|  |
| --- |
|  |
| Multi-Cluster Network Policy |
| **Software Design Specification** |
| **Red Hat** |
| **Aviad Hadad Shoham Danino** |
| **05.08.2021** |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description of Change | Author | Date |
| 1 |  | Aviad and Shoham | 05.08.2021 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[Introduction](#_1fob9te) **3**

[Requirements](#_3znysh7) **3**

[Design Overview](#_2et92p0) **4**

[Architecture & Flow](#_tyjcwt) **4**

[Detailed design](#_3dy6vkm) **5**

[Sequence diagrams of the flows](#_1t3h5sf) **5**

[Component/Service Coast guard controller](#_4d34og8) **6**

[ETCD structure](#_dylz8q9tbol) 6

[CRD schema](#_3irqmovjjyhn) 6

[Coastguard Stack & Infrastructure](#_17dp8vu) **7**

[Coastguard Frontend](#_3rdcrjn) **7**

[Logging](#_35nkun2) **8**

[Monitoring and Measurement](#_1ksv4uv) **8**

[Testing](#_44sinio) **8**

[Cloud costs](#_2jxsxqh) **8**

[Development plan](#_z337ya) **8**

[Reviewers](#_3j2qqm3) **9**

# Introduction

The purpose of this project is to introduce multi-cluster network policy to a submariner cluster.

Kubernetes Network Policy allows cluster administrators to configure granular isolation rules using Network Policy objects. A network policy is a specification of how groups of pods (using labels) are allowed to communicate with each other and other network endpoints.

The purpose of this project is to provide a mechanism for Network Policies across multiple clusters so that policy enforcement from a single cluster will seamlessly expand and work across clusters.

Currently, in a submariner environment which includes multiple k8n clusters, there is no way to define network policy between the two clusters. An administrator must manually collect ranges of ip addresses if he wants to limit traffic. Given the kubernetes pod network model, which is a flat network, this is not granular enough.

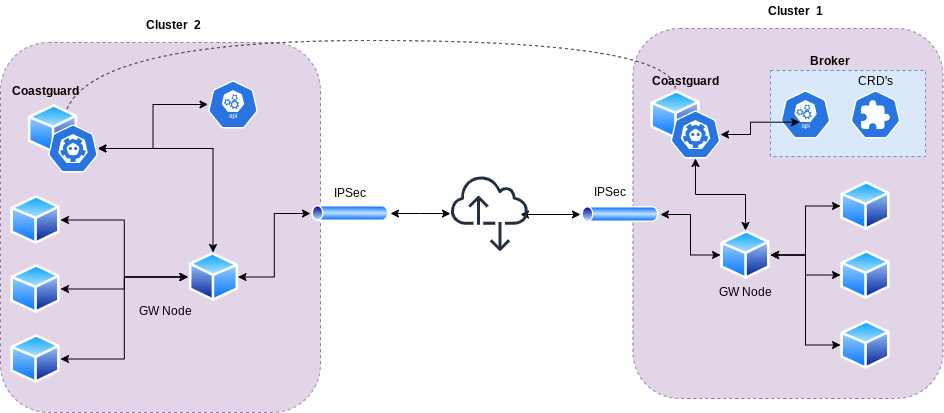
Our service, named **coastguard,** will allow users to define multi-cluster network policies, and **coastguard** will translate the multi-cluster policies into relevant single-cluster network policies, and will keep those policies up to date after new deployments in each cluster.

# Requirements

|  |
| --- |
| Connected Clusters without any network policies should keep the default behavior |
| Allow frontend pods from a remote cluster in the same project to access the backend pods on the local cluster |
| Allow all the PODs from the same project in a remote cluster to access the backend POD on the local cluster |
| Allow frontend POD from a different namespace in a remote cluster to access the backend POD on the local cluster |
| Allow all the PODs from the same namespace which are either in the local cluster or remote cluster to access the logging POD on the local cluster |
| When a new cluster is added, , the resources on the new cluster need to be explored, and processed so it will comply with existing policies. |
| When a cluster previously connected via submariner/federation is removed, any auxiliary resources, rules, or artifacts maintained by the submariner network policy implementation must be properly cleaned up. |

# Design Overview

# Architecture & Flow



Coastguard aims to implement multi-cluster network policies by managing single-cluster network policies in each cluster.

Our main challenge is tracking changes in deployments and pods in each cluster, and thus keeping the single-cluster network policies up to date.

Coastguard is composed of:

1. Operator pattern = k8n controller + custom CRD. The custom crd will define the multi-cluster network policies.
2. Controller will use embedded etcd. as etcd is a distributed key-value store, etcd is responsible for updating the changes from the local cluster, to the controller in the remote cluster.
3. Controller tracks deployments using the Admiral API that federates resources to submariner data planes running in multiple clusters. Admiral api is an existing code component in submariner.

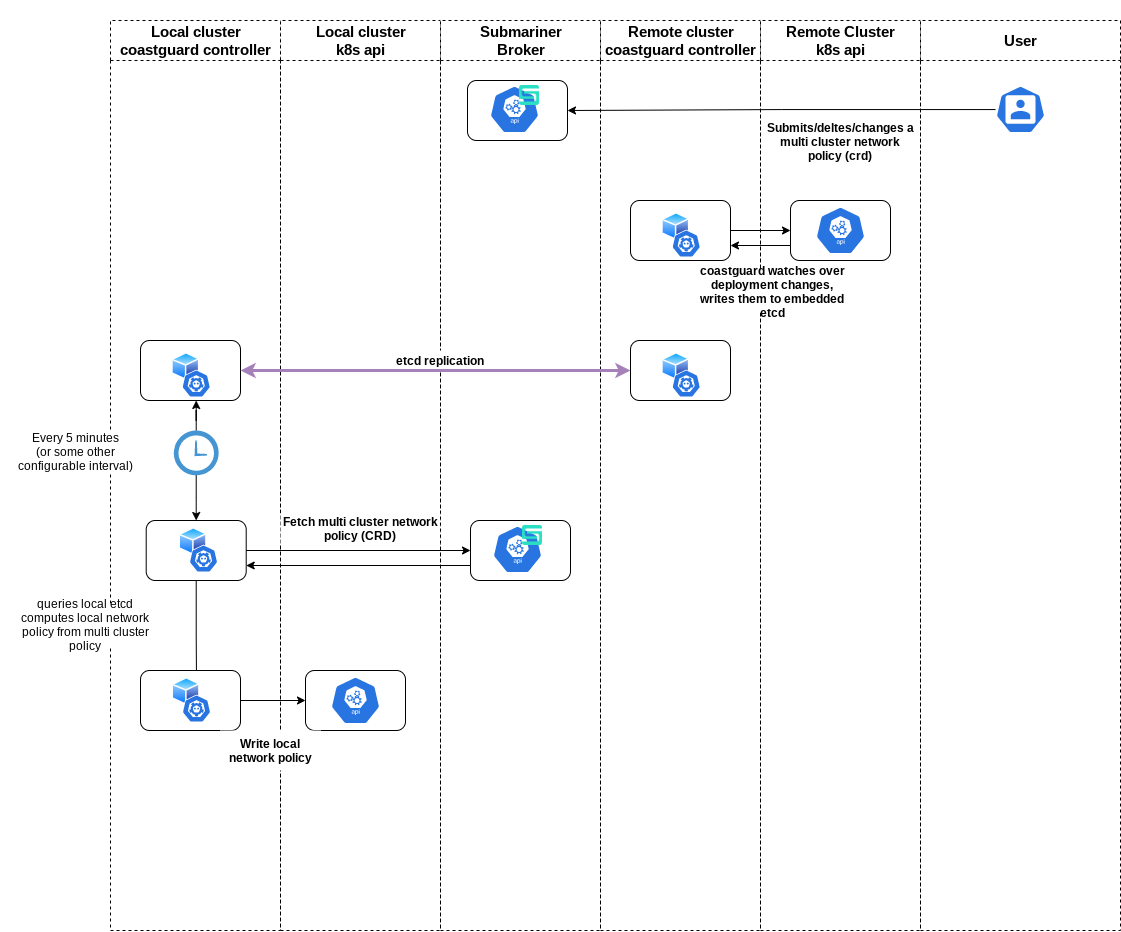
**The core idea here is: that each controller has a reconciliation loop, where it re-generates a local policy based on remote network policy. Each controller has an embedded etcd server which holds the ip address and labels of all pods across all clusters. The controller will generate a “local”/regular network policy and will apply it to k8s api.**

**Creating the local network policy is done in an immutable way, and if in a given timeframe no changes occurred, the controller will simply apply the same local network policy to the k8s api server.**

# Detailed design

# Sequence diagrams of the flows

The following is a diagram of the main “reconciliation loop”. This is our main flow, and any subsequent will use this flow to do the bulk of the work.



We will have 4 main flows:

1. a clusters joins the cluster set
2. a deployment change happens on any cluster
3. a user adds/changes new multi-network policy is added
4. a cluster is removed from the set

Flows 2-3 have no special additions beyond executing the reconciliation loop.

For flow 1 - when a cluster first joins the cluster set, and the coastguard controller runs for the first time, the controller will need to map all existing namespaces and pods on the k8s cluster. It will do so by running the API equivalent of *`kubectl get namespaces` and `kubectl get pods`.*

For flow 4 - once a cluster leaves the cluster set, the coastguard controller on the leaving cluster should delete all it’s entries from etcd.

# Component/Service Coast guard controller

### ETCD structure

[etcd docs](https://etcd.io/docs/v3.4/dev-guide/interacting_v3/#read-keys)

In general, etcd is a key value store. It has the ability to query based on key prefixes.

So, for each pod, we will have the following entries, where ‘$clustername’ for example is a placeholder for clustername.

1. ‘$clustername\_$namespace\_$podname’: ip address
2. ‘$clustername\_$namespace\_’: [list of labels]

We will need the following queries:

1. get all pod ip, in cluster X with namespace Y

*query: etcdctl get --prefix ‘$clustername\_$namespace`*

1. get all pod ips in cluster X with namespace Y with label Z

*query: etcdctl get --prefix ‘labels\_$clustername\_$namespace`*

We will get from etcd a list of all pods, for every pod, all it’s labels. We will filter in memory for label Z.

1. get all pod ips in cluster with label Z

*query: etcdctl get --prefix ‘labels\_$clustername`*

Same idea as 2.

# ETCD is holding all the sensitive data in our network, the ETCD security is out of scoop in this project.

### CRD schema

our multi-cluster network policy CRD:

**apiVersion**: networking.submariner.io/v1

**kind**: MultiClusterNetworkPolicy

**metadata**:

**name**: test-network-policy

**namespace**: default

**spec**:

**clusterSelector**: cluster1

**podSelector**:

**matchLabels**:

**role**: db

**policyTypes**:

- Ingress

- Egress

**ingress**:

- **from**:

- **ipBlock**:

**cidr**: 172.17.0.0/16

**except**:

- 172.17.1.0/24

- **namespaceSelector**:

**matchLabels**:

**key**: value

- **podSelector**:

**matchLabels**:

**key**: value

- **clusterSelector**: value

**ports**:

- **protocol**: TCP

**port**: 6379

**egress**:

- **to**:

- **ipBlock**:

**cidr**: 172.17.1.0/16

**except**:

- 172.17.2.0/24

- **namespaceSelector**:

**matchLabels**:

**key**: value

- **podSelector**:

**matchLabels**:

**key**: value

- **clusterSelector**: value

# Coastguard Stack & Infrastructure

Coastguard will be written in go.

Coastguard will use embedded etcd.

Testing will be done with ginkgo, and the project’s existing e2e testing orchestration infrastructure named “shipyard”.

# Coastguard Frontend

Since coastguard will use custom CRD’s to manage settings, a user will be able to use familiar kubectl tool as a frontend to manage coastguard.

kubectl -f apply

# Logging

Json logs to stdout. In Kuberents outputting to stdout is recommended.

# Monitoring and Measurement

submariner exposes metrics to prometheus. If we will be able to use common infrastructure we might want to expose some metrics ourselves. The only relevant metric seems to be raising a “flat” (1/0 gauge in prometheus terms) whenever a configuration change happens. That way coastguard users will be able to overlay this information (configuration change time) with other graphs (such as errors over time) to debug quickly.

# Testing

Generating local network policy will be tested with unit testing. meaning converting between multi-cluster network policy to single cluster.

Testing controller communication will be e2e.

# Cloud costs

no cloud costs

# Development plan

|  |  |  |
| --- | --- | --- |
| Week | Milestones/Tasks to be completed | Notes |
| 1 | more research, finish technical design |  |
| 2-3 | 1.kubernetes controller poc 2.embedded etcd poc | we will split these tasks between us |
| 4-5 | unify the two poc’s above,  implement core business logic, meaning generating different filtering policies, tracking changes in clusters and writing to etcd | Big challenges:  1. integrating with existing submariner broker api  2. tracking kubernetes deployment changes |
| 6 | make sure everything works with “manual” deployment,  documentation |  |
| 7-8 | work on providing an easy install script, preferably via subctl |  |
|  |  |  |

# Reviewers

Detail who reviewed your work from the company (if relevant).

|  |  |  |
| --- | --- | --- |
| Department/Role | Name | Date |
| Research | Nir Yechiel  nyechiel@redhat.com |  |
| Research | Maayan friedman  maafried@redhat.com |  |
|  |  |  |
|  |  |  |
|  |  |  |