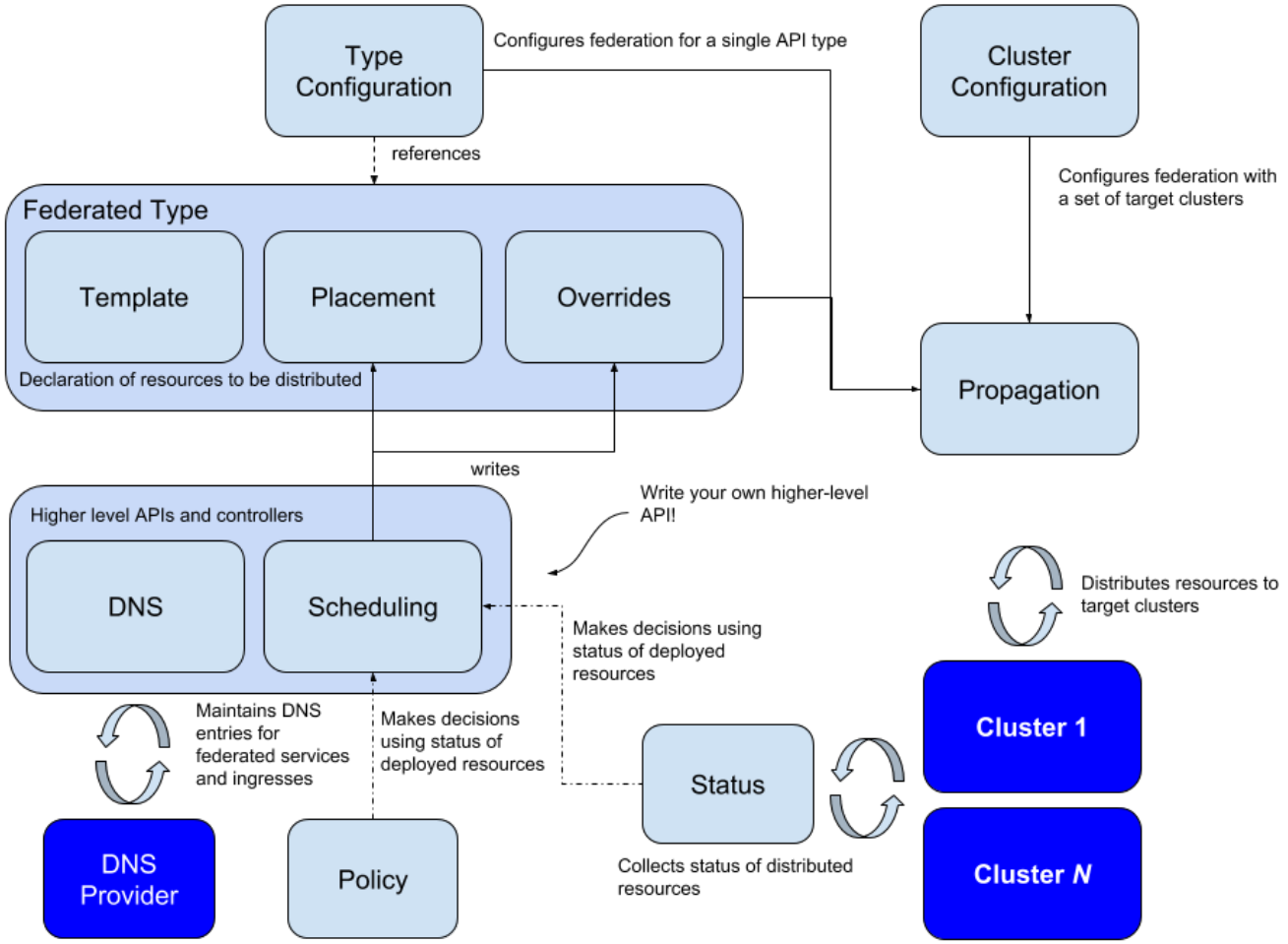
OCM

Federation V1

Federation은 v1기반으로 v2로 진화가 되었으며, 기본적인 컨셉은 v1에서 가져와서 v2에서 기능 추가를 시작하였다.

Federation은 "Hub and Spoke(pull and push)"구조를 가지고 있고 다음과 같이 동작한다.

Native Federation Diagram



Red Hat

OCM

Federation

연합의 주요 목표는 쿠버네티스 자원을 연합 그리고, "API to API"를 그룹으로 정의 하는게 주요 목표이자 목적이다.

쿠버네티스의 기능을 확장하기 위해서 **CRD(CustomResourceDefinitions)**를 통해서 구현이 된다. Federation API를 통해서 요청이 들어오면, 단일 클러스터 API를 배포하는 메커니즘이다.

Kubernetes

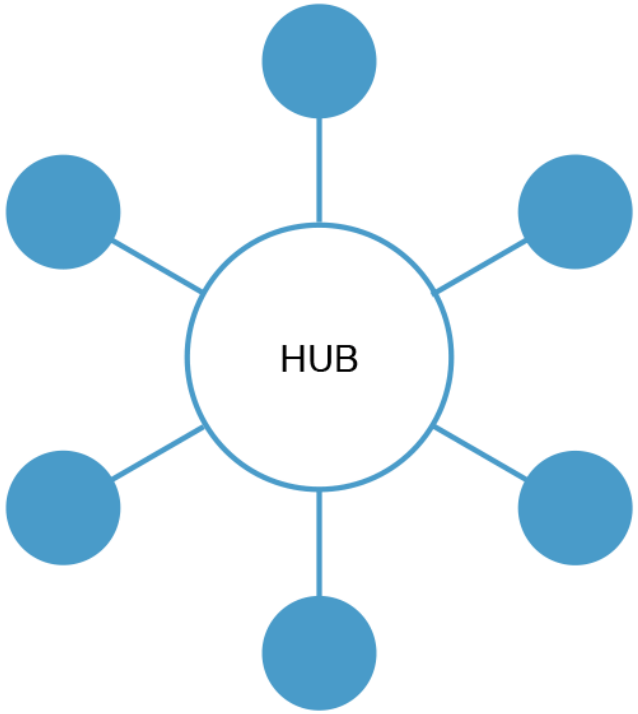
Federation

During the journey of defining building blocks of the federation APIs, one of the near term goals also evolved as 'to be able to create a simple federation a.k.a. simple propagation of any Kubernetes resource or a CRD, writing almost zero code'. What ensued further was a core API group defining the building blocks as a Template resource, a Placement resource and an Override resource per given Kubernetes resource, a TypeConfig to specify sync or no sync for the given resource and associated controller(s) to carry out the sync.

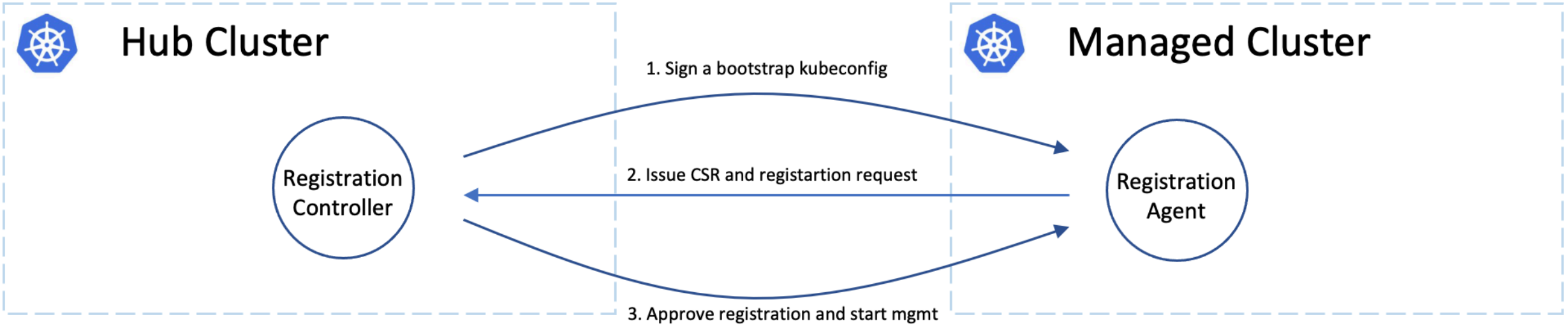
간단한 요약

CRD기반으로 별도의 코드 작성이 필요 없이 CORE API에게 정의하여 자원을 배포하는 형식. 이때 템플릿 자원들로 빌드 블록을 만들어서 Placement자원을 각각 쿠버네티스 자원에 오버라이드 하여 동기화 혹은 비동기화 여부를 컨트롤러를 통해서 결정

HUB and SPOKE



HUB and SPOKE



Template

Federation

Federation에서 주요 자원 지시자는 다음과 같다.

- Template
  - A Template type holds the base specification of the resource - for example, a type called **FederatedReplicaSet** holds the base specification of a **ReplicaSet** that should be distributed to the targeted clusters

Placement

Federation

- Placement
  - A Placement type holds the specification of the clusters the resource should be distributed to - for example, a type called **FederatedReplicaSetPlacement** holds information about which clusters **FederatedReplicaSets** should be distributed to

Override

Federation

- Override
  - An optional Overrides type holds the specification of how the Template resource should be varied in some clusters - for example, a type called **FederatedReplicaSetOverrides** holds information about how a **FederatedReplicaSet** should be varied in certain clusters.

Red Hat

TypeConfig

Federation

- TypeConfig
  - specify sync or no sync for the given resource and associated controller(s) to carry out the sync.

Red Hat

Federation v2

Higher-level behaviour

API v2에서 핵심 기능은 API에서 "템플릿", "배치" 및 "오버라이드"를 통해서 컨트롤러(Controller)통해서 상위 API를 재구성(CRD) 합니다.

클러스터를 가로지르면서 자원을 생성 시 "ReplicaSchedulingPreference"를 통해서 Deployment 및 ReplicaSet를 통해서 연합 클러스터에 자원을 생성 및 구성이 가능하다. 역시 기존 클러스터에서 사용하였던 기능 "limits", "quote"같은 자원 관리도 API를 통해서 배포가 가능하다.

Red Hat

OpenShift vs OCM

Components

오픈 시프트의 ACM은 쿠버네티스의 Federation기반으로 되어 있으며, 레드햇은 OCM기반으로 ACM를 개발하기 시작 하였다. 레드햇 ACM은 다음과 같은 부분에 OCM과 다르다.

- 손쉬운 설치 및 오픈 시프트와 통합된 환경
- 클라우드 환경에서 손쉬운 확장
- Quay.io, Ceph, InSight(OpenScape)와 같은 에코 환경 구성
- 강화된 보안모델 제공

Red Hat

OpenShift ACM

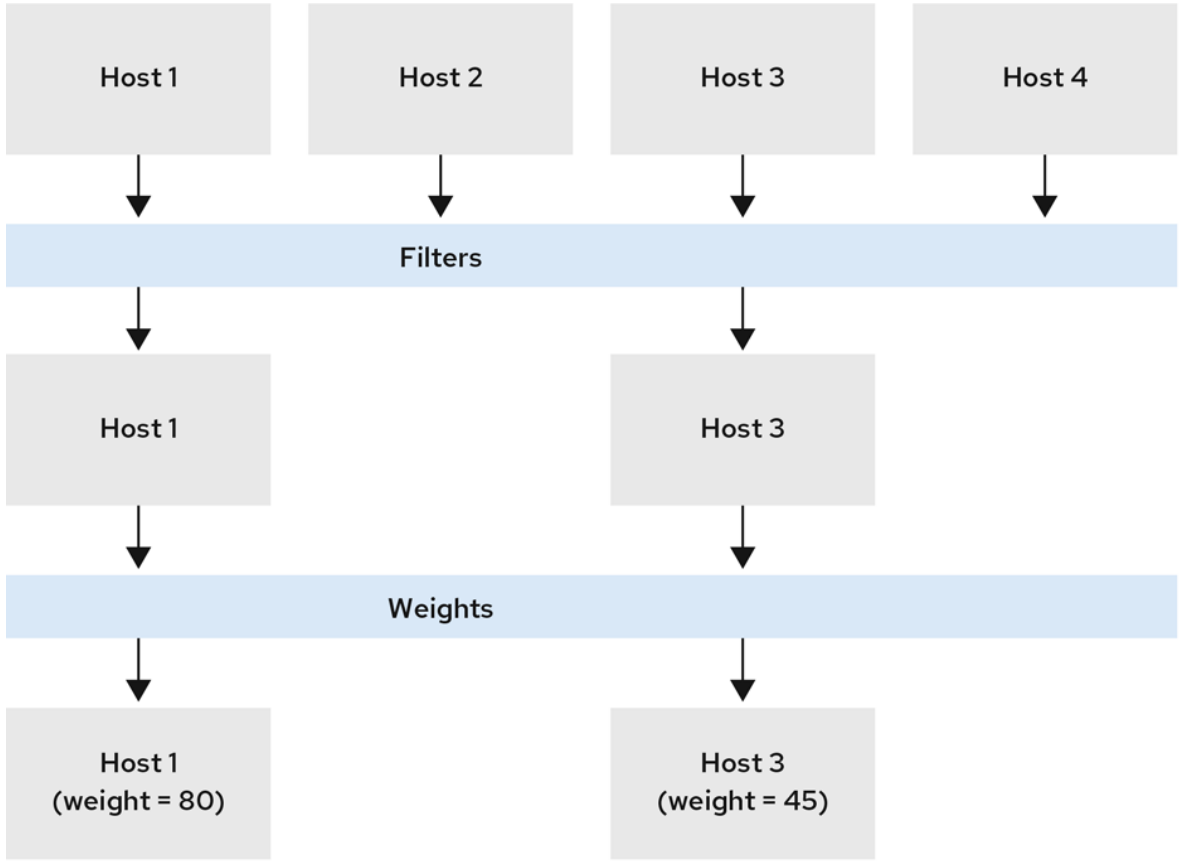
Components

ACM은 오픈시프트와 동일하게 통합된 CI/CD환경을 여러 클러스터에 제공한다. 이때 사용하는 개념은 Hub 및 Subscription기반으로 동작하게 된다. ACM는 Fleets라는 개념기반(앞서 이야기한 Placement)으로 클러스터를 관리한다.

오픈스택의 Placement와 비슷하지만 조금 다른 개념이다.

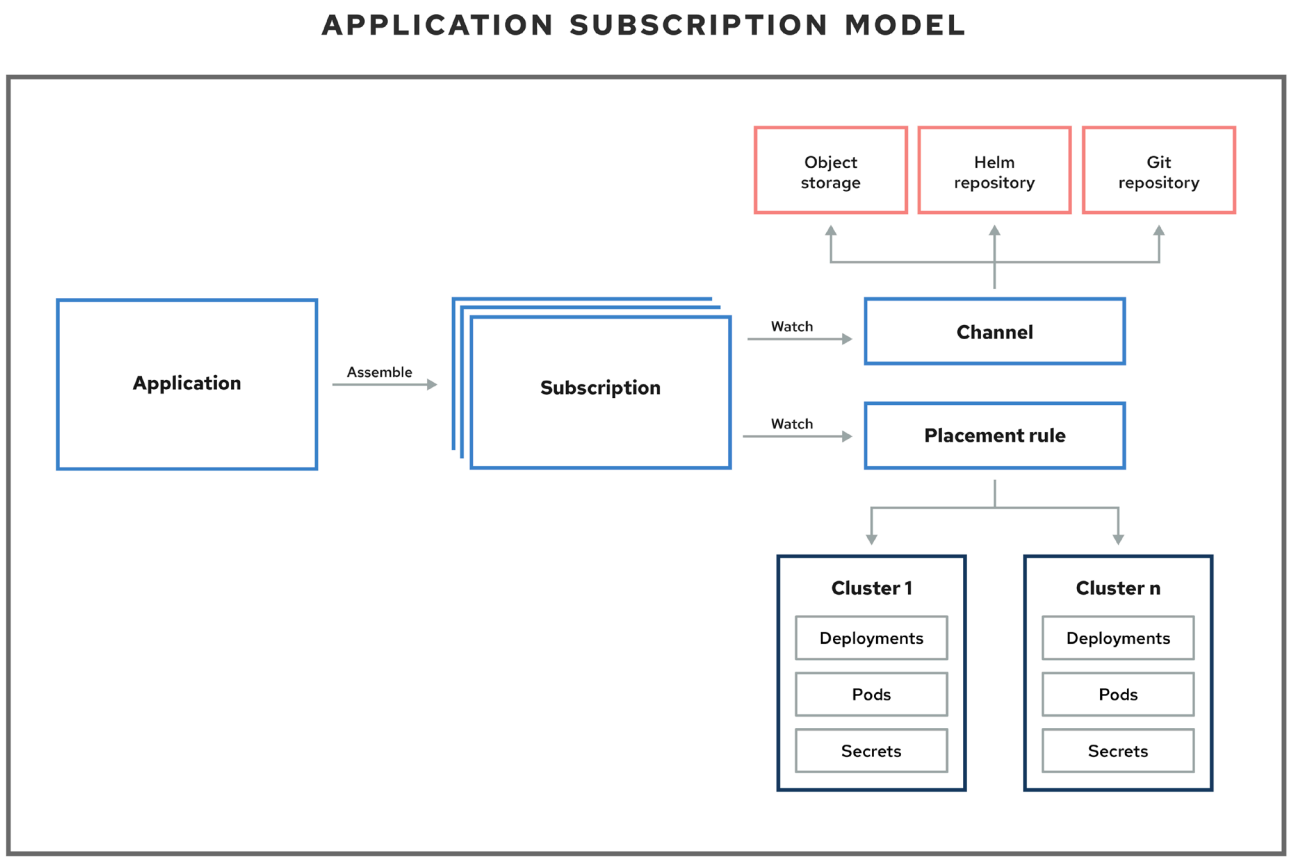
진행 전, 미리 한번 비교 해본다.

OpenStack Placement

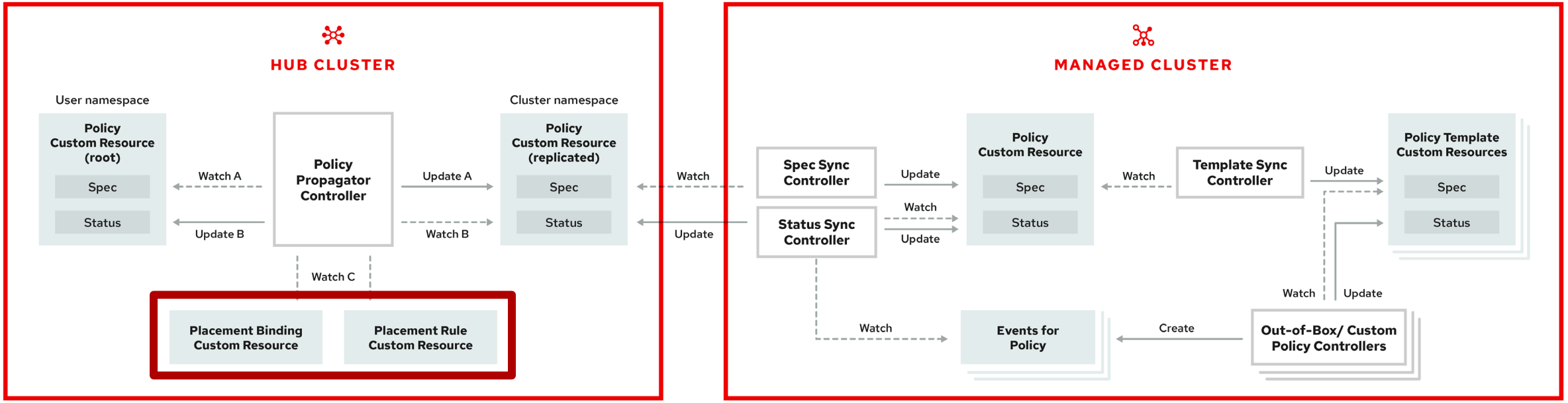


Red Hat

OCM/ACM Placement



Red Hat


HUB CLUSTER
User namespace
Policy Custom Resource (root)
Spec
Status
Watch A
Update B
Policy Propagator Controller
Update A
Watch B
Cluster namespace
Policy Custom Resource (replicated)
Spec
Status
Watch C
Placement Binding Custom Resource
Placement Rule Custom Resource
Watch
Update
MANAGED CLUSTER
Spec Sync Controller
Status Sync Controller
Update
Watch
Update
Policy Custom Resource
Spec
Status
Watch
Template Sync Controller
Update
Policy Template Custom Resources
Spec
Status
Watch
Update
Watch
Events for Policy
Create
Out-of-Box/ Custom Policy Controllers

Red Hat

Klusterlet

Components

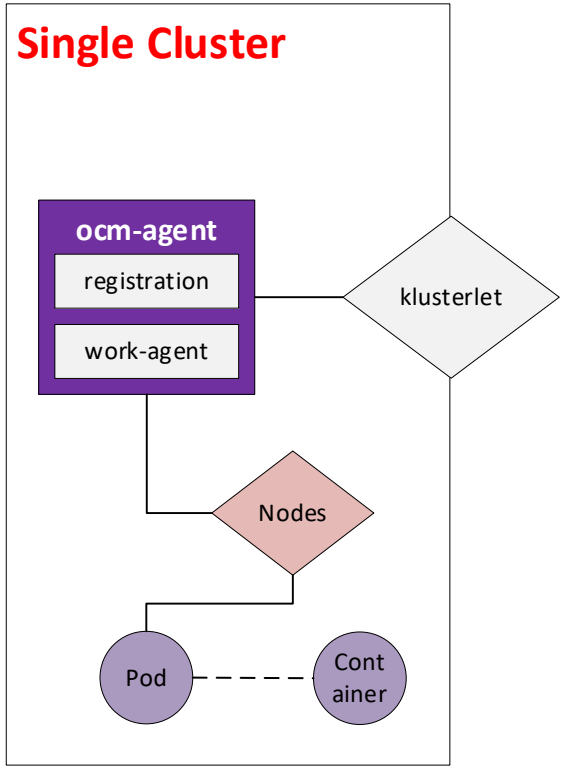
OCM 그리고 ACM에서는 Klusterlet은 매우 중요하다.

Klusterlet은 기존 쿠버네티스나 혹은 오픈 시프트에서 사용하는 kubelet하고 비슷한 기능을 한다.
다만, 차이점이라고 하면, "Node to Node"에서 "Cluster to Cluster"로 변경이 되었다.

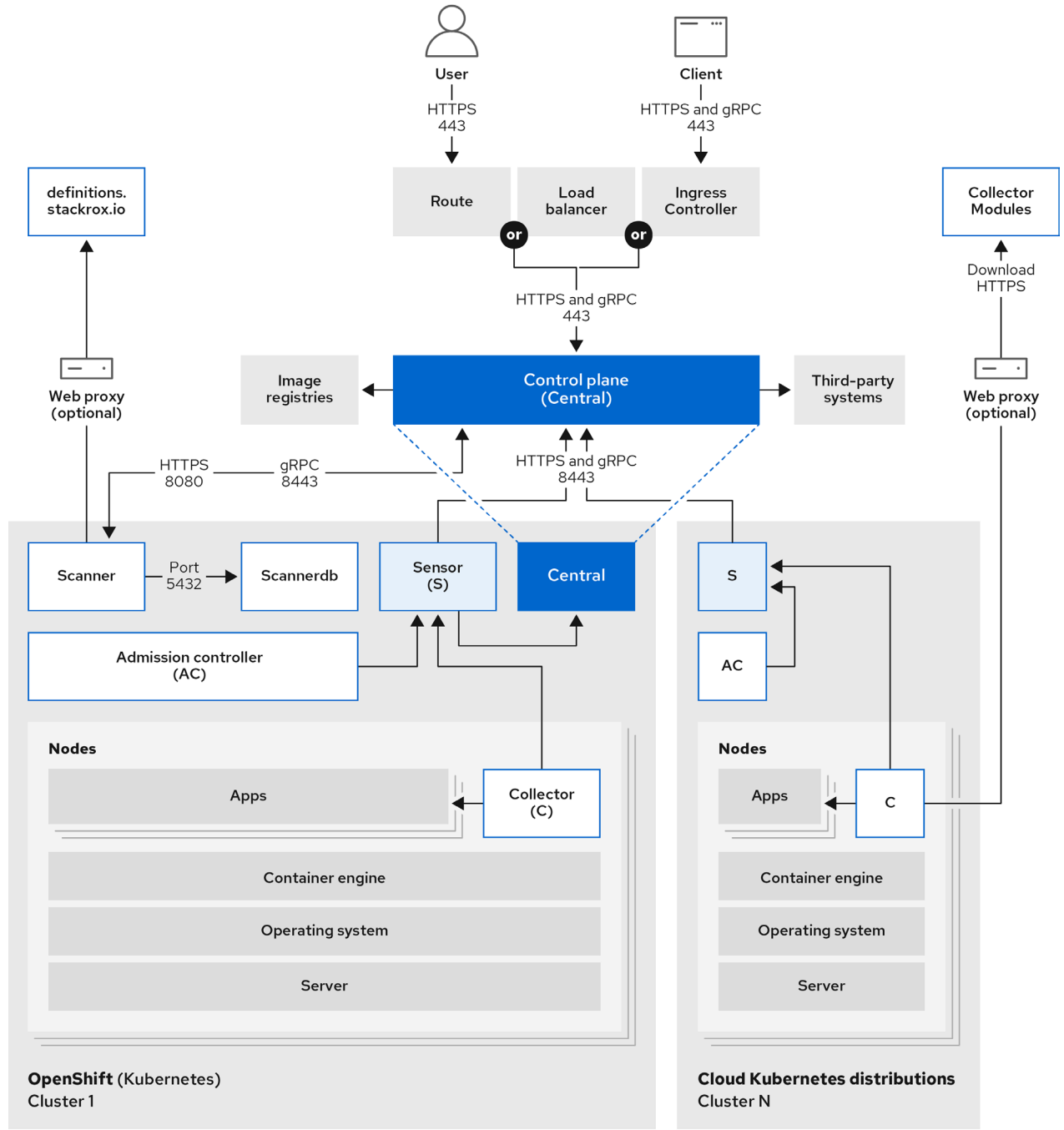
Red Hat

Klusterlet

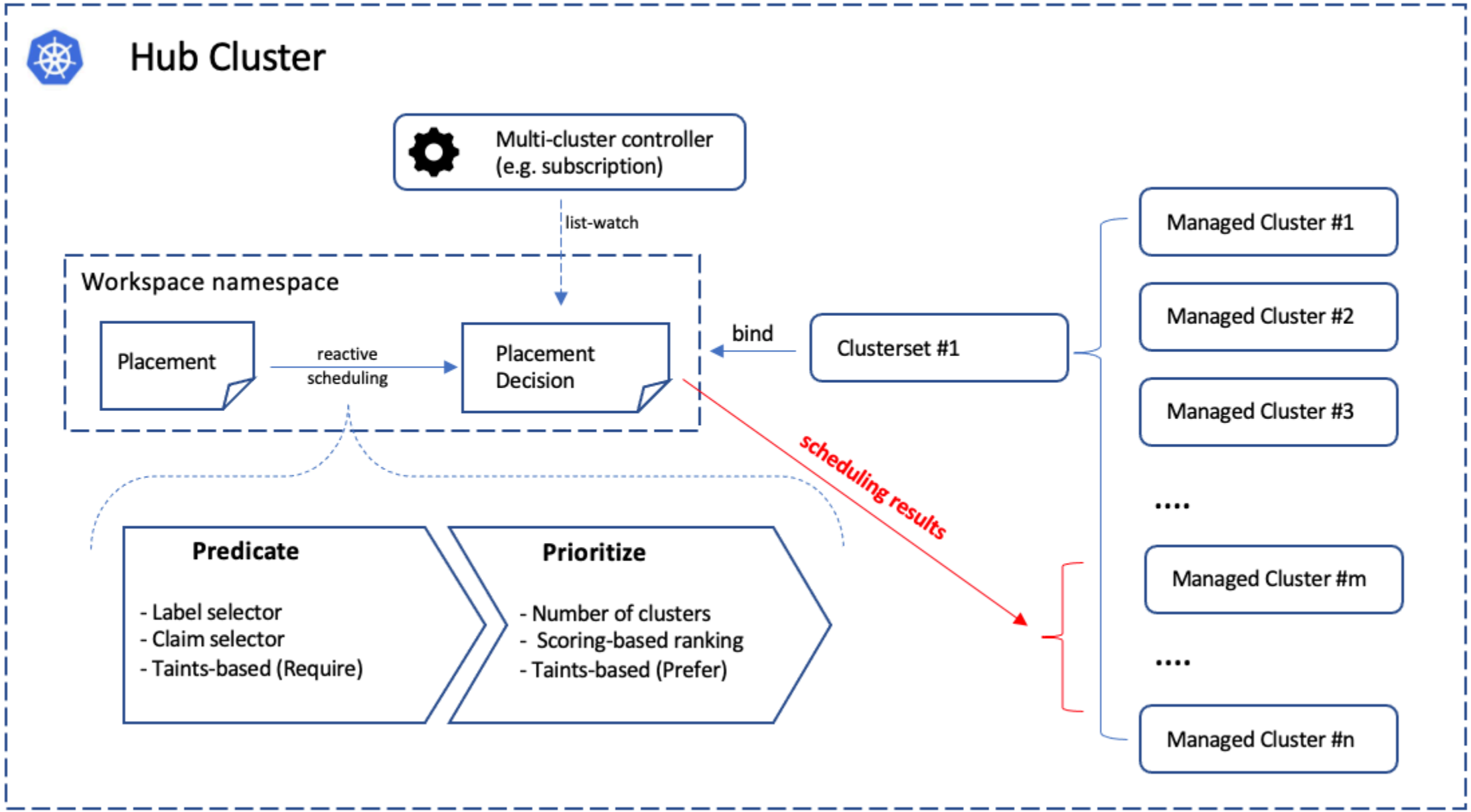
Components



Red Hat


User
HTTPS
443
Client
HTTPS and gRPC
443
definitions.
stackrox.io
Route
Load
balancer
Ingress
Controller
or
or
HTTPS and gRPC
443
Collector
Modules
Download
HTTPS
Web proxy
(optional)
Image
registries
Control plane
(Central)
Third-party
systems
Web proxy
(optional)
HTTPS
8080
gRPC
8443
HTTPS and gRPC
8443
Scanner
Port
5432
Scannerdb
Sensor
(S)
Central
S
Admission controller
(AC)
AC
Nodes
Apps
Collector
(C)
Container engine
Operating system
Server
Nodes
Apps
C
Container engine
Operating system
Server
OpenShift (Kubernetes)
Cluster 1
Cloud Kubernetes distributions
Cluster N

Red Hat


Hub Cluster
Multi-cluster controller
(e.g. subscription)
list-watch
Workspace namespace
Placement
reactive
scheduling
Placement
Decision
bind
Clusterset #1
Managed Cluster #1
Managed Cluster #2
Managed Cluster #3
....
Managed Cluster #m
....
Managed Cluster #n
scheduling results
Predicate
- Label selector
- Claim selector
- Taints-based (Require)
Prioritize
- Number of clusters
- Scoring-based ranking
- Taints-based (Prefer)

Placement

Components

```
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: placement1
  namespace: default
spec:
  numberOfClusters: 3
  clusterSets:
    - prod
```


https://open-cluster-management.io/concepts/placement/

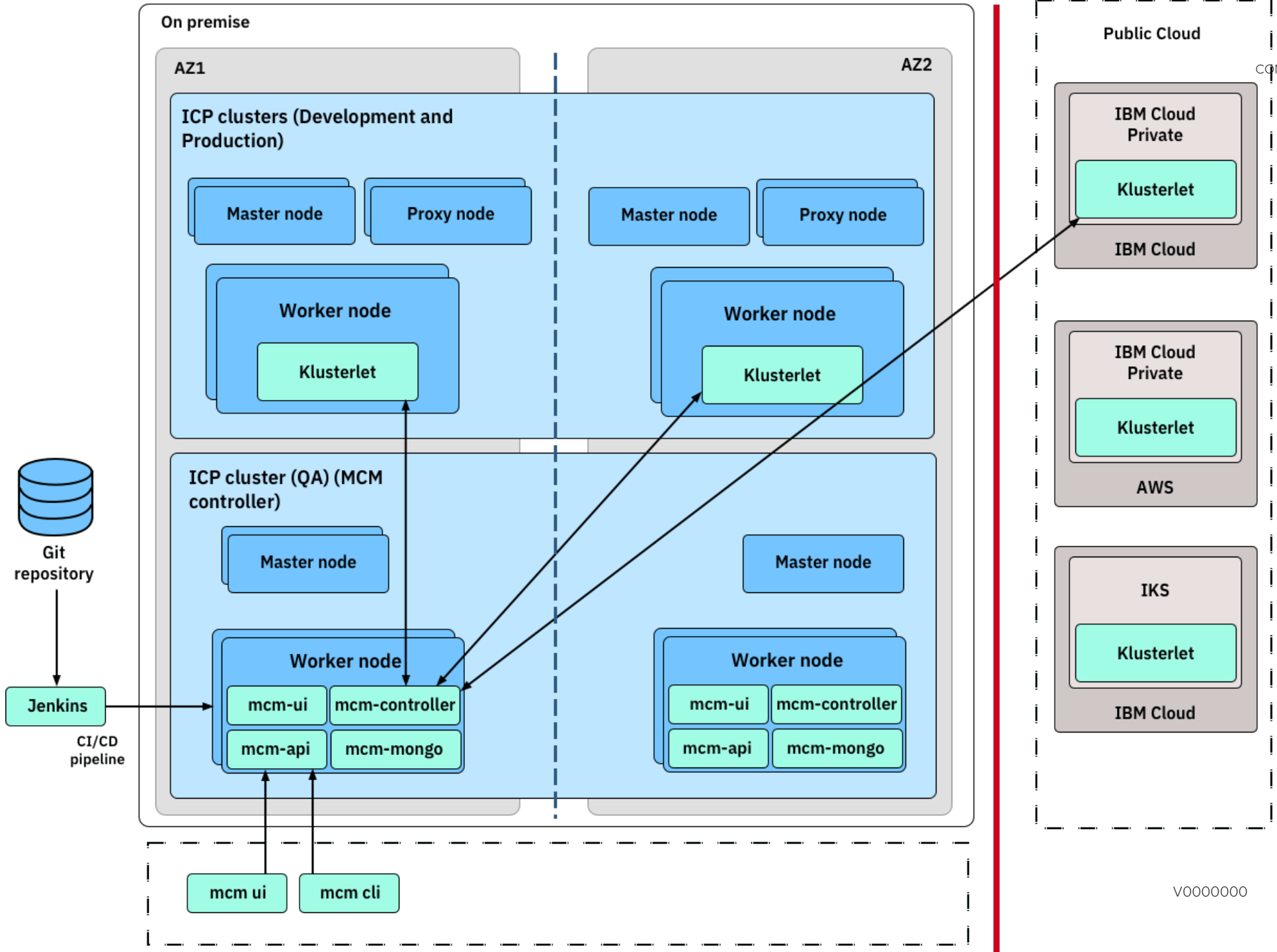
Placement

Components

```
predicates:
  - requiredClusterSelector:
      labelSelector:
        matchLabels:
          purpose: test
      claimSelector:
        matchExpressions:
          - key: platform.open-cluster-management.io
            operator: In
            values:
              - aws
```


Red Hat

https://open-cluster-management.io/concepts/placement/


On premise
AZ1
AZ2
ICP clusters (Development and Production)
Master node
Proxy node
Master node
Proxy node
Worker node
Klusterlet
Worker node
Klusterlet
ICP cluster (QA) (MCM controller)
Master node
Master node
Worker node
mcm-ui
mcm-controller
mcm-api
mcm-mongo
Worker node
mcm-ui
mcm-controller
mcm-api
mcm-mongo
Git repository
Jenkins
CI/CD pipeline
mcm ui
mcm cli
Public Cloud
IBM Cloud Private
Klusterlet
IBM Cloud
IBM Cloud Private
Klusterlet
AWS
IKS
Klusterlet
IBM Cloud

Red Hat

DAY 2

- Role based Access Control

ROLE BASED

ROLE-BASED

ACCESS CONTROL

Red Hat

https://open-cluster-management.io/concepts/placement/

ROLE-BASED

ACCESS CONTROL

open-source Identity and Access Management (IAM)

Red Hat Identity Management (IdM)

- **Identity** (machine, user, virtual machines, groups, authentication credentials)
- **Policy** (host based access control)
- **Audit** (this component is deferred)

Keycloak

https://open-cluster-management.io/concepts/placement/

Propagator/Propaganda

Propagator

watch and update

https://github.com/open-cluster-management-io/governance-policy-propagator

https://open-cluster-management.io/concepts/placement/

propagation.go

update

https://github.com/open-cluster-management-io/governance-policy-propagator/blob/main/controllers/propagator/propagation.go

```
54  const (
55          startDelim             = "{{hub"
56          stopDelim              = "hub}}"
57          TriggerUpdateAnnotation = "policy.open-cluster-management.io/trigger-update"
58  )
```


Red Hat

https://open-cluster-management.io/concepts/placement/

propagation.go

watch

https://github.com/open-cluster-management-io/governance-policy-propagator/blob/main/controllers/propagator/propagation.go

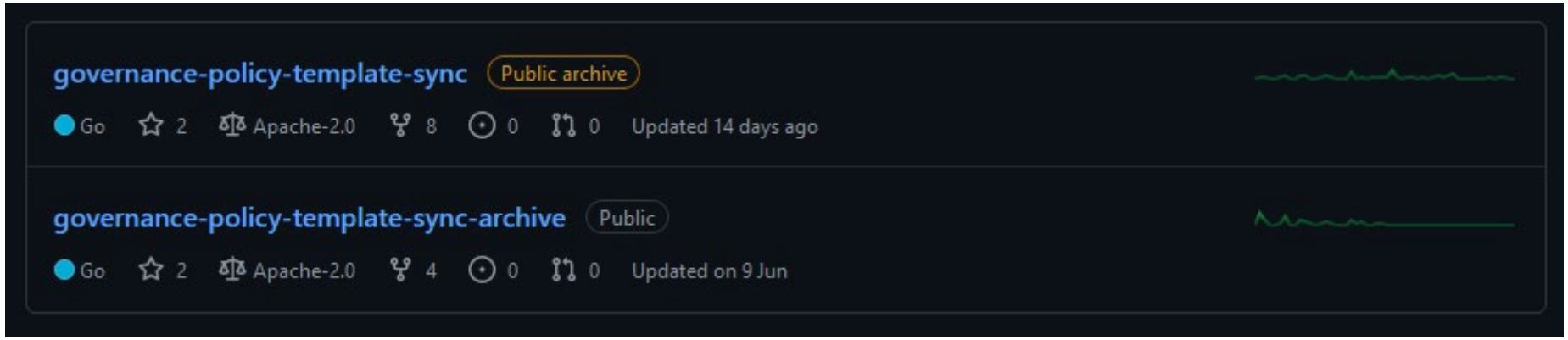
```
243  // decisionResult contains the result of handling a placement decision of a policy. It is intended
244  // to be sent in a channel by handleDecisionWrapper for the calling Go routine to determine if the
245  // processing was successful. Identifier is the PlacementDecision, with the ClusterNamespace and
246  // the ClusterName. TemplateRefObjs is a set of identifiers of objects accessed by hub policy
247  // templates. Err is the error associated with handling the decision. This can be nil to denote success.
248  type decisionResult struct {
249          Identifier      appsv1.PlacementDecision
250          TemplateRefObjs map[k8sdepwatches.ObjectIdentifier]bool
251          Err             error
252  }
```


Red Hat

https://open-cluster-management.io/concepts/placement/

template-sync

the archived



https://open-cluster-management.io/concepts/placement/

template-sync

https://github.com/open-cluster-management-io/governance-policy-template-sync-archive

```
38  var log = ctrl.Log.WithName(ControllerName)
39
40  //+kubebuilder:rbac:groups=policy.open-cluster-management.io,resources=*,verbs=get;list;watch;create;update;patch;delete
41  //+kubebuilder:rbac:groups=core,resources=events,verbs=get;list;watch;create;update;patch;delete
```


Red Hat

https://open-cluster-management.io/concepts/placement/

policy controller

NIST-800-53

https://github.com/open-cluster-management-io/governance-policy-addon-controller

Red Hat

https://open-cluster-management.io/concepts/placement/

policy controller

NIST-800-53

https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

https://csf.tools/reference/nist-sp-800-53/r5/cm/cm-2/

https://open-cluster-management.io/concepts/placement/

policy controller

image

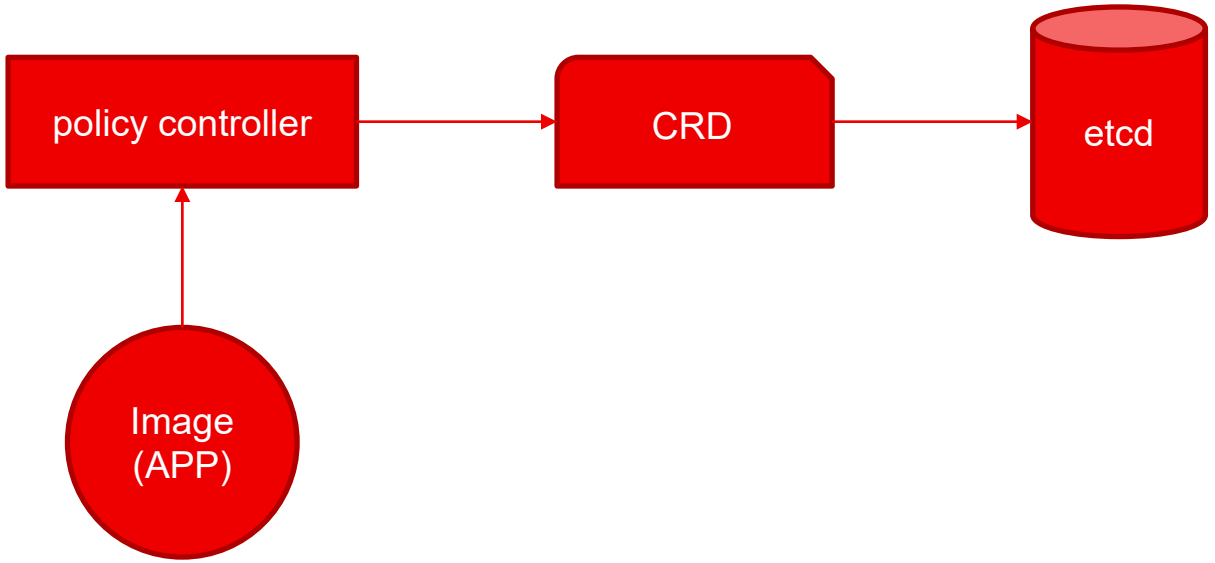
```
  image: quay.io/open-cluster-management/governance-policy-propagator:latest
  imagePullPolicy: Always
  name: governance-policy-propagator
  ports:
  - containerPort: 8383
    name: http
    protocol: TCP
serviceAccountName: governance-policy-propagator
```


Red Hat

https://open-cluster-management.io/concepts/placement/

policy controller

image



https://open-cluster-management.io/concepts/placement/

IAM Policy

controller

https://github.com/stolostron/iam-policy-controller

Red Hat

https://open-cluster-management.io/concepts/placement/

IAM Policy

사용자화

https://github.com/stolostron/governance-policy-framework

https://open-cluster-management.io/concepts/placement/

subscription

사용자화

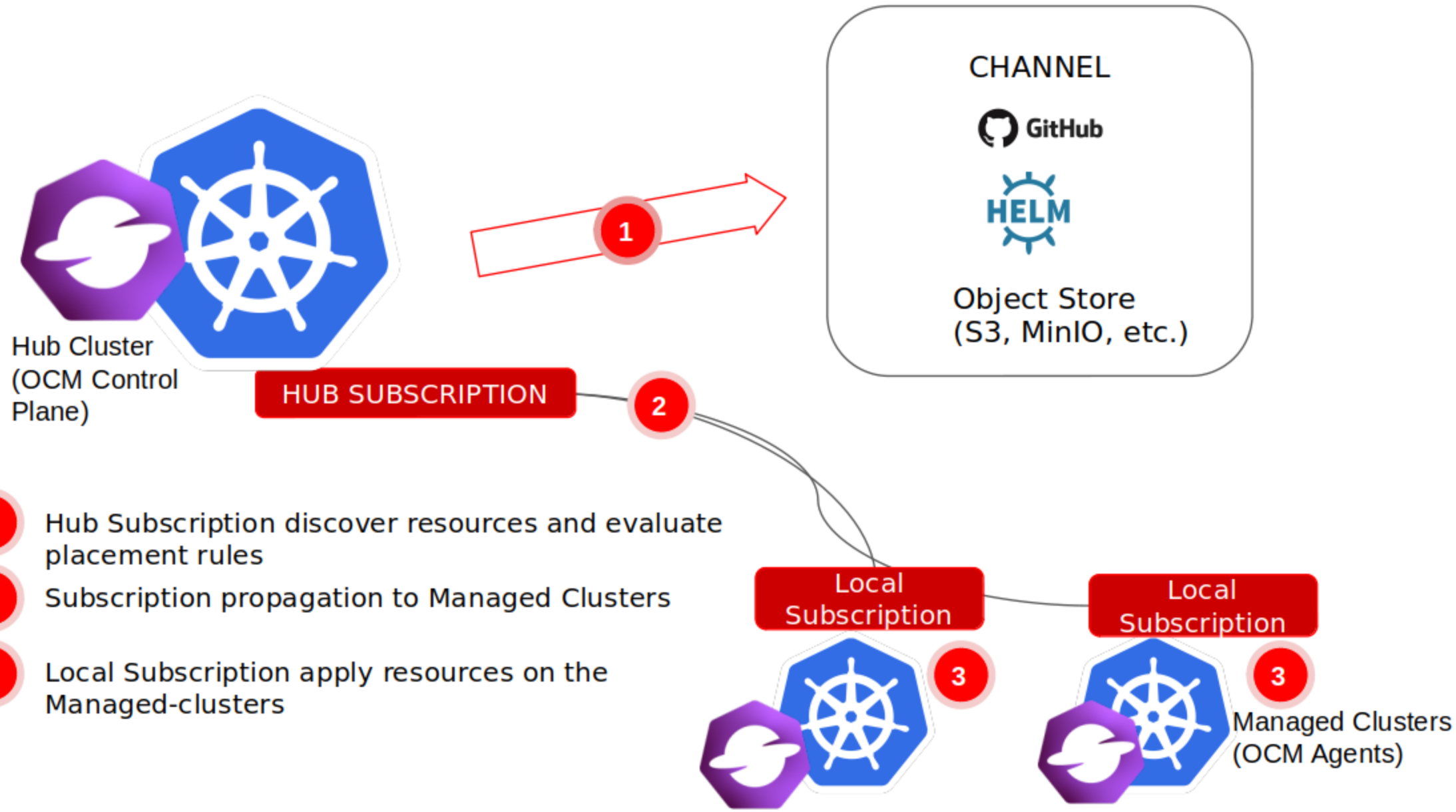
https://github.com/open-cluster-management-io/multicloud-operators-subscription

Red Hat

subscription

사용자화

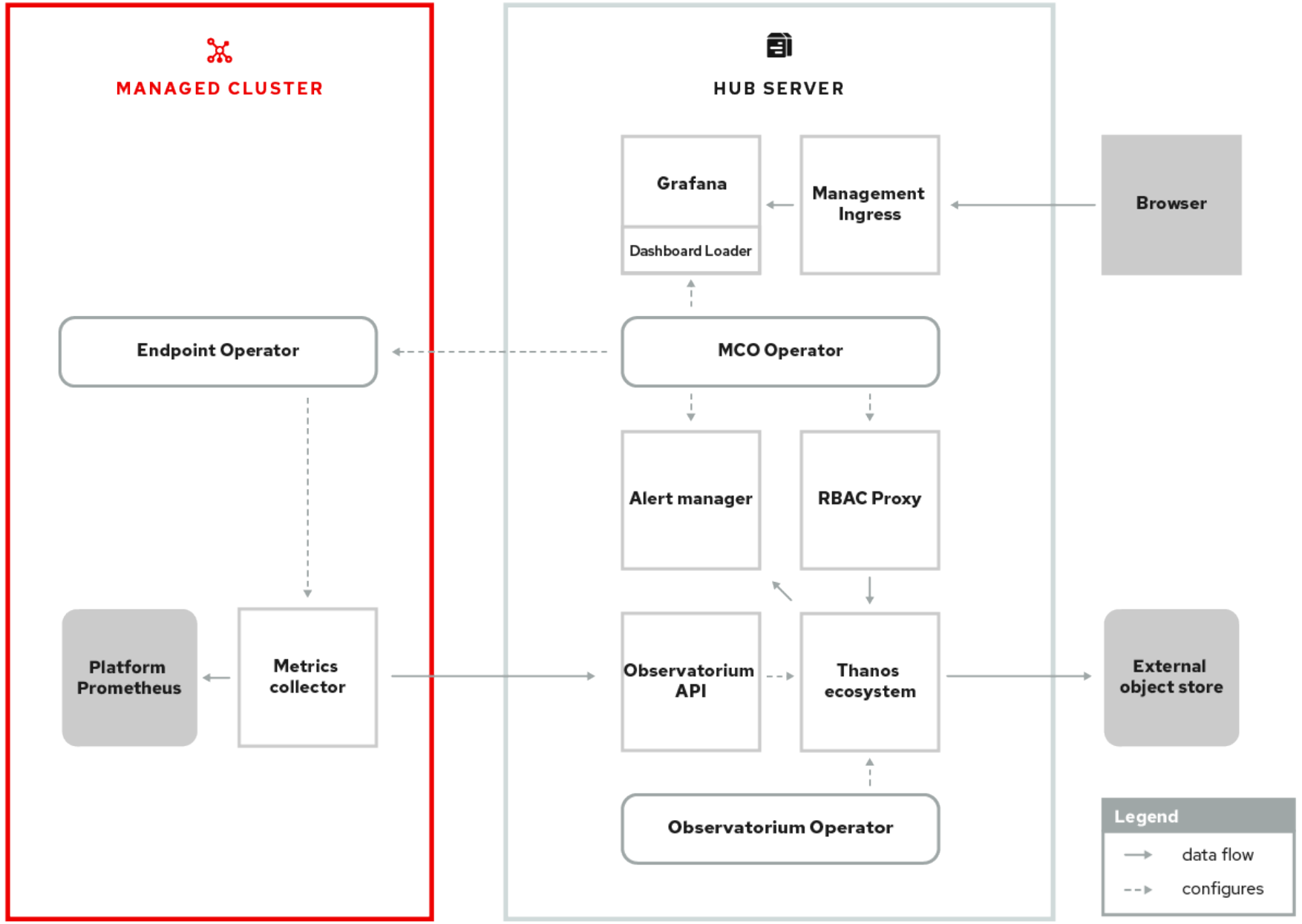
https://github.com/open-cluster-management-io/multicloud-operators-subscription


CHANNEL
GitHub
HELM
Object Store
(S3, MinIO, etc.)
1
Hub Cluster
(OCM Control
Plane)
HUB SUBSCRIPTION
2
1 Hub Subscription discover resources and evaluate placement rules
2 Subscription propagation to Managed Clusters
3 Local Subscription apply resources on the Managed-clusters
Local Subscription
Local Subscription
3
3
Managed Clusters
(OCM Agents)

Observability

Observability

모니터링 및 분석


MANAGED CLUSTER
HUB SERVER
Grafana
Dashboard Loader
Management Ingress
Browser
Endpoint Operator
MCO Operator
Alert manager
RBAC Proxy
Platform Prometheus
Metrics collector
Observatorium API
Thanos ecosystem
External object store
Observatorium Operator
Legend
data flow
configures

Red Hat

Thanos Project

사용자화

https://open-cluster-management.io/concepts/placement/

Thanos Project

모니터링

이 프로젝트의 기반은 **Prometheus**이며, 컨테이너 기반의 pv,pvc에서 오랜 기간 데이터 저장 및 분석하는 도구.

프로젝트 홈페이지는 다음과 같다.

https://thanos.io/

https://open-cluster-management.io/concepts/placement/

Thanos Project

조건

- Put Prometheus in the same failure domain. This means same network, same datacenter as monitoring services.
- Use persistent disk to persist data across Prometheus restarts.
- Use local compaction for longer retentions.
- Do not change min TSDB block durations.
- Do not scale out Prometheus unless necessary. Single Prometheus is highly efficient (:

Red Hat

https://open-cluster-management.io/concepts/placement/

Thanos Project

조건

- Sidecar: connects to Prometheus, reads its data for query and/or uploads it to cloud storage.
- Store Gateway: serves metrics inside of a cloud storage bucket.
- Compactor: compacts, downsamples and applies retention on the data stored in cloud storage bucket.
- Receiver: receives data from Prometheus's remote-write WAL, exposes it and/or upload it to cloud storage.
- Ruler/Rule: evaluates recording and alerting rules against data in Thanos for exposition and/or upload.
- Querier/Query: implements Prometheus's v1 API to aggregate data from the underlying components.
- Query Frontend: implements Prometheus's v1 API proxies it to Query while caching the response and optional splitting by queries day.

https://open-cluster-management.io/concepts/placement/

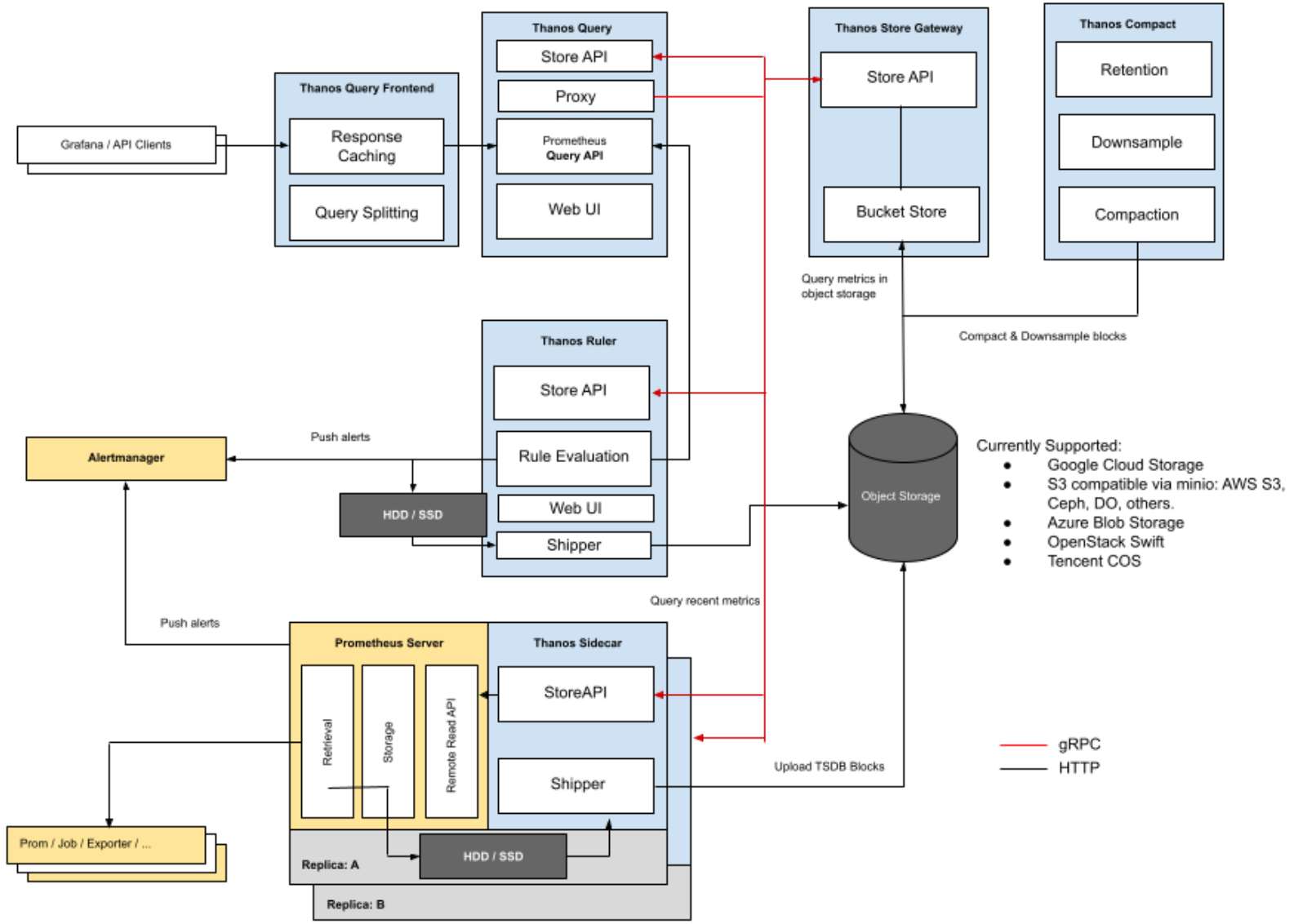
Thanos Project

what is the Sidecar?

- Thanos integrates with existing Prometheus servers through a Sidecar process, which runs on the same machine or in the same pod as the Prometheus server.
- The purpose of the Sidecar is to backup Prometheus data into an Object Storage bucket, and give other Thanos components access to the Prometheus metrics via a gRPC API.

https://open-cluster-management.io/concepts/placement/

사이드카

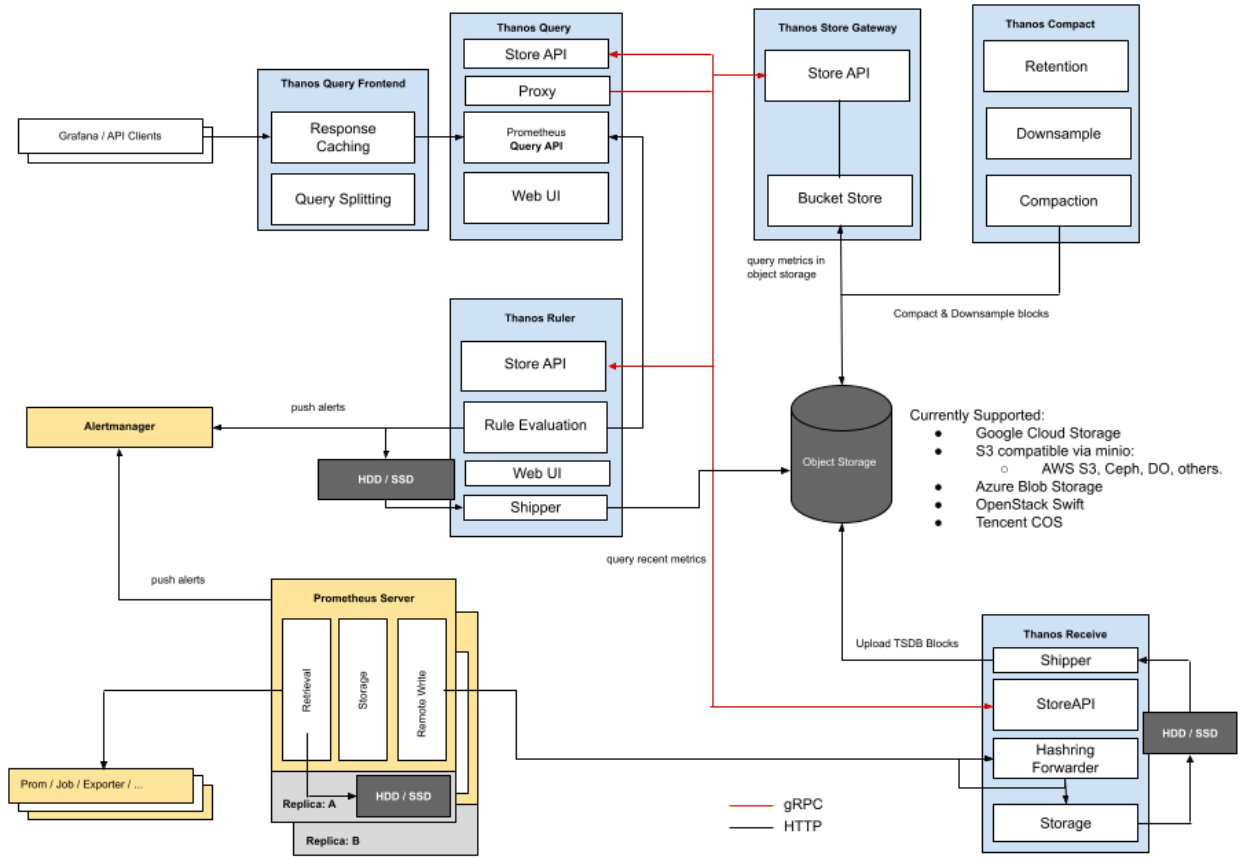


https://open-cluster-management.io/concepts/placement/

Thanos Project

사용자화

배포



https://open-cluster-management.io/conce

Red Hat

gatekeeper

사용자화

https://github.com/open-policy-agent/gatekeeper

Red Hat