Evaluation of Istio and Cilium to operate workloads across Kubernetes clusters

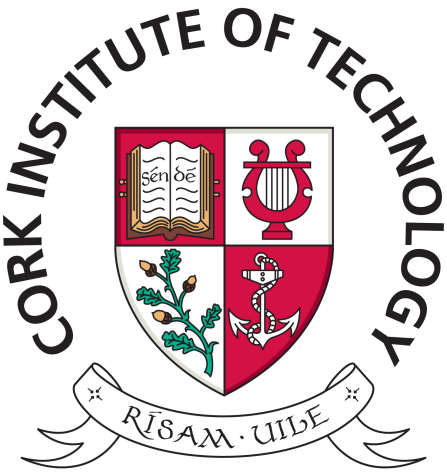
Thesis · May 2019

DOI: 10.13140/RG.2.2.12292.88969

1 author:

Francois Nature
cit
1 PUBLICATION **0** CITATIONS

SEE PROFILE



Evaluation of Istio and Cilium to operate workloads across Kubernetes clusters

by

Francois Nature

This thesis has been submitted in partial fulfillment for the degree of Master of Science in MSc in Cloud Computing

in the
Faculty of Engineering and Science
Department of Computer Science

May 2019

Declaration of Authorship

I, Franois Natur , declare that this thesis titled, 'THESIS TITLE' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an masters degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institiute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- I understand that my project documentation may be stored in the library at CIT, and may be referenced by others in the future.

Signed: _____

Date: _____

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science
Department of Computer Science

Master of Science

by Francois Nature

Today companies must manage an increasing number of separate Kubernetes clusters. This led technologies such as Cilium and Istio to propose solutions to ease the communication and portability of workloads across them. These solutions are still nascent, the multicluster implementation is still considered challenging and there hasn't been any academic evaluations of the multicluster features of Cilium and Istio. This research built proof-of-concepts and evaluate these solutions on several aspects such as feature set, complexity, security and performance. Results show that Istio provides more possibility of network designs but is also more complex to implement. Both are able to provide service discovery but in one design scenario, Istio doesn't deliver service name consistency. Both are able to provide load-balancing across clusters. Cilium has an advantage in security by also allowing the network plugin policies to span across clusters. Finally, there isn't any performance overhead to run a multicluster operation with Istio but Cilium presents unexpected timeouts that would require further investigation.

Acknowledgements

I would like to extend my sincere gratitude to my research supervisor David Stynes for his previous guidance and support. To my wife Sunayana, not even one phase of this journey would have been possible without you.

Contents

List of Figures

Chapter 1

Introduction

A Cloud Native Application (CNA) is an application that is designed to leverage the benefits of running in a cloud environment. It is typically composed of microservices running in containers and orchestrated by Elastic container platforms which provide high-availability, load-balancing, scalability to the containers. Today the open-source Kubernetes is the leading platform [1] and the rest of the paper focus on this solution.

A CNA is traditionally managed by a single Kubernetes cluster as shown in figure 1.1. A cluster is a set of nodes that can be bare metal servers or virtual machines, and containers (or pods) run on these nodes. Services are the entities that load-balance the requests to the pods. Kubernetes is originally created to run stateless applications (an applications that don't need to store their state), but supports now stateful applications (applications that store their state, example a database application).

Today companies have to manage more and more container clusters. This situation relates to different use-case including: having to managing multiple on-premises clusters,

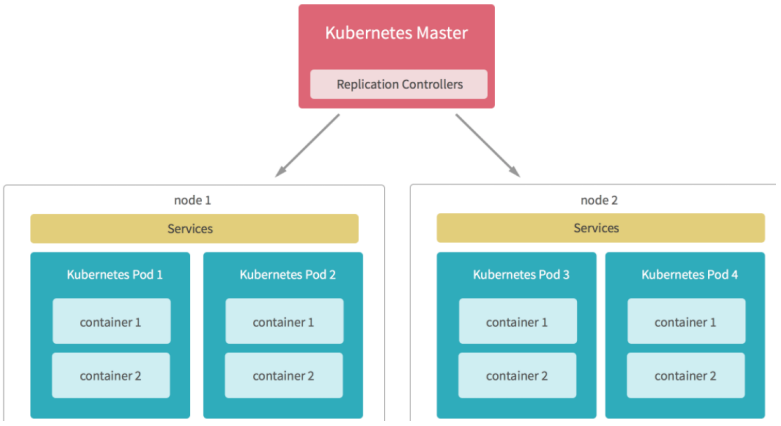


FIGURE 1.1: A single Kubernetes cluster.

bursting containers from on-premise to the cloud, Fault isolation (better to have multiple small clusters than a single large one), need to build new cluster to fulfill security and data sovereignty, clusters may correspond to different business units, going beyond scalability limit of a single cluster, reaching closer to the users by creating new clusters closer to them (global presence), Disaster recovery across regions, avoiding Cloud Service Provider (CSP) lock-in by operating across cloud vendors. [2]

In the latest Kubecon 2018, "Kubernetes is still considered as "difficult to run and operate at scale, particularly for multicloud/hybrid environments- spanning on-premises data centers and public cloud infrastructure." [3]. most existing vendors do not solve the question of how to manage clusters in multiple locations[4] "nothing has yet solved the problem of running a distributed system that spans multiple clusters"[5]. Issues to operate multiple clusters are numerous, they were summarized in different categories: Application topology, Data replication, Traffic Routing, Service Discovery, Logging and monitoring, Networking, CI/CD, security, image distribution, scarcity [2].

Networking is one of the challenge: The aim is to provide pod-to-pod connectivity across clusters [5]. There can be many solutions to connect multiple Virtual Private Cloud across regions [6]. A common solution is using a VPN. Others are looking at avoiding VPN because of the performance penalty of IPsec and packet overhead for example [7]. Coddiwomple [8] proposes an alternative to connect the clusters and others propose to use IPv6 instead [7]. Istio is working also on a method to avoid VPN, but the software is not released yet [9]. These approaches are categorized as zero vpn solutions in this paper.

New open-source technologies have emerged to make multicluter easier and extend the capabilities of a single cluster: Kubernetes federation v2 [10], Istio multicluster [11] and Cilium clustermesh [12]. These solutions are very novel, and were presented at the latest Kubecon in december 2018. Kubernetes Federation v2 is a complex project that aims to resolve lots of use-cases but it is still under development (alpha maturity). In comparison Istio and Cilium are considered more stable versions and resolve a few use-cases.

Istio and Cilium have different and common features. Istio is an application that runs in a Kubernetes cluster and brings benefits such as security, visibility and traffic management. Cilium is a network plugin that provides the network connectivity to a cluster and bring features such as security and performance. Cilium eases the pod-to-pod network connectivity across clusters. Istio multicluster and Cilium clustermesh both allow to resolve service names across clusters (service discovery) and load-balance pods traffic across clusters. Additionally these technologies can work together, Cilium and Istio can be deployed and work together in the same cluster.

These solution have also the potential to ease the deployment of stateful application across clusters. Stateful application have different requirements than the stateless ones. Cockroachdb team mentions some of the current challenges to run stateful application with Kubernetes and presents Istio multicluster as one of the solution [5].

There hasn't been to the best of our knowledge any academic publication to evaluate Istio multicluster and Cilium clustermesh. Therefore our aim is to evaluate and compare these open-source solutions to operate stateless and stateful workloads across multiple kubernetes clusters.

Chapter 2

Background

Istio provides a service mesh to a Kubernetes cluster. "With A Service Mesh, a given microservice wont directly communicate with the other microservices. Rather, all service-to-service communications will take place on top of a software component called the Service Mesh proxy (or sidecar proxy). Sidecar or Service Mesh proxy is a software component that is co-located with the service in the same VM or pod (Kubernetes)" [13]. Istio is an "inter-service communication infrastructure". Functions of the mesh are: Resiliency for Inter-Service Communications (circuit-breaking, retries and timeouts, fault injection, fault handling, load balancing, and failover), Routing (based on, certain headers, versions, etc), Observability (all the traffic data is captured at the sidecar proxy level, sidecar proxy can publish those data), security (distributed firewall, Role based access control, TLS encryption between service-to-service) [13]. With a service Mesh, "the developers dont need to worry about the inter-service communication and most of the other crosscutting features of a service such as security, observability, etc" [13]. Istio uses the Envoy proxy, a high-performance proxy developed in C++, and leverage many of its features to provide the benefits of the service mesh. In the open-source community, Istio has a lot of traction. "Istio is stable and feature rich.. and is backed by Lyft, Google and IBM... has pioneered many of the ideas currently being emulated by other service meshes" [14]. Today Istio has 15.9K stars and 332 contributors on Github [15]. There has been no academic evaluation on Istio multicluster capability and only one research of Istio itself: [16] investigated the service isolation capability of Istio and concluded in June 2018 that "Istio provides granular access control at the application layer, the architecture and current implementation failed to meet the requirements of availability and performance by introducing a critical dependency in the request path".

Cilium is a network plugin which aims to transparently secure the network connectivity between application services in Kubernetes. It provides security to the layer 3 - 7 network

stack . At its foundation it relies on a new Linux kernel technology called extended BPF (eBPF), "which enables the dynamic insertion of powerful security visibility and control logic within Linux itself".[17] . More specifically, it replaces the module kube-proxy in Kubernetes. Kube-proxy is the proxy in Kubernetes that provides load-balancing of traffic to the pods and firewalling with linux firewall iptables. Cilium replaces iptable and use eBPF to achieve firewalling [18]. Cilium doesn't filter on traditional IP but on service / pod / container identity [17]. Cilium uses also the same Envoy proxy as Istio to apply Layer 7 security policies to the pods [19].

Regarding the academic research, Cilium performance was investigated and compared to docker swarm overlay based on iptable in the research [20]. They found that Cilium outperforms docker swarm overlay in terms of total throughput without filtering rules, but that the application of the filtering rules didn't have an impact on the total throughput for both docker swarm overlay and Cilium. In another research [21], eBPF was proven to provide better performance over iptables . It must be noted also that Cilium uses also the power of eBPF to run the multicluster routing according to Thomas Graf (Cilium Clustermesh) [22].

There hasn't been yet an academic study on Cilium performance compared to the other famous network plugins for Kubernetes (Calico, flannel etc) and no evaluation on the Cilium Clustermesh capabilities. A research on the web (not an academic evaluation) [23] shows that Cilium has a poorer maximum throughput and more resource consumption. In the contrary, Cilium provided benchmarks showing that they outperform other network plugins Calico and Flannel [24].

2.1 Multicluster designs

Here is a description of the different designs that Istio and Cilium propose to operate workloads across Kubernetes clusters.

2.1.1 Cilium ClusterMesh

Cilium proposes a design with a control plane running in each cluster 2.1. One of the requirement of Cilium is to have IP connectivity between all nodes. The full list of requirements are in [25] Cilium runs the state with an etcd cluster in each cluster and a cilium agent runs in each node to provide network connectivity to the pods. There should be no performance overhead to run pods across clusters: "Pod IP routing across multiple Kubernetes clusters at native performance via tunneling or direct-routing

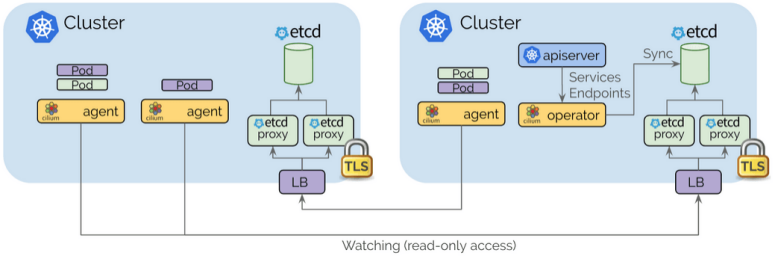


FIGURE 2.1: Cilium Clustermesh architecture

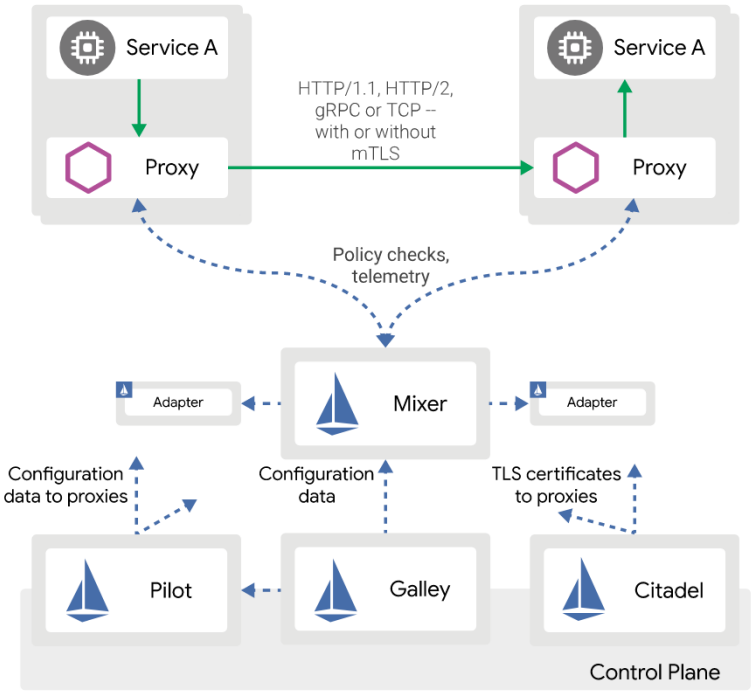


FIGURE 2.2: Istio control plane

without requiring any gateways or proxies" [26]. A full description of the architecture is available at [27].

2.1.2 Istio service mesh

Before diving into its multicluster solution, here is a recap of Istio. The Istio service mesh implementation is described in [28]. The control plane is composed of Mixer, Citadel, Galley and Pilot as shown in figure 2.2. The control plane is stateless and its configuration is stored in the Kubernetes etcd [29]. The data plane is formed by the Envoy sidecar proxies. A sidecar proxy is deployed in each pod. They provide many functions including service discovery and service isolation [28].

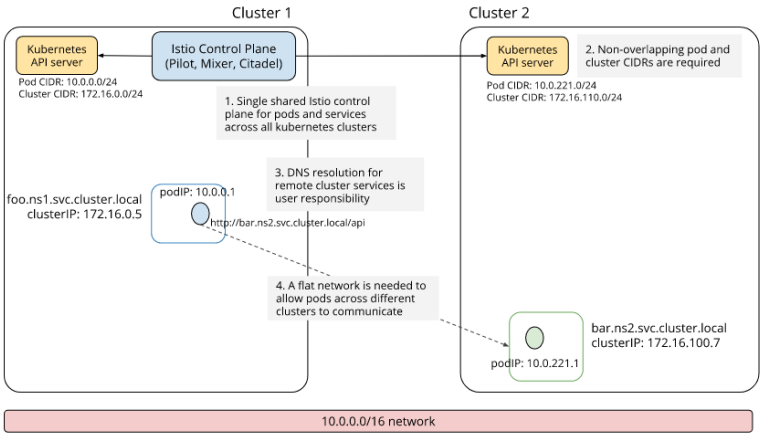


FIGURE 2.3: Istio service mesh across cluster with single Istio control plane and VPN connectivity

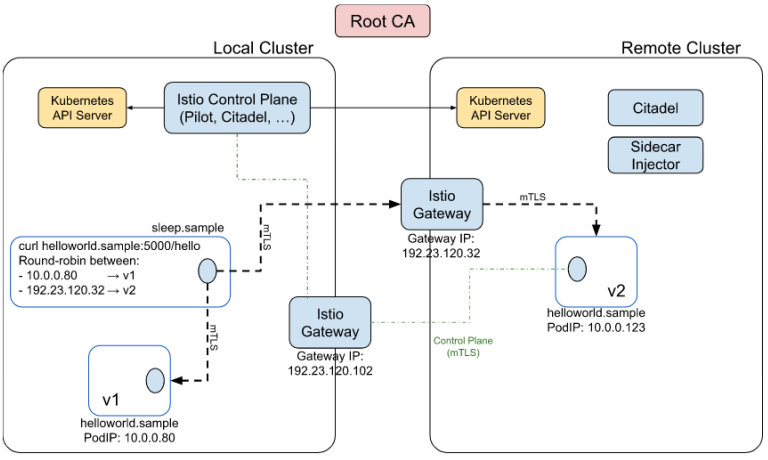


FIGURE 2.4: Istio service mesh across cluster with single Istio control plane and gateway connectivity

2.1.3 Istio Multicluster with single control plane and VPN connectivity

The figure 2.3 shows the architecture of this solution. Description is available at [30]. There should be no performance overhead to run pods across clusters as pod connect directly on a flat network.

2.1.4 Istio Multicluster with single control plane and gateway connectivity

The figure 2.4 shows the architecture of this solution. Description is available at [30]. The solution doesn't require a VPN across clusters, and can rely on existing Istio ingress gateways that are typically backed by external load balancers.

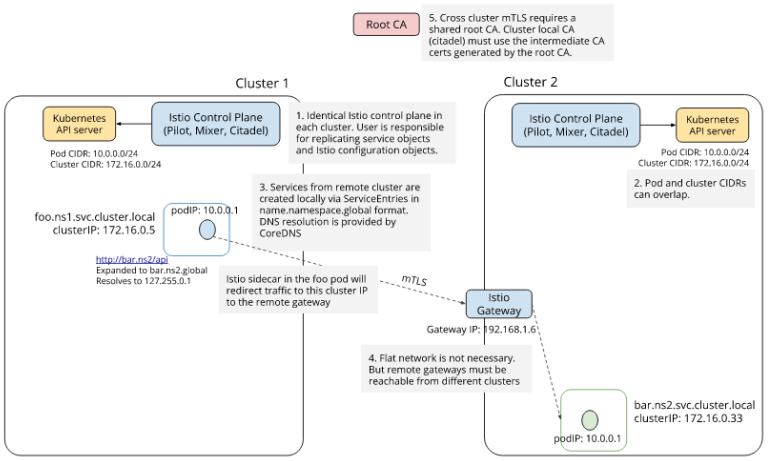


FIGURE 2.5: "Istio mesh spanning multiple Kubernetes clusters with direct network access to remote pods over VPN"

2.1.5 Istio Multicluster with multiple control planes and gateway connectivity

The figure 2.5 shows the architecture of this solution. Description is available at [30]. The solution doesn't require a VPN across clusters, and can rely on existing Istio ingress gateways that are typically backed by external load balancers.

Chapter 3

Problem - Evaluation of Istio and Cilium to operate workloads across Kubernetes clusters

3.1 Problem definition

The aim of this research is to evaluate and compare Cilium and Istio to operate stateless and stateful workloads across multiple kubernetes clusters.

3.2 Questions

- How do Istio and Cilium multicluster solutions compare in terms of feature set, complexity, security and performance for the operation of stateless workloads?
- How do Istio and Cilium allow the deployment of stateful workloads across clusters?
- What is the feasibility and benefits of implementing zero VPN solutions with these solutions?

3.3 Functional requirements

To answer these questions, several requirements were found to evaluate these solutions. They were inspired by the Cilium literature on multicluster use-cases and the academic evaluation of Istio service isolation in [16]. Note that Cilium published a list of use-cases after this research in [31].

- Requirement 1 - Service Discovery - The solution should permit to access services across clusters. Services become global services that can be accessed from both clusters.
- Requirement 2 - Consistent service name across clusters - This means each service can be resolved across clusters with a unique name. This requirement provides the benefit of lowest implementation cost inside microservices.
- Requirement 3 - Load Balancing - The solution should be able to load-balance global service traffic to deployment pods across multiple clusters. There should be the ability to apply weights. Example, an operator can load-balance half of the traffic to both clusters, or decide to only send one quarter to a remote cluster.
- Requirement 4 - Service isolation - It should be possible to enforce service isolation across clusters.
- Requirement 5 - No performance overhead - As shown by the designs of Istio and Cilium in 2.1, there shouldn't be a performance overhead attributed to the multicluster solution itself compared to the operation of a single cluster. The additional overhead and latency should only come from the network separating each cluster.
- Requirement 6 - Failover - The solution should allow to failover a service to another cluster with no downtime when underlying backend pods fail.
- Requirement 7 - Bursting - One should be able to burst a service based on local metrics to another cluster.
- Requirement 8 - Stateful application - The solution should support the deployment of a stateful application across clusters.
- Requirement 9 - Zero VPN - One should be able to operate a multicluster solution without the need to setup a VPN solution.

3.4 Non-functional requirements

- Requirement 1 - Service Discovery. It should be possible to configure a service to become global with minimal configuration needed.
- Requirement 2 - Consistent service name across clusters. An operator shouldn't have to implement inside a microservice a different name to call a remote service. There must be minimal configuration from the multicluster solution to achieve that.

- Requirement 3 - Load Balancing. The configuration should allow to load balance traffic with weight and minimal configuration required.
- Requirement 4 - Service isolation. The solution should allow the application of layer 3,4 and 7 policies across clusters.
- Requirement 5 - No performance overhead. It is expected that latencies and success rates between services should be similar in single and across clusters.
- Requirement 6 - Failover. When backend pods fails, the traffic should be redirected with no downtime.
- The other requirements were not tested 4.14.

Chapter 4

Implementation Approach

4.1 Methodology

The methodology consists of several tests that evaluates the requirements. They are subsequently implemented in both Cilium Clustermesh and Istio multicluster.

4.1.1 Test 1 for requirement 1 and 2 - Service Discovery with service name consistency - Figure 4.1

This test evaluates that service discovery is working and evaluates that the service name is consistent across clusters.

- Service A on Cluster 1 can resolve Service B on Cluster 2. Service B on Cluster 1 can resolve Service A on Cluster 2.
- Service A can resolve Service B with the same name as defined in Cluster 2. Service B can resolve Service A with the same name as defined in Cluster 1.

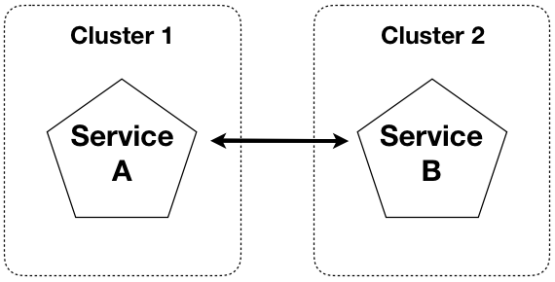


FIGURE 4.1: Service discovery

4.1.2 Test 2 - Load balancing - Figure 4.2

The test evaluates that the traffic to a shared service can be load-balanced successfully across clusters and that the weight can be changed.

Service A is defined on cluster 1. Service C is defined as a shared service: it load balances traffic to deployment pods in both cluster 1 and 2.

- Service A sends requests and it is verified that it receives replies from Service C originating from both cluster 1 and 2.
- The weight on service C is changed: 1/4 of traffic comes cluster 1 and 3/4 comes from cluster 2.

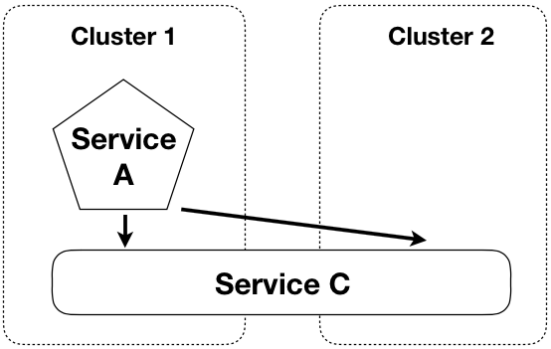


FIGURE 4.2: Load balancing

4.1.3 Test 3 - Service Isolation

The test evaluates that service isolation can be achieved in a multicluster environment for both shared and none-shared services.

The approach is inspired by the research on service isolation in a single cluster with Istio from [16].

This requirement is evaluated using proof of concepts scenarios that are chosen to include typical use cases for service isolation :

- Bidirectional: allow 2 services to talk to each others.
- Unidirectional: allow 1 service to talk to another but not the other way around.
- Allow one service to be ”opened” to all the services. All other services are allowed to access it.

The two proof of concepts are:

- One environment where services are not shared (deployment pods exist only in one cluster.) Service A and Service B exist in cluster 1. Service C and Service D exist in cluster 2. This scenario is shown in figure 4.3

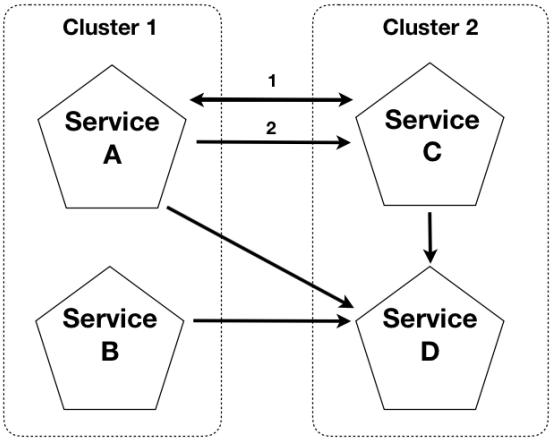


FIGURE 4.3: Service isolation

- One environment where services are shared across clusters (deployment pods exist in both clusters). Service C and D exist in both clusters. Service A exist in cluster 1 and service B exist in cluster 2. This scenario is shown in figure 4.4.

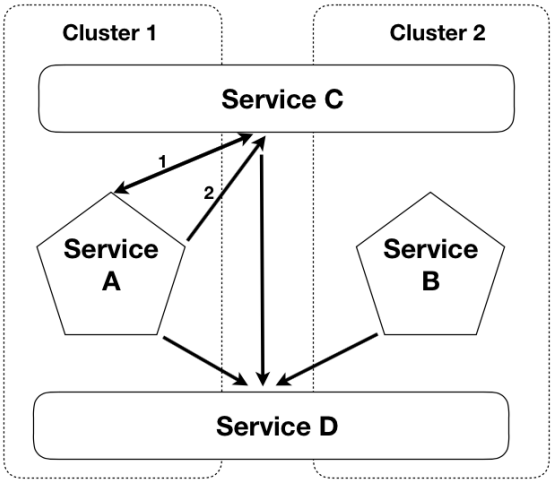


FIGURE 4.4: Service isolation with shared services

For both scenarios, the following service isolation rules are tested as shown in figures 4.3 and 4.4:

- As a first step, policy is enforced so that all services are isolated. It is evaluated that no access is permitted across services.

- Service D is allowed to be accessed by all other services. We verify that all services A,B,C can access service D.
- 1/ Service A is allowed to access service B and service C is allowed to access service B. It is evaluated that both access are permitted.
- 2/ Service A only is allowed to access service C. It is evaluated that service A is allowed access to service C and service C is not allowed access to service A.

The two scenarios are implemented and tested against all the layers that Cilium and Istio support. A research is conducted to find which layers Cilium and Istio support for service isolation. Potentially, the result is the implementation of the following testbeds shown in 4.5.

Cilium Clustermesh	shared services scenario with layer 3 rules
Cilium Clustermesh	shared services scenario with layer 4 rules
Cilium Clustermesh	shared services scenario with layer 7 rules
Cilium Clustermesh	none-shared services scenario with layer 3 rules
Cilium Clustermesh	none-shared services scenario with layer 4 rules
Cilium Clustermesh	none-shared services scenario with layer 7 rules
Istio single control plane	shared services scenario with layer 3 rules
Istio single control plane	shared services scenario with layer 4 rules
Istio single control plane	shared services scenario with layer 7 rules
Istio single control plane	none-shared services scenario with layer 3 rules
Istio single control plane	none-shared services scenario with layer 4 rules
Istio single control plane	none-shared services scenario with layer 7 rules
Istio multiple control plane	shared services scenario with layer 3 rules
Istio multiple control plane	shared services scenario with layer 4 rules
Istio multiple control plane	shared services scenario with layer 7 rules
Istio multiple control plane	none-shared services scenario with layer 3 rules
Istio multiple control plane	none-shared services scenario with layer 4 rules
Istio multiple control plane	none-shared services scenario with layer 7 rules

FIGURE 4.5: Service Isolation testbeds

4.1.4 Test 3.1 - Service isolation with Network plugin policy and Istio multicluster

This evaluation is added later during the research following results from previous service isolation tests. A detailed explanation is provided in results chapter 5.5. The goal is to evaluate if network policies provided by a Kubernetes network plugin (typically layer 3 to layer 4) can span multiple clusters in an Istio multicluster operation. The test is: 2 clusters implemented with Istio multicluster. Service A is running in cluster 1, and service B is running in cluster 2. Service B is allowed to access service A using network plugin policies. It is then evaluated if service B can access service A.

4.1.5 Test 4 - No performance overhead

Performance between services is evaluated within a single cluster as a baseline, then across another cluster. The nodes from all clusters are local or within the same region to each others, and not joined by a VPN. This way, the performance overhead of the internet is removed and results are compared to verify that there is no overhead induced by the multicluster solution.

Service isolation is also applied. Performance is tested again to verify that there is no performance overhead induced by the multicluster solution in a context of service isolation.

Requests are sent from A to C, and we evaluate the rate of success and latency to receive the reply from C. This is indicated by the dashed arrow in the following figures.

- Performance is evaluated between service A and C running in different nodes within a single cluster as shown in figure 4.6

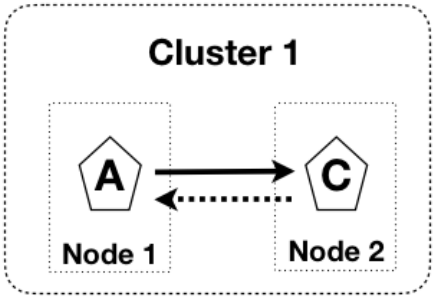


FIGURE 4.6: Performance in single cluster

- Performance is evaluated between service A to service C that is composed of 2 pods (shared service between 2 pods on a single cluster) within a single cluster. A is running on a different node than deployment pods C. 4.7

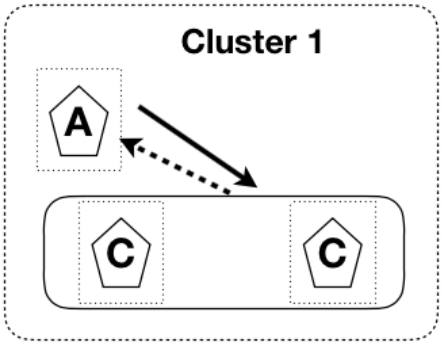


FIGURE 4.7: Performance to shared service in single cluster

- Performance is evaluated between service A running in cluster 1 and service C running in cluster 2 as shown in figure4.8

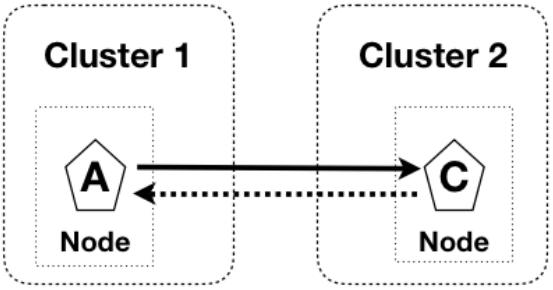


FIGURE 4.8: Performance across clusters

- Performance is evaluated between service A running in cluster 1 to shared service C spanning both cluster 1 and 2, as shown in figure 4.9

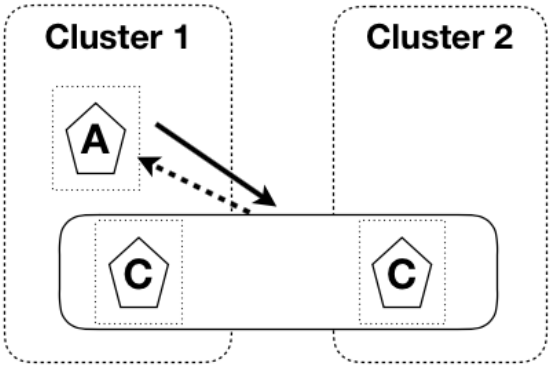


FIGURE 4.9: Performance to a shared service

- A service isolation layer is added: Service C is allowed to be accessed from service A. The performance between A to C is evaluated again for each above scenario.

4.2 Architecture

The testbeds are developed in Google Kubernetes Engine GKE. It is a managed service from Google that provides Kubernetes clusters.

This choice is based on multiple reasons :

- It allows the research to concentrate on the implementation of Cilium and Istio testbeds directly and avoid the consuming tasks of deploying a private infrastructure.
- It allows to delete and create Kubernetes clusters on demand, rather doing these tasks manually.

- The choice of GKE against other public Cloud providers is based on the maturity, performance, and number of features all better in Google GKE than Amazon EKS and Microsoft AKS. [32]. GKE provides Kubernetes clusters on demand. That service manages the operation and availability of a master VM that is the master control plane of the provisioned Kubernetes cluster. This master VM is hidden, and the worker nodes are available to the client of the GKE service.
- GKE provides out of the box the multicluster requirements for Cilium and Istio: Cilium requires IP connectivity between all nodes [25] and Istio requires direct pod-to-pod IP connectivity [33]. GKE provides the first requirement as nodes are deployed in the same VPC, and direct pod-to-pod IP connectivity across clusters is allowed with –enable-ip-alias when a cluster is created [34]. Therefore the benefit is that there is no need to configure these networking requirements for this research.

4.3 Description of the testbed

Kubernetes cluster version: 1.12.7-gke.10. Each cluster is composed of 4 nodes. The nodes are VM instances and their type is "n1-standard-2" with 2 vCPUs and 7.5 GB of memory each.

Istio requires 4 nodes minimum to operate with a machine type "n1-standard-2" in GKE. The testbeds deployed for Cilium are based on these specifications as well to follow the same configuration as Istio.

Each testbed is composed of 2 clusters that are deployed in the same region both Istio and Cilium. The reasons to choose a single region are:

- The default Cilium multicluster installation didn't work across 2 regions because the installation step include the use of internal load balancer that doesn't multiple regions. Other possibilities exist like using Cilum global services but that wasn't tested here.
- Latency across regions is not desired for the purpose of the performance tests here.

The testbeds are deployed and operated from command line using GCP cloud shell. This is a managed service from Google that provides easy access to the Kubernetes API of GKE.

Most of the testbeds deployment are scripted with Bash so they could be deleted and recreated for each test. The scripts are all shared in this dissertation and on Github [35].

All services A,B,C and D are based on a custom application. It perform the following functions: returns a hello message and indicates in which cluster it is located, contains curl so that it can be used from within the pod to test service discover to other services, contains an application to test performance to other pods. The specifications of this image are explained in 4.4.4.

4.4 Description of the installations

The version of Cilium and Istio used are respectively 1.4 and 1.1.2. They both correspond to the latest versions available at the time of the research. Note that the architecture of each solution is described in the section 2.1.

4.4.1 Installation of Cilium clustermesh

The implementation of the Cilium clustermesh implies 2 steps: installation Cilium Network plugin first within each cluster, then configuration of the required components to enable the clustermesh. Installation steps are based on the Cilium documentation: Cilium network installation [36] and Setting up Cluster Mesh [37]. Installation Script is in Appendix A.1.

4.4.2 Installation of Istio single control plane with VPN connectivity

Istio master control plane is installed in cluster-1, the remote Istio is installed in cluster-2, then the master is configured to pair with the remote. Installation steps are based on the Istio documentation: [38]. Installation script is in Appendix B.2.

4.4.3 Installation of Istio multiple control planes with GW connectivity

Istio control plane is installed in each clusters cluster-1 and cluster-2, then DNS(service discovery) is configured to allow name resolution. Installation steps are based on the Istio documentation: [39]. Installation script is in Appendix B.3

4.4.4 Installation of the Microservice

All deployments A,B,C and D are based on a custom image. The original image is ngnix 1.15.8 in which other tools were injected: curl and Vegeta. Curl is the tools used for the

service discovery tests, and Vegeta used for the performance tests. Vegeta version is 12.3 (last version at time of research), it was downloaded and copied to the docker image. The docker file used to build the image is in Appendix C.1. The microservice consists of the web service based nginx. When configured with configmap it returns a hello message. To demonstrate that service discovery and load-balancing are working across clusters, nginx and configmap are configured together. This application follows the same idea as the sample application service in [37]. Nginx acts a simple microservice listening on http port 80, and returns a Configmap message. Configmap is a Kubernetes resource that is local to the cluster. The deployments A,B,C and D return the Configmap message "Hello from Cluster-1" from cluster-1, and return the Configmap message "Hello from Cluster-2" from cluster-2. Yaml configuration files are in Appendix C.2. 2 files are shown here as an example

The configmap : res-clust1.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: res-clust1
data:
  message: "{ \"Hello from Cluster-1\"}\n"
```

The deployment C on cluster 1 : c-p.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: c
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: c
    spec:
      serviceAccountName: c-sa
      containers:
      - name: c
        image: fnature/ngnix-curl:2
        ports:
         - containerPort: 80
           name: http
        volumeMounts:
          - name: html
            mountPath: /usr/share/nginx/html/
      volumes:
        - name: html
          configMap:
            name: res-clust1
            items:
              - key: message
                path: index.html
```

4.5 Test 1 - Service Discovery with consistent service name

Deployment A is created in cluster1 only and Deployment B is created in cluster 2 only. Services configurations are explained for each section. From within pod in deployment A, it is verified that "curl b-svc" gets the reply "Hello from Cluster-2". Same types of deployment are deployed for Cilium and Istio. Configuration files are also in Appendix C.2.

4.5.1 service discovery configuration in cilium clustermesh

Deployment A, global service A and global service B are created in cluster1. Deployment B, global service A and global service B are created in cluster2. The service must be given the global annotation from Cilium in its YAML file.

An example is global annotation is c-svc-g.yaml.

```
apiVersion: v1
kind: Service
metadata:
  name: c-svc-g
  annotations:
    io.cilium/global-service: "true"
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    name: c
```

4.5.2 service discovery configuration in istio single control plane

Deployment A, service A and service B are created in cluster1. Deployment B, service A and service B are created in cluster2.

Services in Istio must be named [40]. An example of http service is b-svc-http.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: b-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
    name: http
  selector:
    name: b
```

4.5.3 service discovery configuration in Istio with multiple control planes

Deployment A and service A are created in cluster1. Deployment B and service B are created in cluster2. Service entry B is created in cluster 1. Scripts were built for this research to implement a service entry and the implementation is based on Istio guidelines [41]. The two following functions need to be run to create a service entry : get-gw-addr and add-serviceentry in Appendix: C.2.

```
function get-gw-addr () {
echo "we get the ip of  ingressgateway from both clusters"
echo "based on https://istio.io/docs/examples/multicluster/gateways/"

kubectl config use-context "gke_${proj}_${zone}_cluster-1"

export CLUSTER1_GW_ADDR=$(kubectl get svc --selector=app=istio-ingressgateway \
    -n istio-system -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')

kubectl config use-context "gke_${proj}_${zone}_cluster-2"

export CLUSTER2_GW_ADDR=$(kubectl get  svc --selector=app=istio-ingressgateway \
    -n istio-system -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')
}

function add-serviceentry () {
echo "---- we add the service entry $2 in $1"
echo "--- the function requires get-gw-addr to be called before"
echo "--- usage : add-serviceentry cluster-2 a-svc 172.255.0.15"

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster=${CLUSTER2_GW_ADDR}
        othercluster_label="cluster-2"
else
        othercluster=${CLUSTER1_GW_ADDR}
        othercluster_label="cluster-1"
fi

echo "$2-default"
echo $3
echo "$othercluster"

kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: $2-default
spec:
  hosts:
  # must be of form name.namespace.global
  - $2.default.global
  # Treat remote cluster services as part of the service mesh
  # as all clusters in the service mesh share the same root of trust.
  location: MESH_INTERNAL
  ports:
  - name: http1
    number: 80
    protocol: http
  resolution: DNS
  addresses:
  # the IP address to which httpbin.bar.global will resolve to
  # must be unique for each remote service, within a given cluster.
  # This address need not be routable. Traffic for this IP will be captured
  # by the sidecar and routed appropriately.
  - $3
  endpoints:
  # This is the routable address of the ingress gateway in cluster2 that
```

```
  # sits in front of sleep.foo service. Traffic from the sidecar will be
  # routed to this address.
  - address: $othercluster
    labels:
     cluster: $othercluster_label
    ports:
      http1: 15443 # Do not change this port value
EOF
}
```

4.6 Test 2 - Load balancing

Load balancing is tested by using curl to the shared service, example curl "c-svc". The command is run at least 10 times and it is verified that replies from cluster-1 and cluster-2 are received.

4.6.1 load balancing configuration in cilium clustermesh

Deployment A, deployment C and global service C are created in cluster 1. Deployment C and global service C are created in cluster 2. YAML files follow the same templates as in 4.5 and 4.5.1

4.6.2 load balancing configuration in istio single control plane

Deployment A, deployment C and service C are created in cluster 1. Deployment C and service C are created in cluster 2. YAML files follow the same templates as in 4.5 and 4.5.2

4.6.3 load balancing configuration in istio multiple control plane

Deployment A, deployment C and service C are created in cluster 1. Deployment C and service C are created in cluster 2. These YAML files follow the same templates as explained in 4.5 and 4.5.2. Service entry for remote service C is created in cluster 1. Service entry for remote service C is created in cluster 2.

Destinationrules and VirtualServices are two Istio abstractions that configure traffic routing. They are created in each cluster to load balance traffic to service C. Their implementation is based on Istio guidelines [42] and following functions were created for this research: setup-lb-c and setup-routing-lb-gw in Appendix B.5.

```
function setup-routing-lb-gw () {

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster_label="cluster-2"
else
        othercluster_label="cluster-1"
fi

echo "------ we create the subsets for the global address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2-global
spec:
  host: $2-svc.default.global
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      cluster: $othercluster_label
EOF

echo "------ we create the subsets for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2
spec:
  host: $2-svc.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      name: $2
EOF

echo "------ we create the routes for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: $2
spec:
  hosts:
    - $2-svc.default.svc.cluster.local
  http:
  - route:
    - destination:
        host: $2-svc.default.svc.cluster.local
        subset: $2
      weight: 50
    - destination:
        host: $2-svc.default.global
        subset: $2
      weight: 50
EOF
}

function setup-lb-c () {
#  apply the deployments manually for testbed : res-clustx, and svc in each clusters
#  then below is that c shared in cluster-1 and cluster-2

#  I want to have remote c available from both clusters
get-gw-addr
add-serviceentry "cluster-1" "c-svc" "127.255.0.23"
add-serviceentry "cluster-2" "c-svc" "123.255.0.13"
```

```
#  I want to load balance c across clusters

if [ $1 = "tcp" ]; then
        setup-routing-lb-tcp-gw "cluster-1" "c"
        setup-routing-lb-tcp-gw "cluster-2" "c"
else
        setup-routing-lb-gw "cluster-1" "c"
        setup-routing-lb-gw "cluster-2" "c"
fi
}
```

The above routing configuration applies to HTTP services. Similar configuration is applied to test load balancing for TCP services except that the configurations must refer to TCP traffic and the services must be named TCP [40]. The corresponding function is setup-routing-lb-tcp-gw in Appendix B.5.

4.7 Test 3 - Service Isolation

The required network policies are applied to conform to the tests as specified in 4.1.3

4.7.1 Service Isolation testbed configuration in cilium

The implementation of the testbeds are based on same configuration files as for service discovery and load balancing testbeds. To block all traffic to all services, each endpoint is only allowed to talk to itself only, example of yaml file is l4-b.yaml on layer 4. Then Cilium network policies are applied on their ingress. As an example, only endpoints that match name=a can access the endpoints that match name=c on layer 3, the corresponding YAML file is l3-c.yaml. To allow access to d from all endpoints on layer 7, the field **fromEndpoints** is defined as , the corresponding YAML file is l7-d.yaml. These files can be found in Appendix C.3.

Example: Layer 7 l7-c.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L7 policy to restrict application a access to only containers in application a network"
metadata:
  name: "l7-c"
spec:
  endpointSelector:
    matchLabels:
      name: c
  ingress:
  - fromEndpoints:
    - matchLabels:
            name: a
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
      rules:
        http:
```

```
      - method: "GET"
        path: "/test"
```

The testbeds were created using the script in Appendix A.2. Note that Ciliumnetworkpolicies were installed manually in each cluster to conform to the testbeds. Example : "kubectl apply -f l7-c.yaml".

4.7.2 Service Isolation configuration in Istio

Service Isolation in Istio is based on service account authentication first. For example to create and link a service account A to a deployment A, the file a-p-rbac.yaml is applied. All deployments are linked with their respective service account that way. Service isolation requires mTLS to be enforced in the service mesh for services and endpoints for authentication purpose before authorization: this is achieved by applying meshpolicy.yaml and default_destinationrule.yaml.

To block traffic to all services, RBAC is enforced on all namespace "default" with ClusterRbacConfig.yaml.

To allow specific traffic at layer 7, example service account A is authorized to access service C (egress only), a servicebinding and servicerole are applied, the corresponding example is role-http-a-c.yaml. To allow access to d from all service accounts, the field user maps to "*", the corresponding YAML file is role-http-d.yaml.

Istio allows also to specify layer 4 polices to TCP services. These rules apply only to services that are named tcp in their configuration file. The same definitions as above apply, except that each configuration references to TCP services. Corresponding examples are c-svc-tcp.yaml, role-tcp-a-c.yaml.

The YAML files can be found in Appendix C.4

Example: role-http-d.yaml

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: d
  namespace: default
spec:
  rules:
  - services: ["d-svc.default.svc.cluster.local"]
    methods: ["GET"]
---
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: all-d
  namespace: default
spec:
  subjects:
  - user: "*"
```

```
  roleRef:
    kind: ServiceRole
    name: d
```

4.7.3 Service Isolation testbed configuration in Istio single control plane

A different installation of Istio with single control plane is required in order to have mTLS enforced in the service mesh as per [38]. mTLS is required for RBAC authentication. Then same configuration as 4.7.2 can be applied. The test was not conducted, see explanations in challenge section 4.12.

4.7.4 Service Isolation testbed configuration in istio multi control planes

Because the testbeds are composed of shared services, and each cluster has its own Istio control plane, the service policies mentioned in 4.7.2 must be applied to both clusters. Both service isolation testbeds tcp(layer4) and http(layer7) entire testbeds were deployed with the following scripts setup-testbed-rbac-shared-tcp-gw, setup-testbed-rbac-shared-gw, setup-testbed-rbac-noneshared-tcp-gw, setup-testbed-rbac-noneshared-http-gw in Appendix B.6

Example:

```
function setup-testbed-rbac-shared-gw {
# this works ok

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f d-p.yaml
k apply -f a-svc-http.yaml
# we need the service b in cluster-1. Issue is that istio doesn't resolve b-svc to b-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f b-svc-http.yaml
k apply -f c-svc-http.yaml
k apply -f d-svc-http.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k apply -f res-clust2.yaml
k apply -f b-b.yaml
k apply -f c-b.yaml
k apply -f d-b.yaml
# we need the service a in cluster-2. Issue is that istio doesn't resolve a-svc to a-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f a-svc-http.yaml
k apply -f b-svc-http.yaml
k apply -f c-svc-http.yaml
k apply -f d-svc-http.yaml
```

```
# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

# the following configures the necessary service entries and routing rules for this testbed in both clusters
setup-lb-c
setup-lb-d
setup-discovery-cluster1tob
setup-discovery-cluster2toa

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml
}
```

4.7.5 Service isolation with Network plugin policy and Istio multicluster

GKE allows to use network policy in the GKE built-in network plugin Calico. The clusters are created the same way as other testbeds except that network policy is enabled at their creation by adding the parameter "enable-network-policy" [43]. Then the scripts to deploy the entire testbed is in Appendix B.7. The command "curl a-svc" is initiated from deployment pod b running in cluster 2.

Example: b-a-l3.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: a-l3
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      name: a
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: b
```

4.8 Requirement 5 - No performance overhead

The tool vegeta is chosen to test performance. This is inspired by the research on service isolation in a single cluster with Istio from [16]. The tools allows to test latency and success ratio at different rates of requests per seconds. Rather than searching the maximum performance a system can sustain, it evaluates how well a system holds different rates.

Vegeta 12.3 is embedded in all microservices A,B,C,D as explained in 4.4.4.

Vegeta is run during 30s with 64 works, from 500 to 3000 requests per seconds. Vegeta is configured to request service c. An example of implementation is in Appendix D.1.

Note the testbeds are created with no other deployments and services running other the one required for their specific Cilium and Istio multicluster environment.

Vegeta produces multiple results, the results that will be looked at are : - the latency: it is the "amount of time taken for a response to be read" [44]. The experiment will report on the mean, 50th (median), 95th percentile, and 99th percentile. - The success ratio: it "shows the percentage of requests whose responses didn't error and had status codes between 200 and 400 (non-inclusive" [44].

An example of Vegeta result is :

```
Requests      [total, rate]            30000, 999.98
Duration      [total, attack, wait]    30.00545466s, 30.000715748s, 4.738912ms
Latencies     [mean, 50, 95, 99, max]  174.086681ms, 410.881s, 1.315057589s, 5.244820385s, 5.822119514s
Bytes In      [total, mean]            702858, 23.43
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  90.11%
Status Codes  [code:count]             0:2967  200:27033
Error Set:
Get http://c-svc-g: EOF
```

4.8.1 Performance testbed configuration in cilium

The first testbed is presented here. Subsequent implement of tests are shared in Appendix D.2.

- Cilium performance test in single cluster

  This test is the baseline to compare the performance with Cilium performance across clusters.

  Deployment a.yaml and c1.yaml are created in cluster1 (cilium19 in the code). Service c-svc.yaml is created in cluster 1. The resulting kubectl code is:

```
kubectx cilium19
kubectl apply -f res-clust1.yaml
kubectl apply -f a.yaml
kubectl apply -f c1.yaml
kubectl apply -f c-svc.yaml
```

Vegeta is then run from a deployment pod a that is manually chosen to be on a different node that deployment pod c. Example is in Appendix D.1

- Cilium performance in single cluster - Baseline
- Cilium performance with shared service in single cluster - Baseline
- Cilium performance with network policy in single cluster - Baseline
- Cilium performance across clusters
- Cilium performance to a shared service across clusters
- Cilium performance with network policy across multiple clusters

4.8.2 Performance configuration in istio single control plane vpn

The first testbed is presented here. Subsequent implement of tests are shared in Appendix D.3. Note that Istio with multiple control planes wasn't tested here.

- Istio performance test in single cluster

  This test is the baseline to compare the performance with Istio performance across clusters.

  Deployment a.yaml and c-p.yaml are created in cluster1. Service c-svc-tcp.yaml is created in cluster 1. The resulting kubectl code is:

```
kx gke_francoisproject_us-east1-b_cluster-1
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f c-svc-tcp.yaml
```

  Vegeta is then run from a deployment pod a that is manually chosen to be on a different node that deployment pod c. Example is in Appendix

- Istio performance in single cluster - Baseline
- Istio performance with shared service in single cluster - Baseline
- Istio performance across clusters - Baseline
- Istio performance to a shared service across clusters

- Istio performance with service isolation in single cluster
- Istio performance with service isolation across multiple clusters

4.9 Implementation challenges

4.9.1 Installation - challenges - Cilium clustermesh

The research was blocked for a long time due to the inability to complete the Cilium clustermesh installation. This would cause all functions across clusters to fail. Issue was investigated by Cilium engineering and root cause was found to be one step missing in the Cilium documentation. A github issue was opened [45]

4.10 Test 1 - challenges - Service Discovery

For the tests, curl was used to test Service Discovery. Indeed ping of a virtual service is not supported in Kubernetes.

4.11 Challenges - Service Isolation with Istio

There were multiple issues to implement basic isolation tasks not related yet to multi-cluster operation:

- After Istio RBAC was enabled (ClusterRbacConfig.yaml in Appendix C.4), pods were crashing. Issue was related to livenessProbe and readinessProbe. Their function is to automatically restart a Container that has gone in a bad state [46]. The probe requests were denied by the Istio sidecar. These functions were removed for the Istio testbed. Cilium in comparison didn't show any issue.
- After Istio RBAC was enabled (ClusterRbacConfig.yaml in Appendix C.4) and traffic should be blocked by default, it still allowed access between services. Issue was that services port were not named. It is a requirement to name the service ports as per [40]. The reason to impose this is that Istio can't recognize automatically the type of traffic. A conclusion is that Istio can't apply layer4 policy(tcp) to a layer 7(http) service. That leads to the requirement of network policies to be managed by underlying network plugin.

- After Istio RBAC was enabled and an RBAC policy was applied, it still wouldn't allow access between services. Issue was resolved after enabling mTLS in cluster. RBAC requires mTLS to be present to provide authentication before authorizing a service account to access a service.

4.12 Test 3 - Challenges - Service Isolation and performance with Istio multicluster single control plane

There was no time to accomplish the following tests : Service Isolation testbed configuration in Istio single control plane, Istio performance with service isolation single and across clusters. As explained above, RBAC requires to have mTLS enabled. A different installation of Istio multiclurster with single control plane was required in order to have mTLS enforced in the service mesh as per [38]. Note it was possible to apply service isolation without mTLS but the configuration was different than what was originally tested here: RBAC with authentication and authorization. mTLS is required for services to be authenticated with certificates. Without authentication, servicerolebinding can be applied to attributes not depending on mTLS like source.ip [47]

4.13 Test 4 - Challenges - Performance with Istio multiple control plane

This test would also be more challenging to interpret as this configuration is based on Instio ingress gateway connectivity that is provided with external load balancer and would therefore include internet based latencies. To avoid using external load balancers, nodeports could be use [48]. There was not enough time during this research to accomplish this task.

4.14 Other Requirements

The following other requirements were not tested due to lack of time: Failover, Bursting, Stateful Application, Zero VPN.

Chapter 5

Results

5.1 Note on system requirements

Both Istio multicluster and Cilium clustermesh can be compared in terms of the differences in system requirements.

There is no minimum requirement found for the installation of Cilium in a single cluster and the clustermesh. The installation of Cilium didn't yield any issue: 2 worker nodes of "n1-standard-1 in a single cluster for a total of 2 vCPU and 7.5GB. Cilium clustermesh also run successfully with 2 vCPU and 7.5GB in each cluster.

Official minimum requirements to run Istio and Istio multicluster were not found as well. Instead, tutorials and forums suggested that Istio required 4 nodes on GKE with standard VM instances. The tests conducted in this current research show that it didn't work with amount of worker nodes less than 4, and it didn't work with 4 nodes of instances types "n1-standard-1: Pods were going to crashing state. It worked only with 4 nodes of instances types "n1-standard-2. So the total minimums by the standard installation of Istio on a single cluster appear to be 8 vCPUs and 30GB of ram [49]. If Istio multicluster with single control plane is deployed, then the second cluster only requires one node minimum and the number of nodes can be expanded automatically which is ideal for a bursting use case [50]. Therefore a multicluster operation with Istio single control plane and multiple control planes requires at minimum 8 vCPUs and 30GB of ram in one cluster at least.

Therefore it can be concluded that Istio requires more CPU and RAM resources than Cilium to deploy a multicluster operation.

5.2 Testbeds 1 to 3 results

The table 5.1 summarizes the results for service discovery, load balancing, and service isolation.

	Cilium clustermesh	Istio multicluster with Single Control plane	Istio multicluster with Multiple Control planes
Service Discovery with consistent service name	Success	Success	Requirement to manage service entries + Service name is not consistent.
Load Balancing	Success but weight was not tested.		
Service Isolation	Success	TCP layer 4 and 7 with Istio RBAC policies **not tested**.	Success for TCP layer 4 and 7 with Istio RBAC policies.
		Network plugin policies for all other types of traffic are inoperative across clusters.	

FIGURE 5.1: Testbeds 1 to 3 results

The table 5.2 summarizes the results for service isolation in comparison with the tests initially planned in 4.5.

Cilium Clustermesh	shared services scenario with layer 3 rules	Success
Cilium Clustermesh	shared services scenario with layer 4 rules	
Cilium Clustermesh	shared services scenario with layer 7 rules	
Cilium Clustermesh	none-shared services scenario with layer 3 rules	
Cilium Clustermesh	none-shared services scenario with layer 4 rules	
Cilium Clustermesh	none-shared services scenario with layer 7 rules	
Istio single control plane	shared services scenario with layer 3 rules	Not applicable
Istio single control plane	shared services scenario with layer 4 rules	Not tested
Istio single control plane	shared services scenario with layer 7 rules	
Istio single control plane	none-shared services scenario with layer 3 rules	Not applicable
Istio single control plane	none-shared services scenario with layer 4 rules	Not tested
Istio single control plane	none-shared services scenario with layer 7 rules	
Istio multiple control plane	shared services scenario with layer 3 rules	Not applicable
Istio multiple control plane	shared services scenario with layer 4 rules	Success
Istio multiple control plane	shared services scenario with layer 7 rules	
Istio multiple control plane	none-shared services scenario with layer 3 rules	Not applicable
Istio multiple control plane	none-shared services scenario with layer 4 rules	Success
Istio multiple control plane	none-shared services scenario with layer 7 rules	

FIGURE 5.2: Service Isolation testbeds

5.3 Testbed 1 - Service Discovery with consistent service name

Service Discovery were all successful except for Istio multicluster with multiple control planes where service name was not consistent.

Both cilium and Istio single control have minimal coding required: cilium requires adding the global service annotation with 2 lines of code in each global service. Istio single control plane requires no extra line of code. Both also require to create services in all clusters so that name of the remote services can be resolved.

Istio with multiple control planes is more complex to implement. It requires more coding: to recall the istio ingressgateway ips and implement the definition of the service entries and their ip address. It also requires to manage the pool of affected ip addresses manually as each ip must be different per cluster. The automation of the life cycle of these ip addresses is down to the operator. The cost of this implementation is counterbalanced with the fact that Istio multiple control planes get rid of the requirement to implement a VPN across clusters. Another benefit is that it didn't require to create services in both clusters as the name resolution is provided by the service entries.

Istio with multiple control planes had also one caveat: the service name resolution was not consistent. The Istio documentation specifies: "Note that if all of the versions of the reviews service were remote, so there is no local reviews service defined, the DNS would resolve reviews directly to reviews.default.global. In that case we could call the remote reviews service without any route rules." [42]. This was not the case during the test, error received was similar to "host not found". On the other hand, service across cluster could be accessed successfully with the fqdn "c-svc.default.global". For this research, a workaround was attempted by implementing a virtualservice that would direct request to c-svc to c-svc.default.global, see functions setup-discovery-cluster2toa and setup-routing-discovery-gw in Appendix B.4. This workaround didn't fully work with unexpected issue: it would work only if the other cluster had the same virtual service running on the other cluster. Further investigation would be required for service discovery.

5.4 Testbed 2 - Load balancing

The tests were all successful. Note that both Cilium and Istio provides the ability to change the weight of traffic but there was no time to test it (For Istio it requires virtualservice and destination rules).

Cilium and Istio single control plane required the same implementation steps as for service discovery.

For Istio multiple control planes the configuration required more coding as loadbalancing across cluster required to implement traffic management with virtualservice and destinationrules configurations. Contrary to the caveat mentioned with service discovery,

short name of service like c-svc was resolved correctly with the traffic going to deployment c in both cluster 1 and 2. That is because the virtualservice was directing traffic c-svc to both c-svc.default.global and c-svc.default.svc.cluster.local. See setup-lb-c and setup-routing-lb-gw in Appendix B.4

On the other hand, there was not enough time to test the implementation of weight with Cilium Clustermesh and Istio single control plane. Most real use-case require to implement this weight. So we can't conclude on the complexity aspect of the solutions. We can mention that Istio has a rich set for traffic management [51], that Cilium may not match. Another point to mention, not tested in this research, is that Istio provides the bursting use case by allowing to change the weight of load-balancing across clusters depending on local metrics [50]. Cilium does not provide this functionality yet.

5.5 Test 3 - Service Isolation

The tesbed service isolation in Istio multicluster with single control plane was not tested for the reasons mentioned in challenges section 4.12.

Across all testbeds, curl would hang at layer 3 and layer 4 when access to a service is not allowed. At layer 7, curl received the reply access is denied. These are expected behaviours.

In Cilium clustermesh, the tests were all successful with no issue at both layer3, 4 and 7 in both shared and none-shared services testbeds. The network policies must be created in both clusters for the shared services.

In Istio multiple control planes, the tests were all successful for the network layers where RBAC policies apply: layer 4 and 7 and TCP services only. It is more complex to implement Istio RBAC than Cilium network policies as it requires to configure a service account, a servicerole and servicerolebinding. A http network policy would supports only a http service. For TCP network policy, the service port name was changed to tcp in service configurations and the virtualservice had to be changed to tcp. These aspects are not related to the specificity of the Istio multicluster operation. They are inherent to the Istio design for the delivery of identity based authentication and authorization in a service mesh.

More generally, the security features of Istio and Cilium have fundamental differences and overlapping features. These differences are not the subject of this research but impact also what can be achieved in a multicluster environment.

Istio adds a higher layer to implement authorization and authentication betweeen services which makes it more complex. During the tests, it was not possible to apply Istio RBAC policies to pods by matching labels of deployments like in Cilium or Kubernetes network policies [52]. RBAC policies can be applied to subjects that can be service accounts (this is how the testbeds were implemented) or be found with other properties [47] (this wasn't tested).

As a network plugin, Cilium provides security policies (or firewall) from layer 3 to layer 7 for all types of traffic. Istio provides authorization at layer 4 and 7, for TCP traffic only, and brings "a rich set of attributes to base policy decisions" [53]. This shows that network plugin policies and Istio service isolation must be implemented together to provide full security to the services.

This led this research to experiment and verify that the network policies from the underlying network plugin can't be applied in a multicluster environment that is deployed by Istio only. This is the extra test presented next.

5.5.1 Network plugin policy with Istio multicluster

The results show that deployment b couldn't access service a from cluster 2 to cluster 1 (curl to service a-svc hanged). The explanation is that the network policy could not apply the policies by referencing a label (matchLabels: name: b) that was defined in cluster 2. This demonstrates that the deployment of Istio multicluster alone does not allow network plugin policies to span multiple clusters.

5.5.2 Service Isolation Conclusion

The results show that both Istio and Cilium are functional in their respective domain to achieve service isolation in multicluster scenarios. Cilium network policies and Istio RBAC functionalities can span multiple clusters successfully. However deploying an Istio multicluster does not mean that network policies will also stretch across clusters as shown in the last test. The solution for operators of Kubernetes clusters is to implement network policies based on CIDR (ipBlock field) that would allow to restrict traffic across clusters to only the desired set of services. Note that DNS can't be used as a field and the CIDR solution is not as flexible as using podSelector [54]. This is where Cilium fills the gap. This leads to consider how to deploy both Cilium and Istio in single and multicluster scenarios. In a single cluster, Istio can be installed on top of Cilium [55], and Cilium claims to enhance the running of Istio with BPF [56]. A network plugin must be able to span across clusters. Cilium resolves this issue with its Clustermesh

functionality like this research proves. Cilium provides design recommendations how to implement Istio multicluster and Clustermesh together, a possibility is to make Istio services global in the Clustermesh [31]. Further evaluation would be required in this domain.

Finally, we can also consider the pros and cons of implementing either Istio or Cilium to provide security services in multicluster operations. Istio provides authentication, authorization and encryption across services along with a key management system (public key infrastructure) to automate the key and certificate rotation at scale [57]. In comparison, Cilium aims to provide authentication and encryption transparently with X.509. It just came as a new beta feature in 1.4 but it was not yet built with a key management system (pki) [58]. One would consider then if it is still required to use Istio security features in the context of multicluster if Cilium can provide already most of the benefits and the flexibility to span underlying network policies. These recent Cilium benefits were not tested in the Clustermesh, and further research would be required in this domain. Note that Cilium published in march 2019 a presentation of use cases for their Clustermesh [31]. It was found as part of this research only at the very end.

5.6 Test 4 - No Performance Overhead

All individual results of Vegeta can be also found in the github respository [35].

Performance of multicluster with Istio and Cilium were tested in the same conditions as per the testbed specifications 4.3 with identical number of node, machine type, region, version of GKE cluster, image used to deploy the services.

Performance were only tested in Cilium clustermesh and Istio multicluster with single control plane. Istio multicluster performance with single control plane was not tested with service isolation. Istio multicluster performance with multiple control planes was not tested. The reasons are mentioned in 4.12 and 4.13..

The figure 5.3 shows the success ratios in Cilium clustermesh testbeds for: the baseline single cluster (single), multiple clusters (multi), single cluster with network policy baseline (Single&Policy), multiple cluster with network policy (Multi&Policy), single cluster with shared services baseline (Single&Shared), multiple cluster with shared services (Multi&Shared). Each testbed was tested 5 times for each rate to investigate the success ratio with more reliability. The success ratio of 1.0 corresponds to 100% of "requests whose responses didn't error and had status codes between 200 and 400" (non-inclusive" [44]. The Multi&Policy was not tested, it is an accidental omission then couldn't be tested due to lack of time in this research.

Rate	Single					Multi					Singe&Policy		Multi&Policy		Single&Shared		Multi&Shared	
500	1	1	1	1	1	1	1	1	1	0.99	1	1	1	1	1	1	1	1
1000	1	1	1	1	1	0.9	1	1	1	1	1	1	1	1	1	1	1	0.99
1500	1	1	1	1	1	1	1	1	0.89	1	0.95	0.96	0.99	0.80	1	1	1	1
2000	1	1	1	1	1	1	1	0.99	1	1	0.55	0.52	0.91	0.45	1	1	1	1
2500	1	1	1	1	1	1	1	0.87	1	1	0.47	0.39	0.49	0.27	1	1	1	1
3000	1	1	1	1	1	1	1	0.33	1	1	0.22	0.34	nt	nt	1	1	0.93	1

FIGURE 5.3: Success rates with Cilium Clustermesh testbeds

In Cilium testbeds with network policies, there is no difference between single and multicluster operation. the results shows similar trend with a success ratio starting to drop at 1500 requests/s and both gradually degrading until 3000.

In the other Cilium testbeds, single and single&shared tests all have a success ratio of 1. But Multi and Multi&Shared showed multiple occurrences of success ratio below 1 with the worst case happening for Multi with 3000 request/s. Also this happened at least once for every rate in Multi testbed, and twice at 1000 and 3000 requests/s in Multi&Shared.

Example of errors seen are described in Vegeta results:

```
Get http://c-svc-g: EOF

Get http://c-svc-g: http: server closed idle connection Get http://c-svc-g: net/http: timeout awaiting response headers

Get http://c-svc-g: dial tcp 0.0.0.0:0->10.20.10.43:80 <http://10.20.10.43:80>: i/o timeout
Get http://c-svc-g: http: server closed idle connection
Get http://c-svc-g: dial tcp: i/o timeout
```

The success rates were also evaluated in the following Istio multicluster testbeds: Single, Multi, Single&Shared, Multi&shared. The results show that the success ratio remained at 100% for all tests.

The next results present the latencies reported by Vegeta. Some of these latencies show spikes in Cilium clustermesh testbeds that were caused by the timeout issues already explained in this chapter.

The figures 5.4 and 5.5 present 2 results for Cilium and Istio in single and multicluster configurations and none-shared services (the other 3 results not shown here yield the same conclusions). Except for the timeout measures with cilium, single and multiple clusters testbeds with Istio and Cilium follow the same general trends:

- all counters mean, 50th, 95th and 99th stay in average within the same ranges of latencies and general increase
- All mean and 50th results across single and multicluster configuration stay in average below 1ms for Cilium (except timeouts) and below 2ms for Istio.

Note this would require more tests and statistics to prove any finer differences. Istio has also more latencies than Cilium and that's an expected result as Istio adds the sidecar proxy in the datapath.

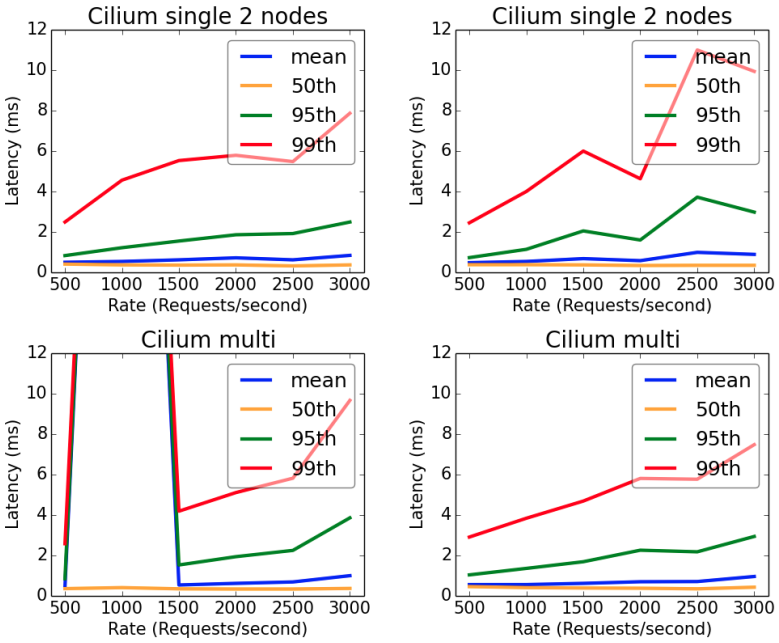


FIGURE 5.4: Performance Cilium single and multiple clusters

The figures 5.6 and 5.7 present 2 results for each following performance test in Cilium and Istio: shared service in single clusters and across clusters. The same conclusion apply as in the testbeds with none-shared service above explained above. The only difference is the average latency values are lower than with none-shared services. This is excepted as the traffic is shared amongst 2 pods instead of one.

The figure 5.8 presents the results for Cilium testbeds with network policies applied in single and across clusters.

Both single and multi cluster operations started to exhibit timeouts as rate hit 1500 requests/s. The research considers that there are not enough values to make conclusions regarding the latency measures that were not impacted by the timeouts. More tests and lower rates within the range up to 1000 requests/s would be required there.

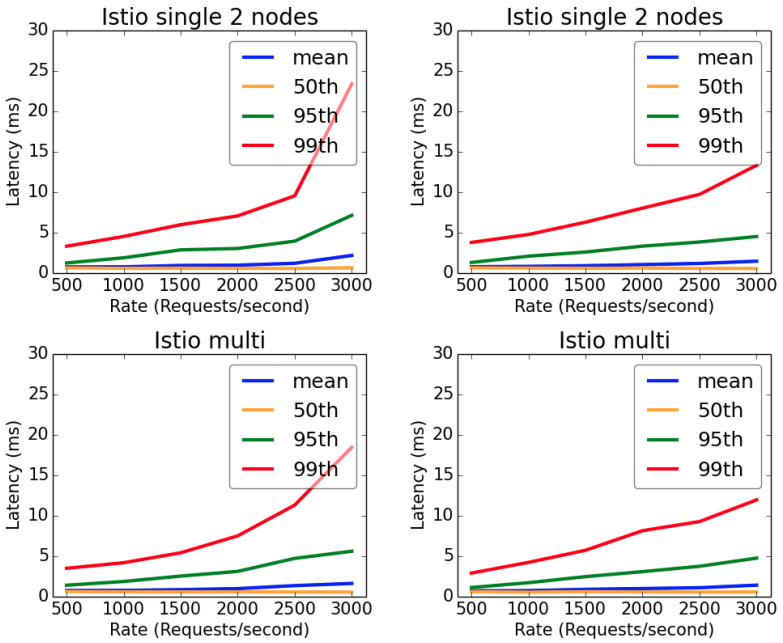


FIGURE 5.5: Performance Istio single and multiple clusters

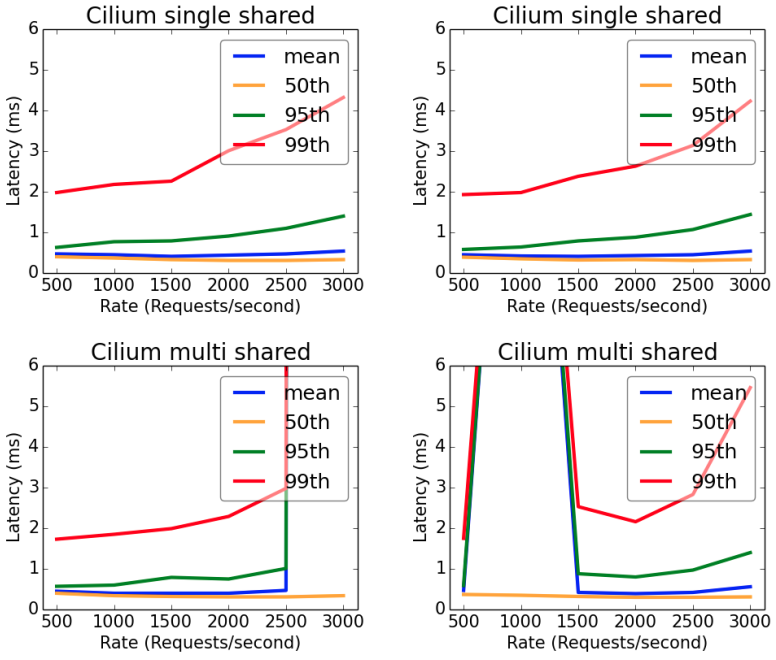


FIGURE 5.6: Performance Cilium to a shared serviced, single and multiple clusters

5.6.1 Conclusion Performance

Istio multicluster with single control plane had 100% success ratio in all the tests. Within the same conditions, Cilium clustermesh testbeds produced intermittent and unexpected timeouts while different rates of requests were sent across clusters. More investigation would be required to understand the root cause of this issue. Regarding latency, there

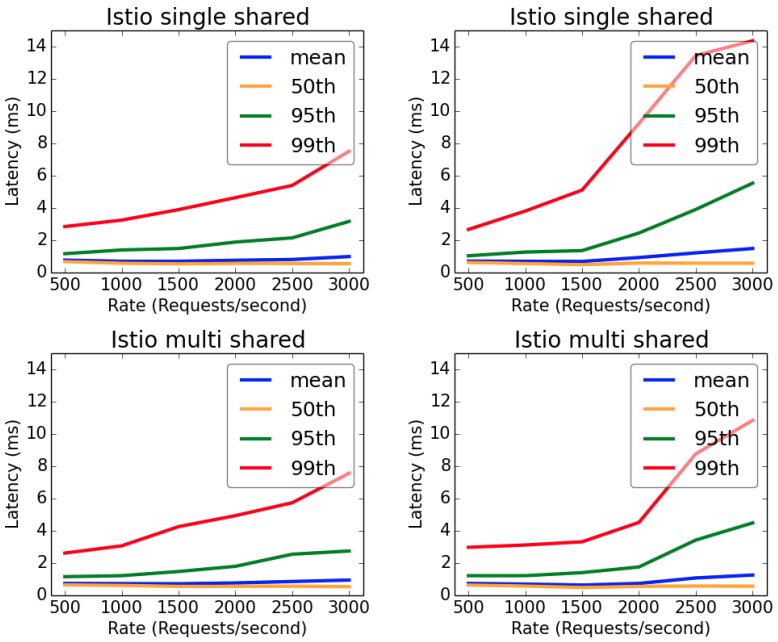


FIGURE 5.7: Performance Istio to a shared serviced, single and multiple clusters

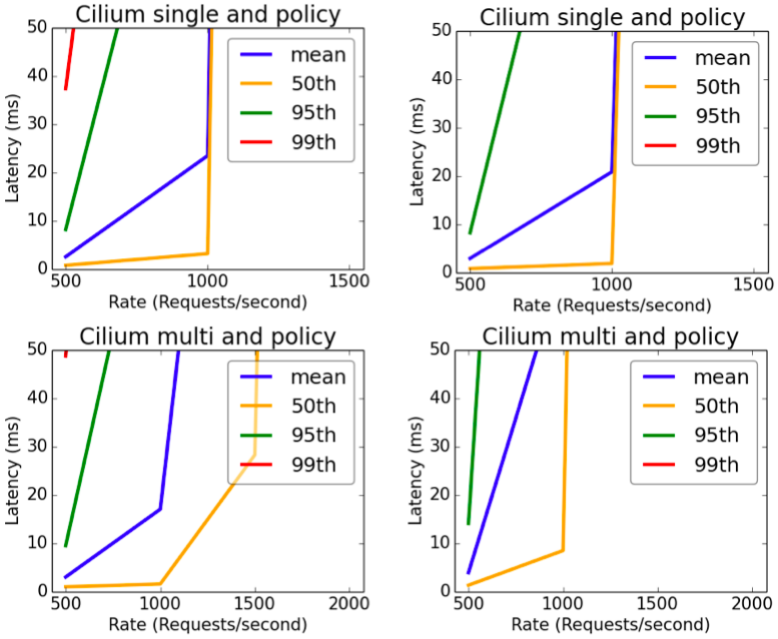


FIGURE 5.8: test

was no overhead found from Istio multicluster with single control plane. Cilium Clustermesh showed no overhead as well for the measures not impacted by the timeouts. When network policies were applied, Cilium Clustermesh exhibited similar timeouts as in a single cluster but there were not enough measures to compare the latencies.

Chapter 6

Conclusions and Future Work

Cilium and Istio are two complementary technologies. Cilium provides the necessary networking to Kubernetes containers with network policies and Istio provides the higher level of benefits of a service mesh: traffic management, observability, and application layer security. But they also overlap in several ways: both provide fine grain isolation at the application layer, encryption and authentication between services, as well as multicluster solutions that allows services to stretch across clusters while still maintaining these benefits.

There hasn't been yet an academic evaluation of their multicluster operation. The goal of this research was to evaluate and compare the multicluster solutions from by Istio and Cilium. Several requirements were found and tested: service discovery, load-balancing, service isolation and no performance overhead. The requirements were tested in: Cilium Clustermesh, Istio multicluster with Single Control plane, Istio multicluster with multiple control plane and gateway connectivity.

Several environments couldn't be tested. Single control plane with gateway connectivity was not included in this project. Service isolation and performance overhead with single control plane 4.12 and performance overhead with multiple control plane 4.13 couldn't be tested as well.

The first result of this research is that Istio provides more possibilities of implementations than Cilium. Istio can work with either single or multiple control planes. Single control planes benefits are mentioned in [59]: "single mesh scenario is most suitable in those use cases where clusters are configured together, sharing resources and typically being treated as one infrastructural component within an organization." Another benefit is that only a minimal version of Istio control plane is required in the remote cluster and doesn't require as much CPU and memory to run. In comparison, Cilium design

supports multiple control planes only. A main benefit of multiple control plane is high avaibility: if a cluster fails, one can still manage the service mesh or networking in the other cluster. Another advantage of Istio is the ability to use the ingress gateway connectivity that allows to implement a multicluster without the need to configure a VPN. This configuration is both possible with Istio single and multiple control planes. Cilium on the other hand, supports only a flat network where pods IPs must be able to reach each others which is typically resolved with a VPN.

The evaluations shows that both Istio and Cilium allows service discovery and consistent service naming in their respective designs. The only caveats found are in Istio with multiple control planes. Service naming consistency is not met out of the box and further research would be required to implement it. The requirement to implement service entries, manage a pool of remote IP addresses make it also more complex to manage as it is down to the operator to automate them [60]. This argument can be counter balanced with the fact that no VPN needs to be managed there.

The results of load-balancing evaluations shows that both Istio and Cilium succeeds in load-balancing a traffic to deployments across clusters. Another point to mention, not tested in this research, is that Istio enables the bursting use case by allowing to change the weight of load-balancing across clusters depending on local metrics [50]. Cilium does not provide this functionality yet.

On the topic of service isolation, both technologies have different benefits: Cilium provides the necessary firewall policies for all types traffics while Istio provides a role based access control only for TCP traffic. Proof-of-concepts were implemented to evaluate the service isolation features across clusters. The results show that Istio [1] and Cilium were successful to operate their respective policies across clusters. The main finding is that setting up a multicluster based on Istio alone does not allow the network policies from the underlying network plugin to span across clusters. Cilium fills this gap by allowing the network layer to stretch and provides the operators more flexibility to apply the network policies across clusters.

Further topics were then considered but not tested in this research. It was presented how Cilium and Istio are complementary and how they can be deployed together in single and multicluster scenarios. It was highlighted that Cilium overlaps with Istio with recent security features such as layer 7 authorization, transparent authentication and encryption. With these overlapping features and ability to stretch network policies across cluster, Cilium appears to provide a more comprehensive security than Istio for the multicluster use case. This could be the topic of future researches.

[1] tested on multiple control planes only

On the topic of no performance overhead, results show Istio with single cluster doesn't add latency overhead. Cilium Clustermesh presented unexpected and intermittent timeouts. No root cause yet was found and this would require to be further investigated. Future work could be to test the no performance overhead with the solutions not tested here: Istio multicluster solutions with multiple control plane and single control plane with gateway connectivity.

Finally there was not enough time to evaluate other requirements initially mentioned for this research which can then represent future works as well.

Bibliography

[1] "Survey shows kubernetes leading as orchestration platform." [Online]. Available: https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform/

[2] M. Caulfield, "Kubecon 2018 - clusters all the way down: Crazy multi-cluster topologies." [Online]. Available: https://www.cockroachlabs.com/blog/experience-report-running-across-multiple-kubernetes-clusters/

[3] "Platform9 - 5 insights from kubecon 2018." [Online]. Available: https://platform9.com/blog/enterprise-kubernetes-5-insights-from-kubecon-2018/

[4] "Mirantis kqueen." [Online]. Available: https://www.mirantis.com/blog/kqueen-open-source-multi-cloud-k8s-cluster-manager/

[5] "Cockroaclabs - experience report running across multiple kubernetes clusters." [Online]. Available: https://www.cockroachlabs.com/blog/experience-report-running-across-multiple-kubernetes-clusters/

[6] "Multiple region multi-vpc connectivity." [Online]. Available: https://aws.amazon.com/answers/networking/aws-multiple-region-multi-vpc-connectivity

[7] "Kubernetes and ipv6." [Online]. Available: https://itnext.io/kubernetes-multi-cluster-networking-made-simple-c8f26827813

[8] "Tetrate coddiwomple." [Online]. Available: https://www.tetrate.io/blog/app-routing/

[9] "Istio 1.0 future zero-vpn." [Online]. Available: https://medium.com/vescloud/battle-testing-istio-1-0-a0248ce68403

[10] "Kubernetes federation v2 github." [Online]. Available: https://github.com/kubernetes-sigs/federation-v2

[11] "Istio multicluster." [Online]. Available: https://istio.io/docs/setup/kubernetes/multicluster-install/

[12] "Cilium clustermesh." [Online]. Available: https://cilium.io/blog/2018/08/21/cilium-12/

[13] "Microservices for the enterprise, designing, developing, and deploying kasun indrasiri, prabath siriwardena."

[14] "Battle of the kubernetes service mesh." [Online]. Available: https://kubedex.com/istio-vs-linkerd-vs-linkerd2-vs-consul/

[15] "Istio on github." [Online]. Available: https://github.com/istio/istio

[16] J. Palm, "Service isolation in large microservice networks - master in computer science," 2018. [Online]. Available: http://www.nada.kth.se/~ann/exjobb/jonas_palm.pdf

[17] "Introduction to cilium." [Online]. Available: https://docs.cilium.io/en/v1.4/intro/#why-cilium

[18] "Building socket-aware bpf programs." [Online]. Available: http://vger.kernel.org/lpc_net2018_talks/lpc18-sk-lookup.pdf

[19] "Cilium architecture." [Online]. Available: http://docs.cilium.io/en/v1.4/architecture/?highlight=envoy

[20] "Evaluation of virtualization and traffic filtering methods for container networks - ukasz makowski, paola grosso." [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18302371

[21] "Accelerating linux security with ebpf iptables." [Online]. Available: https://www.researchgate.net/profile/Sebastiano_Miano/publication/326918200_Accelerating_Linux_Security_with_eBPF_iptables/links/5b870169a6fdcc5f8b710256/Accelerating-Linux-Security-with-eBPF-iptables.pdf

[22] "Connecting kubernetes clusters across cloud providers - thomas graf, covalent." [Online]. Available: https://www.youtube.com/watch?v=U34lQ8KbQow&t=869s

[23] "Benchmark results of kubernetes network plugins (cni) over 10gbit/s network." [Online]. Available: https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-10gbit-s-network-36475925a560

[24] "Cilium 1.4 benchmarking other cnis." [Online]. Available: https://cilium.io/blog/2019/02/12/cilium-14/

[25] "Cilium clustermes prerequisites." [Online]. Available: https://docs.cilium.io/en/v1.4/gettingstarted/clustermesh/#prerequisites

[26] "Cilium deepdive." [Online]. Available: https://cilium.io/blog/2019/03/12/clustermesh/

[27] "Cilium architecture." [Online]. Available: https://cilium.io/blog/2019/03/12/clustermesh/#architecture

[28] "What is istio." [Online]. Available: https://istio.io/docs/concepts/what-is-istio/

[29] "Running istio on kubernetes." [Online]. Available: https://medium.com/avitotech/running-istio-on-kubernetes-in-production-part-i-a8bbf7fec18e

[30] "Istio multicluster deployments." [Online]. Available: https://istio.io/docs/concepts/multicluster-deployments/

[31] "Cilium - relation to istio multicluster." [Online]. Available: https://cilium.io/blog/2019/03/12/clustermesh/

[32] "Gke vs aks vs eks." [Online]. Available: https://kubedex.com/google-gke-vs-microsoft-aks-vs-amazon-eks/

[33] "Istio requirement for inter-cluster pod-to-pod connectivity." [Online]. Available: https://istio.io/docs/setup/kubernetes/install/multicluster/vpn/#prerequisites

[34] "Gke inter-cluster pod-to-pod connectivity." [Online]. Available: https://istio.io/docs/examples/multicluster/gke/#create-the-gke-clusters

[35] "Github research repository - franois natur." [Online]. Available: https://github.com/fnature/gke

[36] "Cilium standard installation." [Online]. Available: https://docs.cilium.io/en/v1.4/gettingstarted/k8s-install-etcd-operator/

[37] "Cilium cluster mesh installation." [Online]. Available: https://docs.cilium.io/en/v1.4/gettingstarted/clustermesh/

[38] "Istio multicluster single control plane installation example on gke." [Online]. Available: https://istio.io/docs/examples/multicluster/gke/

[39] "Istio multicluster multiple control planes installation steps on gke." [Online]. Available: https://istio.io/docs/setup/kubernetes/install/multicluster/gateways/

[40] "Istio requirements for kubernetes." [Online]. Available: https://istio.io/docs/setup/kubernetes/prepare/requirements/

[41] "Istio gateway-connected clusters - configuration of remote services." [Online]. Available: https://istio.io/docs/examples/multicluster/gateways/

[42] "Version routing in a multicluster service mesh." [Online]. Available: https://istio.io/blog/2019/multicluster-version-routing/

[43] "Gke - creating a cluster network policy." [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy

[44] "Vegeta http load testing tool." [Online]. Available: https://github.com/tsenart/vegeta

[45] "Github cilium issue." [Online]. Available: https://github.com/cilium/cilium/issues/7558

[46] "Kubernetes - configure liveness and readiness probes." [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

[47] "Istio constraints and properties." [Online]. Available: https://istio.io/docs/reference/config/authorization/constraints-and-properties/

[48] "Istio ingress gateway and nodeports," https://istio.io/docs/tasks/traffic-management/ingress/#determining-the-ingress-ip-and-ports and https://kubernetes.io/docs/concepts/services-networking/service/#nodeport.

[49] "Resources found on istio minimum requirements," https://stackoverflow.com/questions/51503615/what-is-the-minimum-google-kubernetes-engine-cluster-size-configuration-for and https://cloud.google.com/kubernetes-engine/docs/tutorials/installing-istio.

[50] "Using istio multicluster to "burst" workloads between clusters." [Online]. Available: https://codelabs.developers.google.com/codelabs/istio-multi-burst/#4

[51] "Istio traffic management." [Online]. Available: https://istio.io/docs/concepts/traffic-management/

[52] "Kubernetes network policies." [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/

[53] "Using network policy with istio." [Online]. Available: https://istio.io/blog/2017/0.1-using-network-policy/

[54] "Github issue - networkpolicy egress with cidrselector and dnsselector." [Online]. Available: https://github.com/kubernetes/kubernetes/issues/50453

[55] "Cilium - getting started using istio." [Online]. Available: https://cilium.readthedocs.io/en/v1.4/gettingstarted/istio/

[56] "How cilium enhances istio with socket-aware bpf programs." [Online]. Available: https://cilium.io/blog/2018/08/07/istio-10-cilium/

[57] "Istio security authentication - key management system." [Online]. Available: https://istio.io/docs/concepts/security/

[58] "Cilium 1.4 transparent encryption and authentication (beta)." [Online]. Available: https://cilium.io/blog/2019/02/12/cilium-14/

[59] "Automate single mesh multi-clusters with the istio operator." [Online]. Available: https://banzaicloud.com/blog/istio-multicluster-federation-1/

[60] "Summary of istio multicluster with gateway connectivity." [Online]. Available: https://istio.io/docs/setup/kubernetes/install/multicluster/gateways/#summary

Appendix A

Cilium Code Snippets

All codes are also available on github git.

A.1 Cilium Clustermesh installation

```
#!/bin/bash
# File name is gke-clustermesh-test.sh
# The script requires kubectx to be installed first.

# Variables
export GCLOUD_PROJECT=$(gcloud config get-value project)
export GKE_ZONE="europe-west4-a"
export NAMESPACE="cilium"
export CLUSTER1="cilium19"
export CLUSTER2="cilium20"
export CLUSTERID1="19"
export CLUSTERID2="20"
export FOLDER="gke-clustermesh-test"

cd ~
rm -rf $CLUSTER1$CLUSTER2
mkdir $CLUSTER1$CLUSTER2
cd $CLUSTER1$CLUSTER2



# Creation of clusters

# gcloud container clusters create ${CLUSTER1} --username "admin" --image-type COS --num-nodes 2 --zone ${GKE_ZONE} --enable-ip-
      ↪ alias --cluster-version=1.12.6-gke.10
# gcloud container clusters create ${CLUSTER2} --username "admin" --image-type COS --num-nodes 2 --zone ${GKE_ZONE} --enable-ip-
      ↪ alias --cluster-version=1.12.6-gke.10

# Following testbed to test performance
gcloud container clusters create ${CLUSTER1} --username "admin" --image-type COS --num-nodes 4 --zone ${GKE_ZONE} --enable-ip-
      ↪ alias --cluster-version latest --machine-type "n1-standard-2"
gcloud container clusters create ${CLUSTER2} --username "admin" --image-type COS --num-nodes 4 --zone ${GKE_ZONE} --enable-ip-
      ↪ alias --cluster-version latest --machine-type "n1-standard-2"


# kubectx alias

kubectx ${CLUSTER1}=gke_${GCLOUD_PROJECT}_${GKE_ZONE}_${CLUSTER1}
kubectx ${CLUSTER2}=gke_${GCLOUD_PROJECT}_${GKE_ZONE}_${CLUSTER2}
```

```
# Cilium installation

cilium_install () {
kubectl create clusterrolebinding cluster-admin-binding \
    --clusterrole=cluster-admin \
    --user=$(gcloud config get-value core/account)

kubectl create namespace cilium
kubectl -n cilium apply -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/node-init/node-init.yaml

kubectl -n kube-system delete pod -l k8s-app=kube-dns

#kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/1.11/cilium-with-node-init.yaml
kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/1.12/cilium-with-node-init.yaml

kubectl delete pods -n kube-system $(kubectl get pods -n kube-system -o custom-columns=NAME:.metadata.name,HOSTNETWORK:.spec.
      ↪ hostNetwork --no-headers=true | grep '<none>' | awk '{ print $1 }')
}

kubectx ${CLUSTER1}
cilium_install

kubectx  ${CLUSTER2}
cilium_install


# Specify the cluster name and ID

kubectx ${CLUSTER1}
kubectl -n cilium patch cm cilium-config -p '{"data":{"cluster-name":"'$CLUSTER1'"}}'
kubectl -n cilium patch cm cilium-config -p '{"data":{"cluster-id":"'$CLUSTERID1'"}}'

kubectx  ${CLUSTER2}
kubectl -n cilium patch cm cilium-config -p '{"data":{"cluster-name":"'$CLUSTER2'"}}'
kubectl -n cilium patch cm cilium-config -p '{"data":{"cluster-id":"'$CLUSTERID2'"}}'


# Expose the Cilium etcd to other clusters

kubectx ${CLUSTER1}
kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/clustermesh/cilium-etcd-external-service
      ↪ /cilium-etcd-external-gke.yaml -n cilium

kubectx ${CLUSTER2}
kubectl apply -f https://raw.githubusercontent.com/cilium/cilium/v1.4/examples/kubernetes/clustermesh/cilium-etcd-external-service
      ↪ /cilium-etcd-external-gke.yaml -n cilium


# We wait for external IP of etcd service to come up
sleep 3m

# Extract the TLS keys and generate the etcd configuration
git clone https://github.com/cilium/clustermesh-tools.git
cd clustermesh-tools

kubectx ${CLUSTER1}
./extract-etcd-secrets.sh

kubectx ${CLUSTER2}
./extract-etcd-secrets.sh

./generate-secret-yaml.sh > clustermesh.yaml



# Ensure that the etcd service names can be resolved
./generate-name-mapping.sh > ds.patch

kubectx ${CLUSTER1}
kubectl -n cilium patch ds cilium -p "$(cat ds.patch)"

kubectx ${CLUSTER2}
kubectl -n cilium patch ds cilium -p "$(cat ds.patch)"
```

```
# Establish connections between clusters
# the cilium guide applies to default namespace only..

kubectx ${CLUSTER1}
kubectl -n cilium apply -f clustermesh.yaml

kubectx ${CLUSTER2}
kubectl -n cilium apply -f clustermesh.yaml

kubectx ${CLUSTER1}
kubectl -n cilium delete pod -l k8s-app=cilium
echo "waiting for daemon set cilium to be ready"
until [ $(kubectl -n cilium get ds cilium -o jsonpath="{.status.numberReady}") == 2 ]; do echo -n "."; sleep 1; done; echo

kubectx ${CLUSTER2}
kubectl -n cilium delete pod -l k8s-app=cilium
echo "waiting for daemon set cilium to be ready"
until [ $(kubectl -n cilium get ds cilium -o jsonpath="{.status.numberReady}") == 2 ]; do echo -n "."; sleep 1; done; echo



# missing step in doc: cilium-operator must be restarted
# "the cilium-operator deployment [...] is responsible to propagate Kubernetes services into the kvstore"
kubectx ${CLUSTER1}
kubectl -n cilium delete pod -l name=cilium-operator
echo "waiting for deployment cilium-operator to be available..."
kubectl -n cilium wait deploy/cilium-operator --for condition=available --timeout=60s

kubectx ${CLUSTER2}
kubectl -n cilium delete pod -l name=cilium-operator
echo "waiting for deployment cilium-operator to be available..."
kubectl -n cilium wait deploy/cilium-operator --for condition=available --timeout=60s
```

A.2 Cilium Clustermesh service isolation testbed

The following is a script to install deployment and services for shared and non-shared service isolation testbeds. It does not create the network policies.

```
#!/bin/bash

source myalias.sh

cluster1="${CLUSTER1:-cilium19}"
cluster2="${CLUSTER2:-cilium20}"
id1="${CLUSTERID1:-19}"
id2="${CLUSTERID2:-20}"
clusters=( $cluster1 $cluster2 )

# Error if no parameter are provided
if [ $# -eq 0 ] ; then
        echo "Error: no parameter provided"
        echo "for help type encrypt.sh --help"
        exit 2;
fi

echo "creating global services in both clusters"
for cluster in "${clusters[@]}"
do
 kubectx $cluster
 k create -f a-svc-g.yaml
 k create -f b-svc-g.yaml
 k create -f c-svc-g.yaml
 k create -f d-svc-g.yaml
done

echo "creating configmaps in both clusters"
```

```
for i in {1..2}
do
 kubectx ${clusters[$i-1]}
 k create -f res-clust$i.yaml
done


# Error if no parameter are provided
if [ $1 = "noneshared" ] ; then
 echo "noneshared"

 kubectx $cluster1
 k create -f a.yaml
 k create -f b.yaml

 kubectx $cluster2
 k create -f c2.yaml
 k create -f d2.yaml;

 else
  if [ $1 = "shared" ]; then
   echo "shared"

   kubectx $cluster1
   k create -f a.yaml

   kubectx $cluster2
   k create -f b2.yaml


   for i in {1..2}
   do
    kubectx ${clusters[i-1]}

    k create -f c$i.yaml
    k create -f d$i.yaml
   done;

  else
   echo "bad input"
   exit 2;
  fi
fi
```

Appendix B

Istio code snippets

All codes are also available on github git.

B.1 Istio shared code

Following is the code that is common to all functions used for Istio implementation.

myalias.sh

```
#!/bin/bash

function k () { kubectl "$@"; }
function k19 () { kubectx cilium19; }
function k20 () { kubectx cilium20; }
function kp () { kubectx primary; }
function kb () { kubectx burst; }
function kx () { kubectx; }

function ga () { git add *; }
function gc () { git commit -m 'CommitGKE'; }
function gp () { git push -u origin master; }
function gitsaveall () { git add * && git commit -m 'moi' && git push -u origin master; }
```

```
function clean-testbed-gw {
kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k delete deployment --all
k delete svc --all
k delete virtualservice --all
k delete destinationrule --all
k delete serviceentry --all
k delete serviceaccount --all
k delete servicerolebinding --all
k delete servicerole --all
k delete ClusterRbacConfig default

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k delete deployment --all
k delete svc --all
k delete virtualservice --all
k delete destinationrule --all
k delete serviceentry --all
k delete serviceaccount --all
k delete servicerolebinding --all
k delete servicerole --all
```

```
k delete ClusterRbacConfig default
}
```

```
export proj=$(gcloud config get-value project)
export zone="europe-west2-a"
source myalias.sh

# Following steps were followed to download Istio
# Download istio 1.1.2
# curl -LO https://github.com/istio/istio/releases/download/1.1.2/istio-1.1.2-linux.tar.gz
# tar xzf istio-1.1.2-linux.tar.gz && rm istio-1.1.2-linux.tar.gz
root="/home/ed_mitchell/myscript/istio/istio-1.1.2"
```

```
function setup-cluster () {

if [ $2 = "policy" ]; then
echo "creating cluster with network policy enabled"
gcloud container clusters create $1 --zone $zone --username "admin" \
--cluster-version latest --machine-type "n1-standard-2" --image-type "COS" --disk-size "100" \
--scopes "https://www.googleapis.com/auth/compute","https://www.googleapis.com/auth/devstorage.read_only",\
"https://www.googleapis.com/auth/logging.write","https://www.googleapis.com/auth/monitoring",\
"https://www.googleapis.com/auth/servicecontrol","https://www.googleapis.com/auth/service.management.readonly",\
"https://www.googleapis.com/auth/trace.append" \
--num-nodes "4" --network "default" --enable-cloud-logging --enable-cloud-monitoring --enable-ip-alias --enable-network-policy
else
echo "creating cluster with network policy disabled"
gcloud container clusters create $1 --zone $zone --username "admin" \
--cluster-version latest --machine-type "n1-standard-2" --image-type "COS" --disk-size "100" \
--scopes "https://www.googleapis.com/auth/compute","https://www.googleapis.com/auth/devstorage.read_only",\
"https://www.googleapis.com/auth/logging.write","https://www.googleapis.com/auth/monitoring",\
"https://www.googleapis.com/auth/servicecontrol","https://www.googleapis.com/auth/service.management.readonly",\
"https://www.googleapis.com/auth/trace.append" \
--num-nodes "4" --network "default" --enable-cloud-logging --enable-cloud-monitoring --enable-ip-alias
#  async option  while cluster-2 creation is processing didn't work....
fi

gcloud container clusters get-credentials $1 --zone $zone

kubectl config use-context "gke_${proj}_${zone}_$1"
kubectl get pods --all-namespaces
kubectl create clusterrolebinding cluster-admin-binding --clusterrole=cluster-admin --user="$(gcloud config get-value core/account
    ↪ )"
}

function setup-firewall () {
gcloud compute firewall-rules delete istio-multicluster-test-pods --quiet

function join_by { local IFS="$1"; shift; echo "$*"; }
ALL_CLUSTER_CIDRS=$(gcloud container clusters list --format='value(clusterIpv4Cidr)' | sort | uniq)
ALL_CLUSTER_CIDRS=$(join_by , $(echo "${ALL_CLUSTER_CIDRS}"))
ALL_CLUSTER_NETTAGS=$(gcloud compute instances list --format='value(tags.items.[0])' | sort | uniq)
ALL_CLUSTER_NETTAGS=$(join_by , $(echo "${ALL_CLUSTER_NETTAGS}"))
gcloud compute firewall-rules create istio-multicluster-test-pods \
  --allow=tcp,udp,icmp,esp,ah,sctp \
  --direction=INGRESS \
  --priority=900 \
  --source-ranges="${ALL_CLUSTER_CIDRS}" \
  --target-tags="${ALL_CLUSTER_NETTAGS}" --quiet
}

function setup-clusters () {
echo "usage :  setup-clusters nameofthezone"
echo "if no zone specified, default is europe-west2-a"

zone=${1:-"europe-west2-a"}
setup-cluster "cluster-1"
setup-cluster "cluster-2"

setup-firewall
}


function setup-clusters-policy () {
```

```
echo "usage :  setup-clusters nameofthezone"
echo "if no zone specified, default is europe-west2-a"

zone=${1:-"europe-west2-a"}
setup-cluster "cluster-1" "policy"
setup-cluster "cluster-2" "policy"

setup-firewall
}
```

B.2 Istio multicluster installation with single control plane

```
function setup-istio-master () {
echo "------setup istio master control plane"
echo "based on https://istio.io/docs/setup/kubernetes/install/kubernetes/#installation-steps "
cd $root
kubectl config use-context "gke_${proj}_${zone}_$1"
for i in install/kubernetes/helm/istio-init/files/crd*yaml; do kubectl apply -f $i; done
kubectl apply -f install/kubernetes/istio-demo-auth.yaml
#  I didn't have to create the namespace !!
kubectl label namespace default istio-injection=enabled
}

function setup-istio-master-vpn () {
echo "--------setup istio controle plane for multicluster vpn config"
echo "taken from https://istio.io/docs/examples/multicluster/gke/"

cd $root
kubectl config use-context "gke_${proj}_${zone}_$1"
cat install/kubernetes/helm/istio-init/files/crd-* > istio_master.yaml
helm template install/kubernetes/helm/istio --name istio --namespace istio-system >> istio_master.yaml
kubectl create ns istio-system
kubectl apply -f istio_master.yaml
kubectl label namespace default istio-injection=enabled

echo "wait before setting up the remote istio"
}

function setup-istio-remote-vpn () {

echo "--------setup remote istio vpn config"
echo "taken from https://istio.io/docs/setup/kubernetes/install/multicluster/vpn/"
echo "you must wait istio control plane to be up before applying this config"

cd $root
echo "fetching cluster-1 ips"
kubectl config use-context "gke_${proj}_${zone}_cluster-1"
export PILOT_POD_IP=$(kubectl -n istio-system get pod -l istio=pilot -o jsonpath='{.items[0].status.podIP}')
export POLICY_POD_IP=$(kubectl -n istio-system get pod -l istio=mixer -o jsonpath='{.items[0].status.podIP}')
export TELEMETRY_POD_IP=$(kubectl -n istio-system get pod -l istio-mixer-type=telemetry -o jsonpath='{.items[0].status.podIP}')

echo "pilot pod ip is ${PILOT_POD_IP}"
echo "policy pod ip is ${POLICY_POD_IP}"
echo "telemetry pod ip is ${TELEMETRY_POD_IP}"

echo " creating helm template"
 helm template install/kubernetes/helm/istio --namespace istio-system \
--name istio-remote \
--values install/kubernetes/helm/istio/values-istio-remote.yaml \
--set global.remotePilotAddress=${PILOT_POD_IP} \
--set global.remotePolicyAddress=${POLICY_POD_IP} \
--set global.remoteTelemetryAddress=${TELEMETRY_POD_IP} > $root/istio-remote.yaml


 echo "-------remote installing.."
kubectl config use-context "gke_${proj}_${zone}_cluster-2"
kubectl create ns istio-system
kubectl apply -f $root/istio-remote.yaml
kubectl label namespace default istio-injection=enabled
```

```
echo "------creating remote config files"

export WORK_DIR=$(pwd)
CLUSTER_NAME=$(kubectl config view --minify=true -o jsonpath='{.clusters[].name}')
CLUSTER_NAME="${CLUSTER_NAME##*_}"
export KUBECFG_FILE=${WORK_DIR}/${CLUSTER_NAME}
SERVER=$(kubectl config view --minify=true -o jsonpath='{.clusters[].cluster.server}')
NAMESPACE=istio-system
SERVICE_ACCOUNT=istio-multi
SECRET_NAME=$(kubectl get sa ${SERVICE_ACCOUNT} -n ${NAMESPACE} -o jsonpath='{.secrets[].name}')
CA_DATA=$(kubectl get secret ${SECRET_NAME} -n ${NAMESPACE} -o jsonpath="{.data['ca\.crt']}")
TOKEN=$(kubectl get secret ${SECRET_NAME} -n ${NAMESPACE} -o jsonpath="{.data['token']}" | base64 --decode)

cat <<EOF > ${KUBECFG_FILE}
apiVersion: v1
clusters:
   - cluster:
       certificate-authority-data: ${CA_DATA}
       server: ${SERVER}
     name: ${CLUSTER_NAME}
contexts:
   - context:
       cluster: ${CLUSTER_NAME}
       user: ${CLUSTER_NAME}
     name: ${CLUSTER_NAME}
current-context: ${CLUSTER_NAME}
kind: Config
preferences: {}
users:
   - name: ${CLUSTER_NAME}
     user:
       token: ${TOKEN}
EOF

echo "instantiating secrets on cluster-1..."
kubectl config use-context "gke_${proj}_${zone}_cluster-1"
kubectl create secret generic ${CLUSTER_NAME} --from-file ${KUBECFG_FILE} -n ${NAMESPACE}
kubectl label secret ${CLUSTER_NAME} istio/multiCluster=true -n ${NAMESPACE}
}

function setup-istio-all-vpn () {
setup-istio-master-vpn "cluster-1"
sleep 10m
setup-istio-remote-vpn
}

function setup-multicluster-vpn () {
setup-clusters
setup-istio-all-vpn
}

function setup-multicluster-vpn-policy () {
# Setup Istio testbed with network policy enabled in GKE clusters.
setup-clusters-policy
setup-istio-all-vpn
}
```

B.3 Istio multicluster installation with multiple control planes

```
function setup-istio-gw () {
echo "------setup istio control plane in 1 cluster for multicluster with gateway connectivity"
echo "based on https://istio.io/docs/setup/kubernetes/install/multicluster/gateways/"

cd $root
```

```
cat install/kubernetes/helm/istio-init/files/crd-* > istio.yaml
helm template install/kubernetes/helm/istio --name istio --namespace istio-system \
    -f install/kubernetes/helm/istio/example-values/values-istio-multicluster-gateways.yaml >> istio.yaml

kubectl config use-context "gke_${proj}_${zone}_$1"

kubectl create namespace istio-system
kubectl create secret generic cacerts -n istio-system \
    --from-file=samples/certs/ca-cert.pem \
    --from-file=samples/certs/ca-key.pem \
    --from-file=samples/certs/root-cert.pem \
    --from-file=samples/certs/cert-chain.pem

kubectl apply -f istio.yaml

kubectl label namespace default istio-injection=enabled

}

function setup-dns-gw () {

echo "creating DNS on $1"

kubectl config use-context "gke_${proj}_${zone}_$1"
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
data:
  stubDomains: |
    {"global": ["$(kubectl get svc -n istio-system istiocoredns -o jsonpath={.spec.clusterIP})"]}
EOF


}


function setup-istio-all-gw () {

echo "------setup istio control plane in both clusters with gateway connectivity"
echo "based on https://istio.io/docs/setup/kubernetes/install/multicluster/gateways/"

setup-istio-gw "cluster-1"
setup-istio-gw "cluster-2"

setup-dns-gw "cluster-1"
setup-dns-gw "cluster-2"

}

function setup-multicluster-gw () {
echo "------setup 2 gke clusters based on istio multicluster with gateway connectivity"
echo "based on https://istio.io/docs/setup/kubernetes/install/multicluster/gateways/"

setup-clusters
setup-istio-all-gw
}
```

B.4 Istio service entry and routing configuration for Istio multicluster with multiple control planes

```


function get-gw-addr () {
echo "we get the ip of  ingressgateway from both clusters"
echo "based on https://istio.io/docs/examples/multicluster/gateways/"
```

```
kubectl config use-context "gke_${proj}_${zone}_cluster-1"

export CLUSTER1_GW_ADDR=$(kubectl get svc --selector=app=istio-ingressgateway \
    -n istio-system -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')

kubectl config use-context "gke_${proj}_${zone}_cluster-2"

export CLUSTER2_GW_ADDR=$(kubectl get  svc --selector=app=istio-ingressgateway \
    -n istio-system -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')
}

function add-serviceentry () {
echo "---- we add the service entry $2 in $1"
echo "--- the function requires get-gw-addr to be called before"
echo "--- usage : add-serviceentry cluster-2 a-svc 172.255.0.15"

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster=${CLUSTER2_GW_ADDR}
        othercluster_label="cluster-2"
else
        othercluster=${CLUSTER1_GW_ADDR}
        othercluster_label="cluster-1"
fi

echo "$2-default"
echo $3
echo "$othercluster"

kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: $2-default
spec:
  hosts:
  # must be of form name.namespace.global
  - $2.default.global
  # Treat remote cluster services as part of the service mesh
  # as all clusters in the service mesh share the same root of trust.
  location: MESH_INTERNAL
  ports:
  - name: http1
    number: 80
    protocol: http
  resolution: DNS
  addresses:
  # the IP address to which httpbin.bar.global will resolve to
  # must be unique for each remote service, within a given cluster.
  # This address need not be routable. Traffic for this IP will be captured
  # by the sidecar and routed appropriately.
  - $3
  endpoints:
  # This is the routable address of the ingress gateway in cluster2 that
  # sits in front of sleep.foo service. Traffic from the sidecar will be
  # routed to this address.
  - address: $othercluster
    labels:
     cluster: $othercluster_label
    ports:
      http1: 15443 # Do not change this port value
EOF
}
```

```
function setup-routing-discovery-gw () {
# The aim is to be able to curl x-svc to remote cluster.
# Setup is not working

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster_label="cluster-2"
```

```
else
        othercluster_label="cluster-1"
fi

echo "------ we create the subsets for the global address"

kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2-global
spec:
  host: $2-svc.default.global
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      cluster: $othercluster_label
EOF

echo "------ we create the subsets for the local address"

kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2
spec:
  host: $2-svc.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      name: $2
EOF

echo "------ we create the routes for the local address"

kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: $2
spec:
  hosts:
    - $2-svc.default.svc.cluster.local
  http:
  - route:
    - destination:
        host: $2-svc.default.svc.cluster.local
        subset: $2
      weight: 0
    - destination:
        host: $2-svc.default.global
        subset: $2
      weight: 100
EOF
}

function setup-discovery-gw {
#  This setup is not working

echo "function makes remote service $1 discoverable from cluster $2 with ip $3"
echo 'example of use : setup-discovery-gw "b" "cluster-1" "172.255.0.15"'

echo "we recall the addresses of istio ingressgateways"
get-gw-addr

echo "We add the service entry in cluster $2 so DNS works.."
add-serviceentry $2 "$1-svc" $3
```

```
echo "We setup routing so that $1-svc is translated to $1.default.global"
setup-routing-discovery-gw $2 $1
}

function setup-discovery-cluster1tob () {
#  I want to have remote b available from cluster-1
get-gw-addr
add-serviceentry "cluster-1" "b-svc" "127.255.0.25"
# below not required normally
# add-serviceentry "cluster-2" "b-svc" "123.255.0.15"

# below should not be required, as x-svc should resolve to x-svc.default.global automatically..
#setup-routing-discovery-gw "cluster-1" "b"
# below not required ?
#setup-routing-discovery-gw "cluster-2" "b"
}

function setup-discovery-cluster2toa () {
#  I want to have remote a available from cluster-2
get-gw-addr
# add-serviceentry "cluster-1" "a-svc" "127.255.0.26"
add-serviceentry "cluster-2" "a-svc" "123.255.0.16"

# below should not be required, as x-svc should resolve to x-svc.default.global automatically..
# setup-routing-discovery-gw "cluster-2" "a"
}
```

B.5 Istio load balancing with multiple control planes

```
function setup-routing-lb-gw () {

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster_label="cluster-2"
else
        othercluster_label="cluster-1"
fi

echo "------ we create the subsets for the global address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2-global
spec:
  host: $2-svc.default.global
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      cluster: $othercluster_label
EOF

echo "------ we create the subsets for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2
spec:
  host: $2-svc.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
```

```
    labels:
      name: $2
EOF

echo "------ we create the routes for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: $2
spec:
  hosts:
    - $2-svc.default.svc.cluster.local
  http:
  - route:
    - destination:
        host: $2-svc.default.svc.cluster.local
        subset: $2
      weight: 50
    - destination:
        host: $2-svc.default.global
        subset: $2
      weight: 50
EOF
}

function setup-routing-lb-tcp-gw () {

kubectl config use-context "gke_${proj}_${zone}_$1"

if [ $1 = "cluster-1" ]; then
        othercluster_label="cluster-2"
else
        othercluster_label="cluster-1"
fi

echo "------ we create the subsets for the global address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2-global
spec:
  host: $2-svc.default.global
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      cluster: $othercluster_label
EOF

echo "------ we create the subsets for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: $2
spec:
  host: $2-svc.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: $2
    labels:
      name: $2
EOF

echo "------ we create the routes for the local address"
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
```

```
  name: $2
spec:
  hosts:
    - $2-svc.default.svc.cluster.local
  tcp:
  - route:
    - destination:
        host: $2-svc.default.svc.cluster.local
        subset: $2
      weight: 50
    - destination:
        host: $2-svc.default.global
        subset: $2
      weight: 50
EOF
}

function setup-lb-c () {
#  apply the deployments manually for testbed : res-clustx, and svc in each clusters
#  then below is that c shared in cluster-1 and cluster-2

#  I want to have remote c available from both clusters
get-gw-addr
add-serviceentry "cluster-1" "c-svc" "127.255.0.23"
add-serviceentry "cluster-2" "c-svc" "123.255.0.13"

#  I want to load balance c across clusters

if [ $1 = "tcp" ]; then
        setup-routing-lb-tcp-gw "cluster-1" "c"
        setup-routing-lb-tcp-gw "cluster-2" "c"
else
        setup-routing-lb-gw "cluster-1" "c"
        setup-routing-lb-gw "cluster-2" "c"
fi
}

function setup-lb-d () {
#  apply the deployments manually for testbed : res-clustx, and svc in each clusters
#  then below is that c shared in cluster-1 and cluster-2

#  I want to have remote c available from both clusters
get-gw-addr
add-serviceentry "cluster-1" "d-svc" "127.255.0.24"
add-serviceentry "cluster-2" "d-svc" "123.255.0.14"

#  I want to load balance c across clusters

if [ $1 = "tcp" ]; then
        setup-routing-lb-tcp-gw "cluster-1" "d"
        setup-routing-lb-tcp-gw "cluster-2" "d"
else
        setup-routing-lb-gw "cluster-1" "d"
        setup-routing-lb-gw "cluster-2" "d"
fi
}
```

B.6 Istio service isolation testbed with multiple control planes

```
function setup-testbed-rbac-shared-gw {
# this works ok

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f d-p.yaml
```

```
k apply -f a-svc-http.yaml
# we need the service b in cluster-1. Issue is that istio doesn't resolve b-svc to b-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f b-svc-http.yaml
k apply -f c-svc-http.yaml
k apply -f d-svc-http.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k apply -f res-clust2.yaml
k apply -f b-b.yaml
k apply -f c-b.yaml
k apply -f d-b.yaml
# we need the service a in cluster-2. Issue is that istio doesn't resolve a-svc to a-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f a-svc-http.yaml
k apply -f b-svc-http.yaml
k apply -f c-svc-http.yaml
k apply -f d-svc-http.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

# the following configures the necessary service entries and routing rules for this testbed in both clusters
setup-lb-c
setup-lb-d
setup-discovery-cluster1tob
setup-discovery-cluster2toa


kubectl config use-context "gke_${proj}_${zone}_cluster-1"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml
}

function setup-testbed-rbac-shared-tcp-gw {

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f d-p.yaml
k apply -f a-svc-tcp.yaml
# we need the service b in cluster-1. Issue is that istio doesn't resolve b-svc to b-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f b-svc-tcp.yaml
k apply -f c-svc-tcp.yaml
k apply -f d-svc-tcp.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml
```

```
kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k apply -f res-clust2.yaml
k apply -f b-b.yaml
k apply -f c-b.yaml
k apply -f d-b.yaml
# we need the service a in cluster-2. Issue is that istio doesn't resolve a-svc to a-svc.default.global
# I disable it here, as we will test by calling b-svc.default.global directly
# k apply -f a-svc-tcp.yaml
k apply -f b-svc-tcp.yaml
k apply -f c-svc-tcp.yaml
k apply -f d-svc-tcp.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

# the following configures the necessary service entries and routing rules for this testbed in both clusters
setup-lb-c "tcp"
setup-lb-d "tcp"
setup-discovery-cluster1tob
setup-discovery-cluster2toa

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-tcp-d-gw.yaml
# allows a to access c
k apply -f role-tcp-a-c-gw.yaml
# allows c to access a
k apply -f role-tcp-c-a-gw.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-tcp-d-gw.yaml
# allows a to access c
k apply -f role-tcp-a-c-gw.yaml
# allows c to access a
k apply -f role-tcp-c-a-gw.yaml
}

function setup-testbed-rbac-noneshared-tcp-gw {

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f b-p.yaml
k apply -f a-svc-tcp.yaml
k apply -f b-svc-tcp.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k apply -f res-clust2.yaml
k apply -f c-b.yaml
k apply -f d-b.yaml
k apply -f c-svc-tcp.yaml
k apply -f d-svc-tcp.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

# the following configures the necessary service entries
get-gw-addr
add-serviceentry "cluster-1" "c-svc" "127.255.0.24"
add-serviceentry "cluster-1" "d-svc" "127.255.0.25"
add-serviceentry "cluster-2" "a-svc" "123.255.0.24"
add-serviceentry "cluster-2" "b-svc" "123.255.0.25"
```

```
kubectl config use-context "gke_${proj}_${zone}_cluster-1"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-tcp-d-gw.yaml
# allows a to access c
k apply -f role-tcp-a-c-gw.yaml
# allows c to access a
k apply -f role-tcp-c-a-gw.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-tcp-d-gw.yaml
# allows a to access c
k apply -f role-tcp-a-c-gw.yaml
# allows c to access a
k apply -f role-tcp-c-a-gw.yaml
}

function setup-testbed-rbac-noneshared-http-gw {

kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f b-p.yaml
k apply -f a-svc-http.yaml
k apply -f b-svc-http.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
k apply -f res-clust2.yaml
k apply -f c-b.yaml
k apply -f d-b.yaml
k apply -f c-svc-http.yaml
k apply -f d-svc-http.yaml

# We enforce mTLS required for RBAC
k apply -f meshpolicy.yaml
k apply -f default_destinationrule.yaml

# the following configures the necessary service entries
get-gw-addr
add-serviceentry "cluster-1" "c-svc" "127.255.0.24"
add-serviceentry "cluster-1" "d-svc" "127.255.0.25"
add-serviceentry "cluster-2" "a-svc" "123.255.0.24"
add-serviceentry "cluster-2" "b-svc" "123.255.0.25"


kubectl config use-context "gke_${proj}_${zone}_cluster-1"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml

kubectl config use-context "gke_${proj}_${zone}_cluster-2"
# We enforce RBAC ( default is to block all traffic )
k apply -f ClusterRbacConfig.yaml

# allows all to access d
k apply -f role-http-d-gw.yaml
# allows a to access c
```

```
k apply -f role-http-a-c-gw.yaml
# allows c to access a
k apply -f role-http-c-a-gw.yaml
}
```

B.7 Istio multicluster single cluster and network Policy with Calico

b-l3.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: b-l3
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      name: b
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: b
```

b-a-l3.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: a-l3
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      name: a
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: b
```

testbed

```
kubectl config use-context "gke_${proj}_${zone}_cluster-1"
k apply -f res-clust1.yaml
k apply -f l3/b-a-l3.yaml
k apply -f l3/b-l3.yaml
k apply -f a-p.yaml
k apply -f a-svc.yaml
k apply -f b-svc.yaml


kubectl config use-context "gke_${proj}_${zone}_cluster-2
k apply -f res-clust1.yaml
k apply -f l3/b-a-l3.yaml
k apply -f l3/b-l3.yaml
k apply -f b-b.yaml
k apply -f a-svc.yaml
k apply -f b-svc.yaml
```

Appendix C

Definition files

All codes are also available on github git.

C.1 Dockerfile

```
FROM docker.io/nginx:1.15.8
RUN apt-get update && apt-get install -y --no-install-recommends curl
COPY vegeta .
```

C.2 Deployment and service YAML files

Below are examples of definition file for deployment A and C. This type template is also used for the other deployments B and D. Example, in cluster 1, configmap res-clust1.yaml and deployment c-p.yaml would be applied. In cluster 2, res-clust2.yaml and c-b.yaml would be applied.

res-clust1.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: res-clust1
data:
  message: "{ \"Hello from Cluster-1\"}\n"
```

res-clust2.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: res-clust1
data:
  message: "{ \"Hello from Cluster-2\"}\n"
```

Deployment A in cluster 1: a-p.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: a
spec:
  replicas: 4
  template:
    metadata:
      labels:
        name: a
    spec:
      serviceAccountName: a-sa
      containers:
      - name: a
        image: fnature/ngnix-curl:2
        ports:
         - containerPort: 80
           name: http
        volumeMounts:
          - name: html
            mountPath: /usr/share/nginx/html/
      volumes:
        - name: html
          configMap:
            name: res-clust1
            items:
              - key: message
                path: index.html
```

Deployment C in cluster 1 c-p.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: c
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: c
    spec:
      serviceAccountName: c-sa
      containers:
      - name: c
        image: fnature/ngnix-curl:2
        ports:
         - containerPort: 80
           name: http
        volumeMounts:
          - name: html
            mountPath: /usr/share/nginx/html/
      volumes:
        - name: html
          configMap:
            name: res-clust1
            items:
              - key: message
                path: index.html
```

Deployment C in cluster 2 c-b.yaml or c2.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: c
spec:
  replicas: 1
  template:
    metadata:
```

```
    labels:
      name: c
  spec:
    serviceAccountName: c-sa
    containers:
    - name: c
      image: fnature/ngnix-curl:2
      ports:
       - containerPort: 80
         name: http
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html/
    volumes:
      - name: html
        configMap:
          name: res-clust2
          items:
            - key: message
              path: index.html
```

A http service in Istio c-svc-http.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: c-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
    name: http
  selector:
    name: c
```

Global service in Cilium c-svc-g.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: c-svc-g
  annotations:
    io.cilium/global-service: "true"
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    name: c
```

C.3 Cilium Network policies YAML files

Layer 4 l4-b.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L3 policy to restrict application a access to only containers in application a network"
metadata:
  name: "l3-b"
spec:
  endpointSelector:
    matchLabels:
      name: b
  ingress:
  - fromEndpoints:
```

```
    - matchLabels:
        name: b
```

Layer 3 l3-c.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L3 policy to restrict application a access to only containers in application a network"
metadata:
  name: "l3-c"
spec:
  endpointSelector:
    matchLabels:
      name: c
  ingress:
  - fromEndpoints:
    - matchLabels:
        name: a
```

Layer 4 l4-c.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L4 policy to restrict application a access to only containers in application a network"
metadata:
  name: "l4-c"
spec:
  endpointSelector:
    matchLabels:
      name: c
  ingress:
  - fromEndpoints:
    - matchLabels:
            name: a
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
```

Layer 7 l7-c.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L7 policy to restrict application a access to only containers in application a network"
metadata:
  name: "l7-c"
spec:
  endpointSelector:
    matchLabels:
      name: c
  ingress:
  - fromEndpoints:
    - matchLabels:
            name: a
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
      rules:
        http:
        - method: "GET"
          path: "/test"
```

Layer 7 l7-d.yaml

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L7 policy to restrict application a access to only containers in application a network"
```

```
metadata:
  name: "l7-d"
spec:
  endpointSelector:
    matchLabels:
      name: d
  ingress:
  - fromEndpoints:
    - {}
    toPorts:
    - ports:
      - port: "80"
        protocol: TCP
      rules:
        http:
        - method: "GET"
          path: "/test"
```

C.4 Istio RBAC configuration YAML files

a-p-rbac.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: a-sa
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: a
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: a
    spec:
      serviceAccountName: a-sa
      containers:
      - name: a
        image: fnature/ngnix-curl:1
        ports:
         - containerPort: 80
           name: http
        volumeMounts:
          - name: html
            mountPath: /usr/share/nginx/html/
      volumes:
        - name: html
          configMap:
            name: res-clust1
            items:
              - key: message
                path: index.html
```

clusterRbacConfig.yaml

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ClusterRbacConfig
metadata:
  name: default
spec:
  mode: 'ON_WITH_INCLUSION'
  inclusion:
    namespaces: ["default"]
```

meshpolicy.yaml

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "MeshPolicy"
metadata:
  name: "default"
spec:
  peers:
  - mtls: {}
```

default_destinationrule.yaml

```
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "default"
  namespace: "istio-system"
spec:
  host: "*.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

role-http-a-c.yaml

```
 apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: c
  namespace: default
spec:
  rules:
  - services: ["c-svc.default.svc.cluster.local"]
    methods: ["GET"]
---
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: a-c
  namespace: default
spec:
  subjects:
  - user: "cluster.local/ns/default/sa/a-sa"
  roleRef:
    kind: ServiceRole
    name: c
```

role-http-d.yaml

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: d
  namespace: default
spec:
  rules:
  - services: ["d-svc.default.svc.cluster.local"]
    methods: ["GET"]
---
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: all-d
  namespace: default
spec:
  subjects:
  - user: "*"
  roleRef:
    kind: ServiceRole
    name: d
```

c-svc-tcp.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: a-svc
spec:
  type: ClusterIP
  ports:
  - port: 80
    name: tcp
  selector:
    name: a
```

role-tcp-a-c.yaml

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: c
  namespace: default
spec:
  rules:
  - services: ["c-svc.default.svc.cluster.local"]
    constraints:
    - key: "destination.port"
      values: ["80"]
---
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: a-c
  namespace: default
spec:
  subjects:
  - user: "cluster.local/ns/default/sa/a-sa"
  roleRef:
    kind: ServiceRole
    name: c
```

Appendix D

Performance testbeds

All codes are also available on github git.

D.1 Vegeta - example of performance tests

```
k exec -ti a-7f4459c5fc-864s9 bash

echo "GET http://c-svc" > targets.txt
chmod +x vegeta
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=500 -duration=30s > 500.bin
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=1000 -duration=30s > 1000.bin
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=1500 -duration=30s > 1500.bin
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=2000 -duration=30s > 2000.bin
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=2500 -duration=30s > 2500.bin
./vegeta -cpus=2 attack -targets=targets.txt -workers=64 -rate=3000 -duration=30s > 3000.bin

cat 500.bin | ./vegeta plot > plot500.html
cat 1000.bin | ./vegeta plot > plot1000.html
cat 1500.bin | ./vegeta plot > plot1500.html
cat 2000.bin | ./vegeta plot > plot2000.html
cat 2500.bin | ./vegeta plot > plot2500.html
cat 3000.bin | ./vegeta plot > plot3000.html

cat 500.bin | ./vegeta report --type text > report500.txt
cat 1000.bin | ./vegeta report --type text > report1000.txt
cat 1500.bin | ./vegeta report --type text > report1500.txt
cat 2000.bin | ./vegeta report --type text > report2000.txt
cat 2500.bin | ./vegeta report --type text > report2500.txt
cat 3000.bin | ./vegeta report --type text > report3000.txt

mkdir single-perf
mv *00* ./single-perf
exit

mkdir single-perf5

kubectl cp a-7f4459c5fc-864s9:single-perf/ ./single-perf5
cat ./single-perf5/report* > ./single-perf5/single-perf5-report.txt
gsutil -m cp -r ./single-perf5 gs://francoisnature/cilium/
```

D.2 Cilium performance testbeds

Note : In the code, kx is an alias for kubectx. It allows to change context and interact with the desired cluster name. k is an alias for kubectl.

Cilium performance in single cluster

```
kx cilium19
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c1.yaml
k apply -f c-svc.yaml
```

Cilium performance with shared service in single cluster

```
kx cilium19
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c1.yaml
k scale --replicas=2 deployment/c
k apply -f c-svc.yaml
```

Cilium performance with network policy in single cluster

```
kx cilium19
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c1.yaml
k apply -f c-svc.yaml
k apply -f l7/l7-c.yaml
```

Cilium performance across clusters

```
kx cilium19
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c-svc-g.yaml

kx cilium20
k apply -f res-clust2.yaml
k apply -f c2.yaml
k apply -f c-svc-g.yaml
```

Cilium performance to a shared service across clusters

```
kx cilium19
k apply -f res-clust1.yaml
k delete -f a.yaml
k apply -f a.yaml
k apply -f c1.yaml
k apply -f c-svc-g.yaml

kx cilium20
k apply -f res-clust2.yaml
k delete -f c2.yaml
k apply -f c2.yaml
k apply -f c-svc-g.yaml
```

Cilium performance with network policy across multiple clusters

```
kx cilium19
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c-svc-g.yaml

kx cilium20
k apply -f res-clust2.yaml
k apply -f c2.yaml
k apply -f c-svc-g.yaml
k apply -f l7/l7-c.yaml
```

D.3 Istio performance testbeds

Note : In the code, kx is an alias for kubectx. It allows to change context and interact with the desired cluster name. k is an alias for kubectl.

Istio performance in single cluster

```
kx gke_francoisproject_us-east1-b_cluster-1
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f c-svc-tcp.yaml
```

Istio performance with shared service in single cluster

```
kx gke_francoisproject_us-east1-b_cluster-1
k apply -f res-clust1.yaml
k apply -f a.yaml
k apply -f c-p.yaml
k scale --replicas=2 deployment/c
k apply -f c-svc-tcp.yaml
```

Istio performance across clusters

```
kx gke_francoisproject_us-east1-b_cluster-1
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-svc-tcp.yaml

kx gke_francoisproject_us-east1-b_cluster-2
k apply -f res-clust2.yaml
k apply -f c-b.yaml
k apply -f c-svc-tcp.yaml
```

Istio performance to a shared service across clusters

```
kx gke_francoisproject_us-east1-b_cluster-1
k apply -f res-clust1.yaml
k apply -f a-p.yaml
k apply -f c-p.yaml
k apply -f c-svc-tcp.yaml

kx gke_francoisproject_us-east1-b_cluster-2
k apply -f res-clust2.yaml
k apply -f c-b.yaml
k apply -f c-svc-tcp.yaml
```