Istio Service Mesh vs. Capsule Operator for Kubernetes Multi-tenancy

Sooraj Shajahan
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00193249@myTUDublin.ie

Introduction

The growing adoption of Kubernetes and complex deployment requirements gives rise to the need for companies to identify possible opportunities to reduce infrastructure costs, improve resiliency, and optimize operational efforts. One of the possible solutions to realizing these operational benefits is the usage of multi-tenant architectural patterns on Kubernetes. Multi-tenancy pattern in software infrastructure enables multiple users and organizations to use the common underlying infrastructure while allowing for data and resource level isolation. This research aimed to compare the efficacy between a current widely used multi-tenancy implementation pattern that uses the Istio service mesh and a similar multi-tenant implementation that uses the Capsule operator on top of Kubernetes.

The results showed that multi-tenancy implementation using only the Capsule operator did much better in terms of the latency, and requests per second (RPS) and used comparatively less Kubernetes cluster resources than the Istio-based implementation.

Motivation

The motivation for this research is to objectively compare an existing widely used multi-tenancy implementation pattern that uses the Istio service mesh and a similar implementation that uses the Capsule operator to identify, highlight, and provide recommendations to the personas in software development to decide or be informed of the considerations while making the technology choices.

Research Questions

RQ1: *What are the existing hard multi-tenancy implementation models in Kubernetes?*

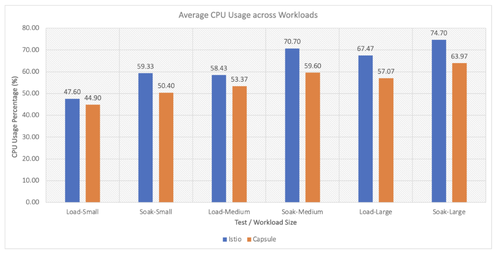
RQ2: *Do multi-tenancy implementation with Capsule perform better than the implementation using the Istio service mesh?*

RQ3: *What are the pros and cons of an implementation using the Capsule operator?*

Results

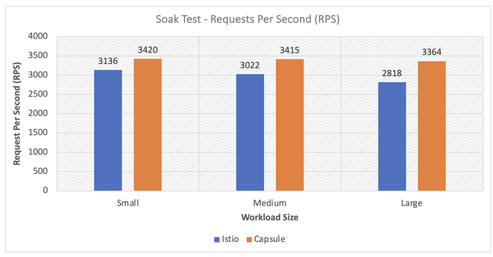
1. Capsule-based implementation was less resource intensive

It was observed that the difference in CPU usage ranges from a minimum of 3% during the load test of small workloads to a maximum of 11% during the soak test of large workloads.



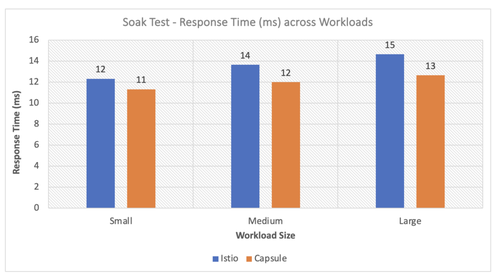
2. Capsule-based implementation performed better

The multi-tenancy setup that was based on the Capsule operator performed better than the setup that was based on Istio. The differences in RPS ranged from 284 RPS for small workloads to 545 requests for large workloads.

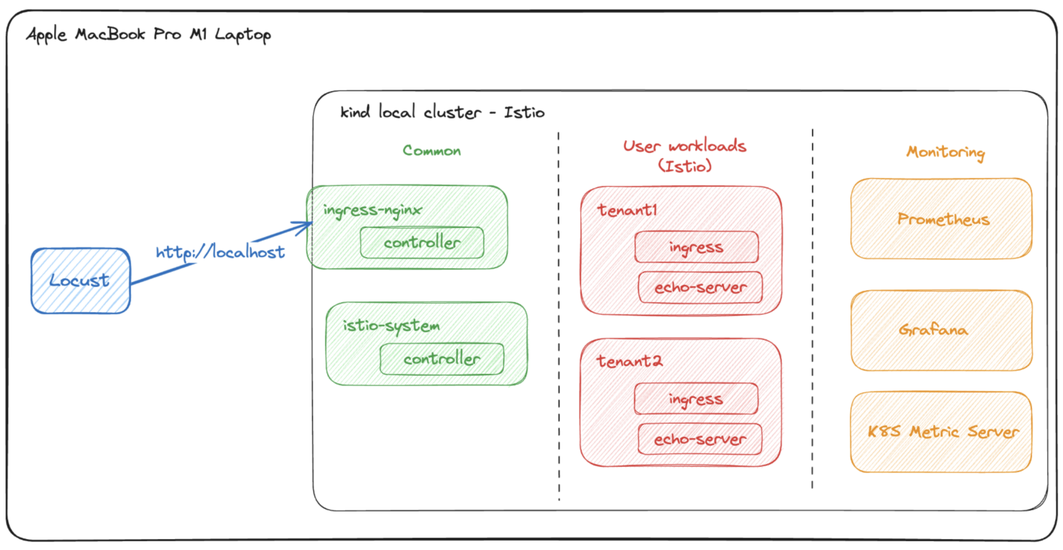


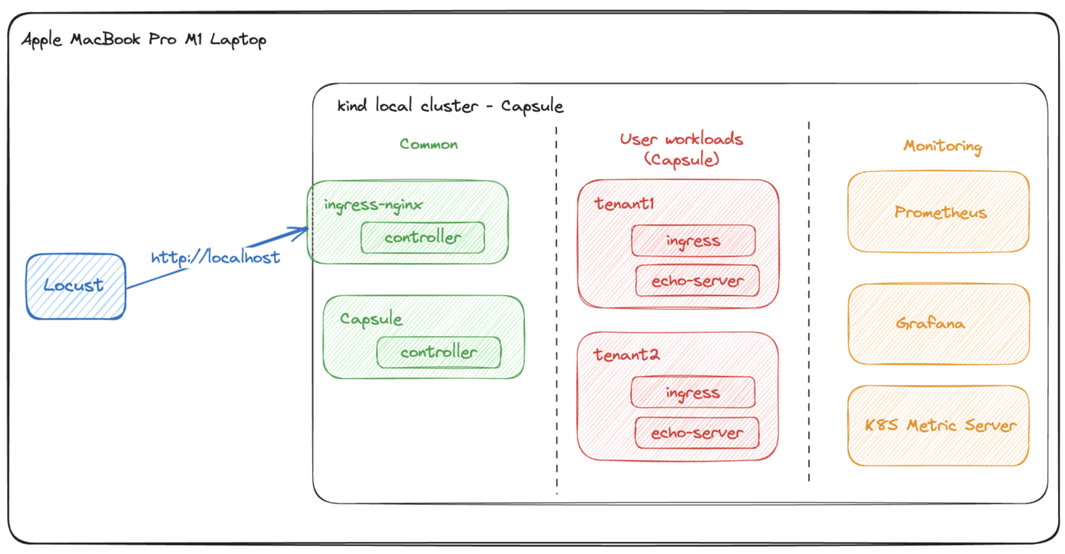
3. Response Time was better on Capsule-based implementation

The differences in response times ranged from 1 ms for small workloads to 2 ms for medium and large workloads between the two implementations.



Performance Test Setup





The above figures show the performance test setups used to compare the multi-tenancy implementations based on the Istio service mesh and Capsule operator. Python-based Locust was used as the performance testing tool. Prometheus, Grafana, and K8s Metrics Server based setup were used to capture the resource usage and performance metrics.

Conclusions and Future Work

RQ1 was answered through the literature review and different implementations one using Istio service mesh and the other using Kubernetes KCP operator were identified.

RQ2, the Capsule-based multi-tenant implementation was proven to be performing better with higher throughput and comparatively lower resource consumption than the Istio-based implementation.

RQ3, it is recommended to use ***Capsule*** as the choice of Operator for implementing multi-tenancy instead of the Istio service mesh, when some advanced features like mutual TLS (mTLS), HTTP Request Retries, and Virtual Services are not required among the micro-services.

Future work in this area could look at (i) the Ambient Mesh pattern from Istio (ii) other service-meshes like HashiCorp Consul, Linkerd, Open Shift, Nginx, etc.

QR Code for Recording