KUBERNETES AND ISTIO AS A ZERO TRUST OVERLAY

By

Collin Roach

May, 2025

Director of Thesis: Ciprian Popoviciu, Ph.D.

Major Department: Technology Systems

ABSTRACT

The emergence of Zero Trust security frameworks led to multiple solutions proposed for creating dynamic, point-to-point overlays for the endpoints of an enterprise information technology (IT) fleet. Some of these solutions reuse old technologies such as virtual private networks (VPNs) and generic route encapsulation (GRE) tunnels which add significant overhead and come with scalability constraints. On the other hand, the rapid adoption of Cloud based services led to the development of hyperscale frameworks to support the creation and maintenance of dynamic overlays. For example, Istio is a management infrastructure that supports Kubernetes with respect to end-to-end authentication, authorization and secure resource connectivity of server instances in a cloud-based application. In application platforms, this is handled by tools such as Kubernetes which orchestrates workloads between nodes; Istio is a management platform that supports Kubernetes to handle end-to-end verification and authentication for these platforms. The objective of this research is to investigate the feasibility of using Istio as an end-point authentication and authorization mechanism combined with dynamic overlay management in support of a zero-trust deployment model. This implementation would adapt Istio to distributed endpoints rather than cloud compute resources used in a micro

services application infrastructure. With Istio, traffic between endpoints was inspected at a central location where relevant policies are applied. With Istio, every endpoint was identified and verified while traffic to and from that endpoint is scrubbed and logged. Following a review of the current research on this topic, the conceptual model was presented, and the practical tests performed in support of the envisioned architecture. To test the alternative hypothesis, Istio's ability to support cloud-based endpoints outside a Kubernetes infrastructure was evaluated. Then, Istio's ability to support endpoints outside a cloud infrastructure was evaluated on devices such as a Raspberry Pi or a laptop encompassing both ARM and Intel-based processors. The impact of Kubernetes and Istio as a Zero Trust framework on intra-cluster communication was promising; however, Kubernetes and Istio experienced high latency during tests evaluating inter-cluster communications. Kubernetes and Istio can be used to effectively manage endpoint assets; however, it may not be ideal for all assets or scenarios.

Kubernetes and Istio as a Zero Trust Overlay

A Thesis

Presented to the Faculty of the Department of Technology Systems

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Network Technology

By

Collin Roach

May, 2025

Director of Thesis: Ciprian Popoviciu, Ph.D.

Thesis Committee Members:

Te-Shun Chou, Ph.D.

John Pickard, Ph.D.

Biwu Yang, Ph.D.

ACKNOWLEDGEMENTS

To my thesis advisor Dr. Ciprian Popoviciu, I thank you for your restlessness in seeing my work through to the end. I am sure that my last minute changes and ideas were not easy to deal with, yet you persisted. To my thesis committee, I am thankful to the time that you have gave me over the last few semesters in guiding and advising me. I am very grateful for all of your inputs.

To my parents Robin and Tim and my brother Devin, you have supported me for a long time through my journey. Through ups and downs, through periods of uncertainty and doubt, I am thankful that you all have continued to be my guidepost and my support system the entire way. To my best friend Sarah, experiencing college alongside you has been a blessing. Through long and often stressful nights of cramming for exams or writing papers, you have offered me your support, and I am thankful that you have been there as I'm sure I could not have done it alone.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

The importance of cybersecurity in today's information technology (IT) infrastructures has increased in correlation with the increasing criticality of IT services and data. Cybersecurity architectures evolved over time with perimeter security being the guiding principle over the past decade. Perimeter security refers to the concept of identifying domains of trust, which include resources and users governed by the same security policies. Resources and users within the perimeter are trusted and can operate unincumbered within the perimeter. The implementation of security perimeter evolved over time as well. Originally, an entire organization was enclosed within a single perimeter. Operational experience led to refinements in perimeter definitions where elements of an organization were further "fenced" in. Ultimately, the industry came to the realization that each user, each resource can be compromised and become a threat, so the perimeter concept was extrapolated to making each resource and each user a separate trust domain, each with its own perimeter. This operational model is known as Zero Trust (Rose et al., 2020). Formally defined in NIST Special Publication 800-207 (Rose et al., 2020), there are seven tenets that govern how asset access is regulated in a Zero Trust environment and the assumptions, or lack thereof, made of assets:

1. All data sources and computing services are considered resources.
2. All communication is secured regardless of network location.
3. Access to individual enterprise resources is granted on a per-session basis.
4. Access to resources is determined by dynamic policy—including the observable state of client identity, application/service, and the requesting asset—and may include other behavioral and environmental attributes.

5. The enterprise monitors and measures the integrity and security posture of all owned and associated assets.
6. All resources authentication and authorization are dynamic and strictly enforced before access is allowed.
7. The enterprise collects as much information as possible about the current state of assets, network infrastructure and communications and uses it to improve its security posture. (pp. 6-7)

These tenets ensure that no assumptions of an asset's identity or security are made, and that all asset-to-asset communication must be continuously checked and documented. In establishing a Zero Trust framework, multiple policies of focus exist. In managing hybrid cloud environments, the policies that are given a particular focus are network and service policies. These policies define what network and service communications are considered acceptable and takes enforceable action on them. This is a significant improvement from a security standpoint; however, it introduces complications related to managing this level of granularity. Various approaches to implementation involved known technologies such as Virtual Private Networks (VPN) (Varvello et al., 2021) and Generic Route Encapsulation (GRE) (Rey & Kieth, 2022). The technology reuse is addressing multiple functional requirements; however, they are plagued by significant overhead and scalability constraints. Borrowing from the experience gained in implementing and managing overlays for microservices architectures, an alternate approach can consist of an agent running on the endpoints and capturing and forwarding traffic in a hub and spoke overlay that meets the requirements of Zero Trust management. By applying this approach with technologies that already exist for container-based solutions including deployment platform Kubernetes and service mesh Istio, a Software Defined Network (SDN) like overlay can be

implemented for endpoint access control management and traffic management. Chapter 1. will continue to elaborate on this problem statement, the motivations, and constraints for the proposed solution.

1.1. Statement of the Problem

Popularity of Zero Trust has grown as more organizations move to modernize their approach to access control; however, this can be difficult for some organizations that lack the resources or experience necessary implement Zero Trust in their environment. Using existing and open-source technologies to provide Zero Trust capabilities will increase Zero Trust adoption and enable more secure IT infrastructure.

1.2. Statement of the Purpose

The purpose of the study is three-fold: (1) Identify a cloud based mechanism for implementing a Zero Trust architecture, (2) Create a Zero Trust infrastructure that does not place significant resources limitations on behalf of the serviced hosts, and (3) Investigate the breadth of scale of a cloud hosted Zero Trust architecture in accommodating geographically distributed endpoints.

1.3. Research Objectives

The objective of the study is to apply cloud hosted overlay orchestrators and endpoint security beyond the footprint of cloud infrastructures. Using a hybrid dumbbell topology supports the isolation of each endpoint, in the cloud or on premise, within its own trust domain. Additionally, the study defines an architecture for such a Zero Trust service including the integration of security services augmenting the end-point isolation. Defining a hybrid Zero Trust architecture will examine the viability of incorporating the Kubernetes and Istio technologies into endpoint device and network management.

1.4. Research Questions and Hypothesis

RQ1: Can the Kubernetes and Istio frameworks be used to create dynamic overlays for distributed, cloud based and remote endpoints in support of a Zero Trust security model?

H_0: Kubernetes and Istio will have no effect to Zero Trust security for hybrid endpoint architectures. Policy enforcement success rate of a Kubernetes and Istio overlay is less than 95%.

H_a: Kubernetes and Istio are capable of supporting Zero Trust security modes in multiple architectures: endpoint to endpoint, endpoint to cloud, and cloud to cloud. Policy enforcement success rate of a Kubernetes and Istio overlay is greater than 95%.

RQ2: Is a Kubernetes and Istio overlay an efficient method of enforcing a Zero Trust security mode on endpoints?

H_0: Kubernetes and Istio overlays for endpoint security are either inoperable or possess network latencies that exceed 200ms in round-trip travel.

H_a: Kubernetes and Istio overlays have successful policy enforcement and possess network latencies that are less than 200ms in round-trip travel.

The determinations made for the 200ms threshold of RQ2 was based on human noticeable thresholds for recognizing latency in Voice over Internet Protocol communications (Jiang et al., 2019) as well as stricter thresholds for industry specific applications such as finance and healthcare where emphasis is placed on real time information access.

1.5. Significance of the Study

The result of this study offers a Zero Trust service alternative that is based on opensource software that could accelerate adoption of Zero Trust implementations.

1.6. Limitations

The following limitations were set in pursuit of this research: endpoint operating systems (OSes) were limited to MacOS and Linux due to resources available at the time of experimentation, cloud-hosted assets were limited to the Google Cloud Platform (GCP) due to resources available during experimentation, and a single-cluster Kubernetes architecture was used to limit initial constraints of the software overlay to that representing a single network.

CHAPTER 2. REVIEW OF LITERATURE

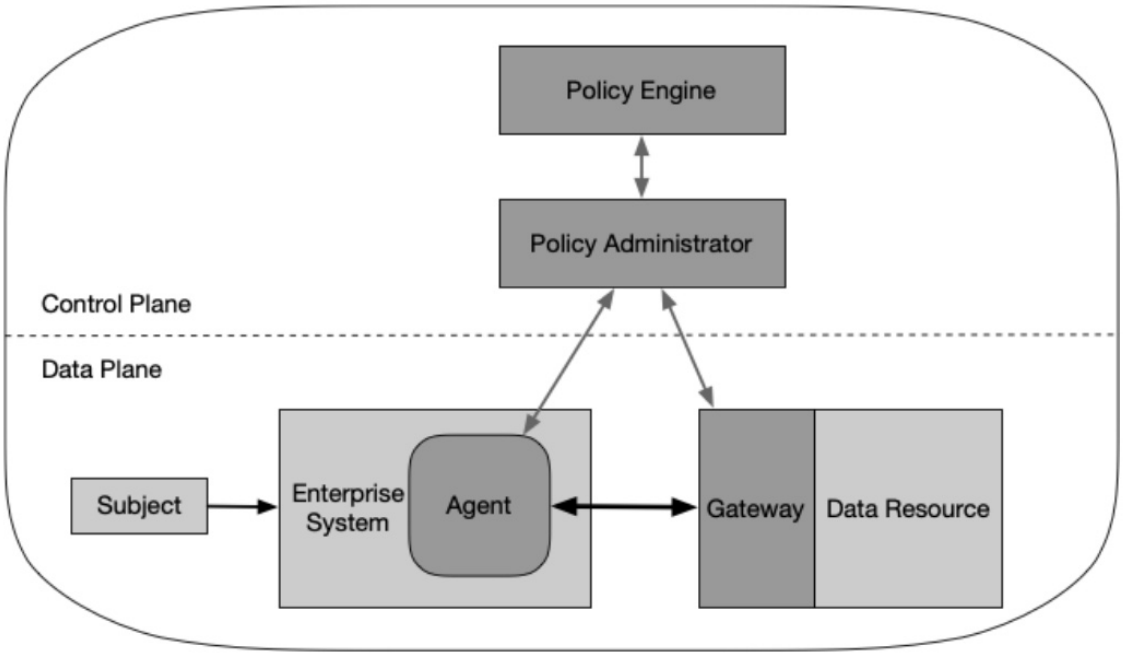
Access control architectures and decision making can be frequently complex to design and implement. In addition, adding hybrid cloud infrastructure to the design will only make it even more so. Although the literature is limited in examining the hybrid cloud infrastructure implementation of Zero Trust, this review focuses on Zero Trust and several types of Zero Trust architectures. This chapter includes a review of similar work regarding Zero Trust security, Zero Trust container architectures, and cloud implementations of Zero Trust frameworks.

2.1. Zero Trust Security

In the early 2010s, the need for a new approach to access control began to take form leading to an entire shift in its approach: Zero Trust. One of its first champions, Google, implemented this in a practical form with BeyondCorp where access is governed exclusively with user and device credentials (Ward & Beyer, 2014). Formalized in *NIST SP 800-207 Zero Trust Architecture*, Rose et al. (2020) also introduces several new technologies that are essential for Zero Trust architecture: a policy engine, a policy administrator, and a policy enforcement point. Combined, these three technologies will enable an untrusted system to be verified and access a trusted enterprise resource by making a series of requests and calls (Brazhuk & Fernandez, 2023). These technologies and methods to verify untrusted systems can take many different forms in architecture. Of particular relevance to this study is the device agent/gateway model (Rose et al., 2020) pictured in Figure 1. It shows how an agent present on a system can act as a proxy to a policy administrator and policy engine to verify a request before it is passed onto a gateway and later a resource.

Figure 1

Device Agent/Gateway Model



Note. From "Zero Trust Architecture," S. Rose et al., 2020, *Zero Trust Architecture, 800-207*(800-207), p. 14. (https://doi.org/10.6028/nist.sp.800-207). In the public domain.

The use of an agent is a common solution to implementing Zero Trust architectures as it serves as an effective funnel of all communications to the policy administrator for decisions to be made. This agent funnel is a key element in what makes this architecture effective in regulating access; although, it is not the only one as this interception system also relies on real-time access and continuous trust determinations made by the policy administrator (Yao et al., 2021). Through a series of mandatory identification of assets, verification of requests, and logging decisions, Zero Trust has become the best successor to access control frameworks.

2.2. Zero Trust Endpoint Architectures

In endpoints, the agent/gateway is one of the predominant forms of Zero Trust architecture especially when accounting for operational models such as Bring Your Own Device (BYOD) and

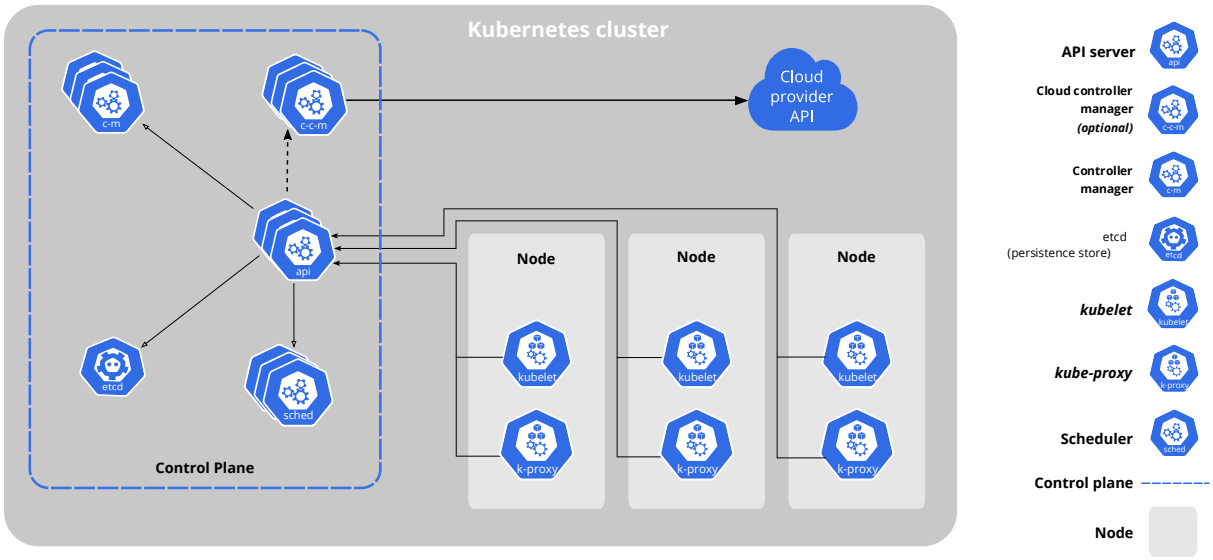
Work From Home (WFH) (Brazhuk & Fernandez, 2023); however, these models centralize around the idea of aggregating these communications to a Policy Enforcement Point that has a physical presence such as an office or server location. This is not ideal for companies or organizations that lack a physical presence or staff to host and manage the on-premises infrastructure necessary for such a solution. Agent based technology can be very robust and enable different types of hardware and software forms (Bicakci et al., 2021) and integrated roles including authentication, identity management, and situational awareness (Shen & Shen, 2024). Beyond being just a tool for aggregating communications, agents can also integrate supplementary agents for additional purposes such as authentication or they can become individual policy enforcement points (Shen & Shen, 2024).

2.3. Zero Trust Container Architectures

Containers are minified resources commonly used in cloud-hosted applications to support specific functionalities: microservices. By being cloud-hosted, container architectures can take many forms including hybrid and multi-cloud while spanning large distances from regional areas to the entire globe (Berenberg & Calder, 2022). The largest benefit of container technology is its ability to scale quickly; however, its emphasis on having only the minimum dependencies for operation, it requires a deployment management service to implement to orchestrate functions outside of its direct purpose such as network and user management. For many applications, this management tool is Kubernetes, an opensource container management platform that enables the management of services identified as nodes and can scale the workload known as pods for every node. Another one of Kubernetes benefits is a result of being opensource; it can be deployed locally and is cloud agnostic (Pizzato et al., 2024) available in GCP, AWS, Azure, and more.

Figure 2

The components of a Kubernetes cluster



Note. From *Kubernetes Components*, by The Kubernetes Authors, 2022. (https://kubernetes.io/docs/concepts/overview/components/). CC-BY 4.0.

Figure 2 shows a common example of how a Kubernetes cluster may look in splitting assets into the control plane and the data plane. The former serving as an orchestration tool to manage kubernetes services including API servers, proxies, and controllers; the latter serves workloads to run tasks that can be deployed at scale. Kubernetes splits its resources into two roles: 1) the master node, a managing node that implements the Istio overlay and interfaces with the data plane, and 2) one or more worker nodes, which exist in the data plane and serve pods. This is not without challenges as managing trust across services which cast a large attack area is a commonly mentioned risk amongst practitioners (Billawa et al., 2022). These challenges can be addressed with existing technologies.

In Zero Trust container architectures, the primary concern is regulating how these containers and services can interact with each other. Kubernetes primary function is to handle instantiation

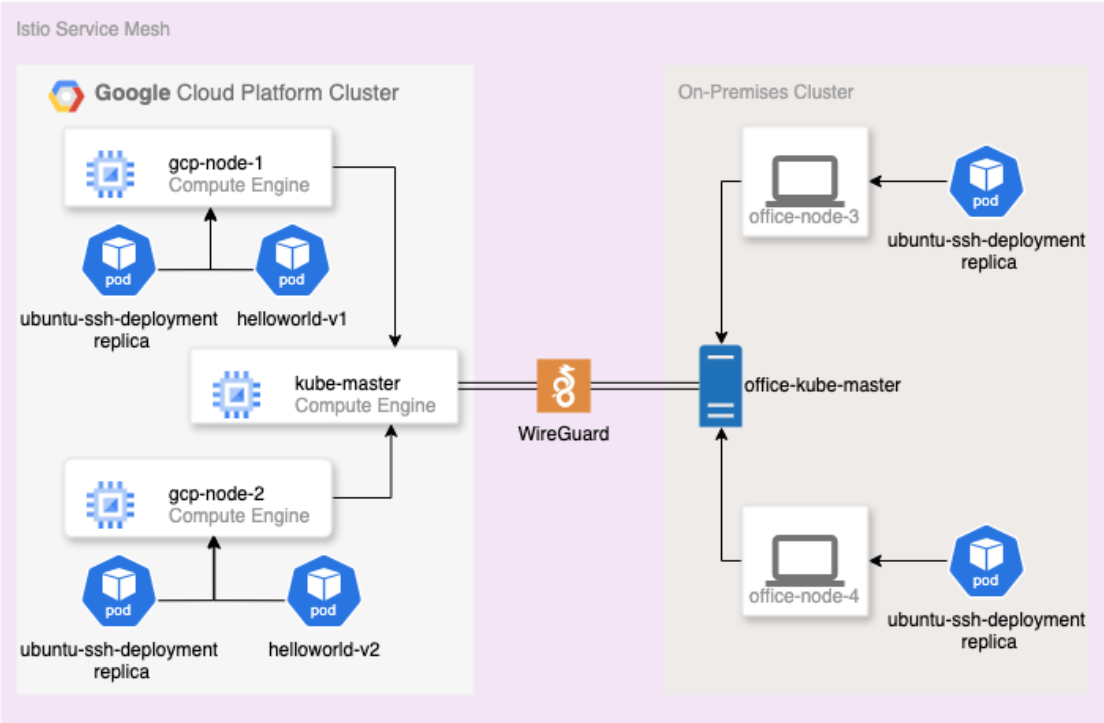
and reclamation of compute resources and to support automatic scaling, but it is not built to explicitly secure infrastructure, functionality that can be implemented through additional frameworks layered on top of Kubernetes. By layering functions better suited for authentication, authorization and traffic forwarding, Kubernetes hosted containers can implement the Zero Trust tenets (D'Silva & Ambawade, 2021; Goel & Thangaraju, 2022). Istio is one such framework, a point-to-point communication and authentication tool that can be used to handle interactions between hosted Kubernetes nodes. Istio is an implementation of a service mesh where node agents and sidecars can be used to abstract the microservices away from the physical data plane and use the node agent or sidecar to dynamically create a hub and spoke overlay (Maia & Correia, 2023; Arora & Hastings, 2024). In service mesh architectures, sidecars are an inter-container technology that serves as an out-of-process proxy sitting on top of services without modifying them (Sheikh et al., 2018; Liu et al., 2025). Since service meshes exist as interfaces of node agents to the control plane, their close proximity to the containers enables them to handle trust evaluation in real time (Alboqmi et al., 2023; Liu et al., 2023; Kurpad et al., 2023) serving to satisfy the third and fourth tenets of Zero Trust: per-session evaluation and dynamic policy-based access. The speed and efficiency of Kubernetes and Istio service meshes is essential to accommodate the rapid scaling needs of container-based architecture (Nathaniel et al., 2023; Singh et al., 2024). Further, this efficiency in upholding Zero Trust principles can be seen across multi-cloud environments where Kubernetes cloud agnostic strong suits come into play to support multi-cluster security (Rodigari et al., 2021). Zero Trust is growing as the predominant framework for enforcing access control. This is apparent as efforts have grown to accommodate this framework in architectures that scale endpoints and container architectures and eventually both.

CHAPTER 3. EXPERIMENTAL METHODOLOGY

The aim of this study is to evaluate Kubernetes and Istio as an open-source alternative implementation of a Zero Trust architecture for cross-cloud and on premise, remote endpoints. The method to explore and test the proposed solution is experimentation. A cloud hosted Kubernetes and Istio infrastructure was created, and a series of scenarios were tested that include various types of endpoints managed by the cluster as assets. Chapter 3. Methodology includes research questions and hypothesis; research design; data collection; and data analysis.

Figure 3

Hybrid approach to a Zero Trust dumbbell network topology



3.1. Research Design

Figure 3 shows a completed research design after clusters have been successfully established in the cloud and in the on-premises. Additionally, it shows a representation of the east-west

gateway after the two clusters have been established under one service mesh. The research design of this study is as follows:

1. The endpoint types tested were laptops, represented by three Intel-based machines from a variety of manufacturers, and Desktops, represented by two ARM-based Raspberry Pi 4s. The memory resources of these machines vary from 8Gb for worker nodes to 16Gb for the master node. In Figure 3, the three Intel machines are listed as office-kube-master, office-node-3, and office-node-4, respectively.
2. The cloud provider GCP was selected as the host platform for the Kubernetes control plane and the Istio service mesh. The GCP Compute platform was chosen over Google Kubernetes Engine (GKE) to offer more options for dynamically adding worker nodes external to GCP's cloud infrastructure.
3. The Kubeadm Kubernetes utility suite was selected for its ability to dynamically enroll endpoints into a Kubernetes cluster via a cli interface without stopping the existing Kubernetes cluster deployment.
4. Supplementary packages Kiali, Prometheus, Grafana, and Calico were selected for data visualization and Container Network Interface (CNI) management of the Kubernetes cluster. Kiali is a data visualization platform that was designed as a unique integration of the Istio platform (Kiali, 2024). Prometheus and Grafana are dependent packages of Kiali and must be present for successful Kiali visualization.
5. Configuration files were authored for the Worker nodes and the Master nodes on a local platform.
6. Connectivity within the local network was verified to support endpoints.

7. The Master node was migrated into the cloud provider GCP. In Figure 3, the cloud master node is identified as kube-master.
8. Two Ubuntu 24.10 VM were created as cloud hosted Worker nodes. In Figure 3, the cloud worker nodes are identified as gcp-node-1 and gcp-node-2, respectively.
9. Connectivity from cloud-to-cloud, endpoint-to-cloud, and endpoint-to-endpoint was tested.
10. An Istio AuthorizationPolicy was implemented on the Kubernetes control plane to act as a policy enforcer for determining endpoint and cloud resource access. These policies can regulate traffic and services to only execute certain procedures (Istio Authors, 2025). Restrictions in origin include namespace, IP, and principal. Mutual TLS is implemented in connections between nodes and services.
11. Policy rules were written to verify allowed and denied examples of endpoint access.

3.2. Data Collection

The methodology for the data collection is as follows:

1. Endpoint requests via hypertext transfer protocol (HTTP) GET and SSH connections were initiated to test the correct policy enforcement. The requests were exchanged between cloud and endpoints, and between pairs of endpoints.
2. Network connectivity response times between endpoints was measured via Traceroute to assess the efficiency of the cloud-based hub and spoke architecture. The round-trip response times for the tool were measured in sets of 10 packets initiated in both directions; the total amount of connections for an asset pair is 20 requests.
3. Response times of endpoint requests were collected using the time command in conjunction with HTTP GET requests and SSH connection requests. The requests were

conducted from cloud-to-cloud assets, cloud-to-endpoint assets, and endpoint-to-endpoint assets. Only one type of request should be used for an asset pair. For HTTP type requests, the test is one directional and consists of 20 requests. For SSH type requests, the test is bi-directional and consists of 10 requests from each endpoint, 20 total.

3.3. Data Analysis

The methodology for the data analysis of this study is as follows:

1. A t test of hypothesis for the mean was performed to find if there is a significant difference in the success rate of resource access requests against the baseline of 95% policy compliance.
2. Hypothesis testing was performed from the average success rate of resource access requests t test to determine if the success rate is above the threshold. Criteria for RQ1 hypothesis rejection:

   reject H_0 if $T_{STAT} < x$

   otherwise, do not reject H_0.
3. A t test of hypothesis for the mean was performed for network connectivity response times from data collected from the traceroute tool.
4. Hypothesis testing was performed from the network connectivity t test to determine if the average response time is below the threshold. Criteria for RQ2 hypothesis rejection:

   reject H_0 if $T_{STAT} < x$

   otherwise, do not reject H_0.
5. A t test of hypothesis for the mean was performed for the response times of resource requests from data collected from using the time command on HTTP and SSH resource requests.

6. Hypothesis testing was performed from the resource requests t test to determine if average resource response time is below the threshold. Criteria for RQ2 hypothesis rejection:

   reject H_0 if $T_{STAT} < x$

   otherwise, do not reject H_0.

3.4. Environment Setup

To set up the assets required for the proposed architecture, a procedural script was developed to install the necessary packages to implement the core node structure. All assets, irrespective of being a master or worker, must implement this core script (see Appendix A). This script begins by identifying the system architecture and optionally manages the Linux control groups to allocate memory management for future processes. Next, it downloads and installs the Docker and cri-docker packages which install the Docker container engine and an adapter that allows interfacing the Kubernetes Container Runtime Interface (CRI) with Docker, respectively. Next, we lock the memory swap functionality to off as recommended by Kubernetes to avoid erratic behavior when Kubernetes is managing system memory. Finally, the script installs the packages for kubelet, kubeadm, and kubectl and holds them at a locked version. The packages kubelet, kubeadm, and kubectl, serve as the core Kubernetes service, a command line interface (CLI) tool to administer enrollment in a Kubernetes cluster, and a CLI interface to manage nodes, pods, and other resources within a cluster, respectively.

For master nodes, the Kubernetes cluster initialization command must be run, and additional steps must be taken with an additional script (See Appendix B). The initialization command, seen in Figure 4, reads from a yaml file that defines the core attributes of the Kubernetes cluster

including its node name, the pod subnet it will use, the registration token, the control plane endpoint it will broadcast from, and any Subject Alternative Names (SAN) that will be accepted.

Figure 4

Kuberenetes Initialization Command

```
kube-master:~$ sudo kubeadm init --config kubeadm-config.yaml
```

Following the creation of the initialization of the Kubernetes cluster, a token and the Certificate Authority cert hash of the master node was provided to use to register worker nodes to the cluster. The command used in this study, shown in Figure 5, is a modified version that also specifies the node name and the CRI socket to use. Additionally, modifications to the master nodes routing tables had to be made as both needed to forward traffic on behalf of their worker nodes to access resources in remote clusters.

Figure 5

Kubernetes Cluster Join Command

```
sudo kubeadm join 35.232.92.23:6443 --token $TOKEN \
  --discovery-token-ca-cert-hash sha256:$HASH \
  --cri-socket=unix:///var/run/cri-dockerd.sock \
  --node-name $NODENAME
```

Note. TOKEN and HASH are variables omitted from the figure for being cluster access secrets. These should be replaced with the values generated from running the kubernetes initialization command in Figure 4.

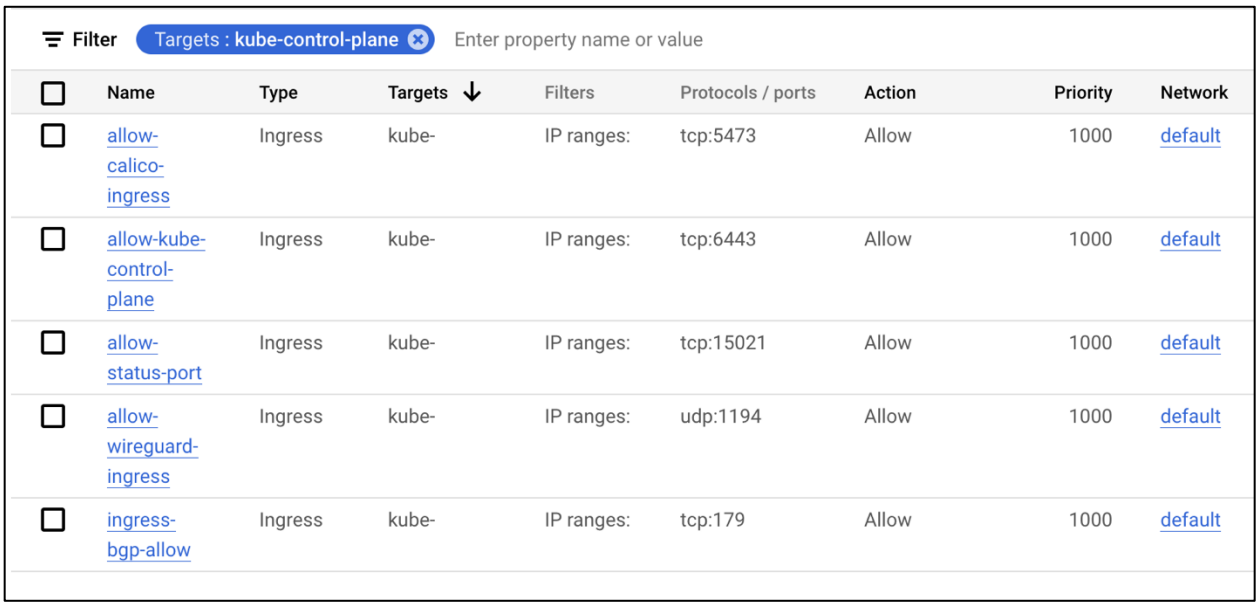
Once workers have been joined to the Kubernetes cluster, the rest of the master node set up script can be complete. This script runs through a couple steps including kubectl config file management, the download and installation of the Calico CNI and CLI, and the tainting of the master node. In Kubernetes, tainting can be considered the equivalent of assigning a key-value

pair as a tag. In this script, it is used to mark the node as a node on the control plane; however, it can be used in other areas to assign behaviors including what nodes can and cannot run certain workloads.

With the master and worker node setup process established, environment specific setups differ slightly in their completion as well. In the cloud using GCP, nodes were implemented as instances of Google Compute Engine (GCE) virtual machines (VMs). This was chosen as GCE VMs offer more visibility and control over how a cluster is created and operated than a default GKE cluster can provide. To setup the nodes in GCE VMs, the prerequisites included verifying that a Virtual Private Cloud (VPC) network is available and that it permits ingress access on the ports that the master node has access to. Thankfully, the default VPC provided by GCP is sufficient for subnetting our cloud environment. By default, GCP subnets the 10.128.0.0 network with a Classless Inter-Domain Routing (CIDR) value of 20 with each subnet representing a different geographical region that GCP has a presence in. The Firewall will however require some additional rules to permit remote traffic.

Figure 6

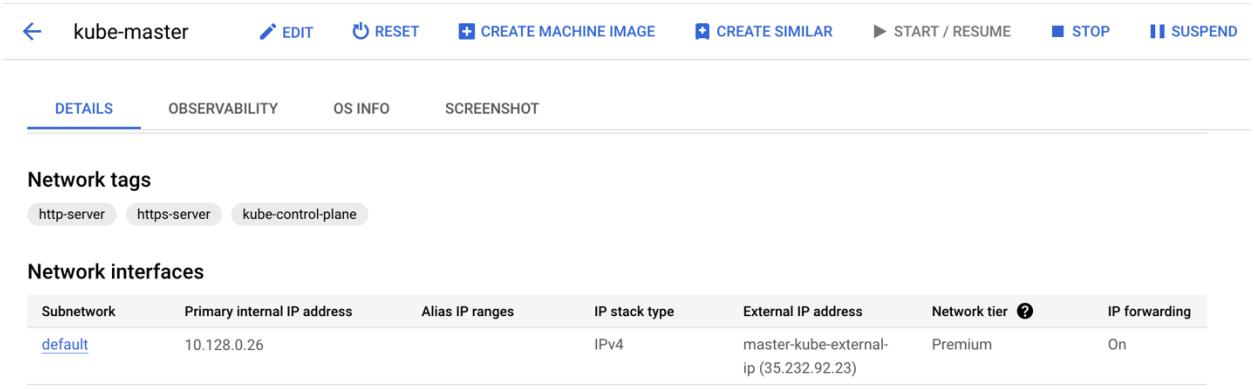
Screenshot of GCP Firewall Rules



By default, the firewall only allows traffic that originates within GCP and responses to messages that originated within the VPC. Several rules, shown in Figure 6 with *allow-calico-ingress* and *allow-kube-control-plane* as examples, were added to permit the Kubernetes control plane and calico to communicate externally. These rules are marked with the target “kube-control-plane" which can be applied to other GCP assets, including GCE VMs, to group rules. After this, the VMs can be setup through the console. The defaults are predominantly left as they are with exceptions to name and networking. For the master node, the networking should be configured with the network tags http-server, https-server, and kube-control-plane, IP forwarding should be enabled, and a static external IPv4 address should be reserved. Configuration of the kube-master node including network tags is seen in Figure 7. This permits the master node to allow inbound traffic on ports 80, 443, and those described in the custom rules as well as having a non-changing external IP address.

Figure 7

GCP VM Instance Dashboard of the Kube-Master Node



In the local cluster, there are no significant differences to the setup with the exception of a few commands that need to be ran and the machine rebooted prior to beginning the node setup. For Debian, the commands needed to modify the control group, seen in Figure 8, cannot be done in place and require a reboot to take effect. No networking modifications were necessary at the router or access point (AP) level.

Figure 8

Linux Control Group Modification Commands

```
echo ' cgroup_enable=memory cgroup_memory=1' | sudo tee -a /boot/firmware/cmdline.txt
sudo reboot now
```

The multi-cluster takes form as a dumbbell network topology where worker nodes report to their local master nodes which possess an east-west gateway to interact with other clusters. The connection between the two master nodes was a WireGuard link. WireGuard is an open source project enabling implementation of VPNs (Donenfeld, 2016). For this study, no assumption is being made for the security of this protocol. It is implemented to circumvent the limitations of a home router for the purposes of ingress and egress routing and to avoid any modification of the on-premises router. As a consumer router, the on-premises router lacked the

ability to make significant modifications beyond Network Address Translation (NAT) and Port Address Translation (PAT). For asset segmentation Istio uses namespace likeness to group assets. In the multi-cluster arrangement, both clusters share the “cloud” namespace. The cloud namespace uses the automatic Istio injection to start an Envoy container whenever a new Deployment is written. In the multi-cluster, there are two core Deployments: helloworld and ubuntu-ssh. The former is a sample application instantiated by default with the Istio installation. It serves an http server on the port 5000 which when called returns information on the services version and the pod it is running in. There are two replicas distributed across the two worker nodes in the GCP cluster. The latter deployment, ubuntu-ssh, is a locally made Docker image which installs the curl and ssh packages to be used to test communications between pods and nodes. The Dockerfile of the ubuntu-ssh including its additionally installed packages is show in Figure 9. There are four replicas of this deployment distributed to the four worker nodes evenly distributed across both the GCP and the on-premises clusters.

Figure 9

Dockerfile of the ubuntu-ssh Docker Image

```
FROM ubuntu:20.04

ARG DEBIAN_FRONTEND=noninteractive

RUN apt-get update && \
    apt-get install -y \
    curl \
    openssh-server \
    sudo
RUN mkdir /var/run/sshd
RUN echo 'root:password' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

EXPOSE 22

CMD ["/usr/sbin/sshd", "-D"]
```

Note. The password for the root command was modified from the actual values used to not disclose environment secrets.

Through route testing of a hybrid cloud architecture, Kubernetes and Istio can be verified as an effective alternative to other Zero Trust solutions. The next chapter, Chapter 4. Results, will include the implementation and findings of Kubernetes and Istio as an effective solution to Zero Trust endpoint security.

CHAPTER 4. EXPERIMENTAL RESULTS

This chapter begins with a description of the steps taken during the experimentation phase, from setting up master nodes in the cloud and local environments, to assigning the environments their respective worker nodes, to establishing an Istio east-west gateway, and to creating service deployments accessible within a single cluster and the multi-cluster boundary. Following the environment creation phase, a review of the three test types using traceroute, SSH, and HTTP, respectively, will be completed concluding with the results of the tests.

4.1. Traceroute Queries

For the queries between the pods, kubectl exec was used to run commands within pods and interact with other pod and node resources in the local and remote clusters. The core command uses the format "kubectl exec -it *podname* -n *namespace* -- *command*" to pass commands. Beginning with the traceroute command, three tests were completed: across pods within the on-premises cluster, across pods within the cloud cluster, and across pods in different clusters. The command used was "traceroute *ip*" where *ip* was replaced with the target IP address.

Figure 10

Traceroute Command Example

```
cgroach@kube-master-office:~/dev$ kubectl exec -it ubuntu-ssh-deployment-7564dc8766-t5jt4 -n cloud -- traceroute 10.244.174.143
traceroute to 10.244.174.143 (10.244.174.143), 30 hops max, 60 byte packets
 1  192-168-1-204.calico-typha.calico-system.svc.cluster.local (192.168.1.204)  0.105 ms  0.043 ms  0.028 ms
 2  192-168-1-133.kubernetes.default.svc.cluster.local (192.168.1.133)  8.504 ms  8.371 ms  8.278 ms
 3  10.132.0.1 (10.132.0.1)  47.437 ms  47.845 ms  47.768 ms
 4  10.132.0.3 (10.132.0.3)  47.716 ms  48.411 ms  47.576 ms
 5  10.244.174.143 (10.244.174.143)  47.500 ms  48.217 ms  48.145 ms
```

Note. ubuntu-ssh-deployment-7564dc8766-t5jt4 refers to the pod name of the ubuntu-ssh-deployment on the office-node-4 node. 10.244.174.143 refers to the IP address of the ubuntu-ssh-deployment pod on the gcp-node-2 node.

For the traceroute command, success was measured by the rate at which a packet reaches the final destination. In Figure 10, the success rate for the command would be 100% as all packets in the fifth entry returned successfully. Round trip times to all router hops were recorded; however, analysis was only taken on the times to reach the terminating destination.

Table 1

Table of Traceroute Results: gcp-node-1, gcp-node-2 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)
AVG	0.967	0.896	1.000	0.643
N (Count)	30	30	30	30
St Deviation	0.0999	0.372	0.000	0.380
P-Value	0.359	<0.0001	-	<0.0001

In Table 1, the intra-cluster traffic within the cloud cluster measured via the traceroute command exceeded the accuracy threshold of RQ1; however, it was not significant with the p-value exceeding 0.05. In the opposite direction, a p-value could not be calculated as there was no variation in the success rate. Therefore, there was no standard deviation. The average latency in both directions was less than 1ms which is well below the RQ2 threshold of 200ms. Both of these averages are statistically significant with p-values less than 0.0001.

Table 2

Table of Traceroute Results: office-node-3, office-node-4 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)

AVG	0.867	14.066	0.900	39.785
N (Count)	30	30	30	30
St Deviation	0.221	22.537	0.153	18.644
P-Value	0.0489	<0.0001	0.0831	<0.0001

In Table 2, the intra-cluster traffic within the office cluster measured by the traceroute command were below the threshold of RQ1 for measuring success rate. This applied to both directions. Traffic originating from office-node-3 average an 86.7% success rate that was statistically significant with a p-value of <0.05. The traffic originating from the offce-node-4 also was below the threshold at a 90% success rate; however, its p-value was above 0.05 and is not considered significant. In measuring latency, the averages of 14ms originating from one node, and 39.8ms from the other node. Both represent increases from the cloud environment, yet they still remain below the RQ2 threshold set at 200ms. This can be attributed to the network infrastructure made available to the cluster as it was using a commercially available home router. Both averages are considered statistically significant with both p-values being below 0.0001.

Table 3

Table of Traceroute Results: gcp-node-2, office-node-4 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)
AVG	0.867	101.477	1.000	59.121
N (Count)	30	30	30	30
St Deviation	0.221	53.175	0.000	33.896

P-Value	0.0489	<0.0001	-	<0.0001

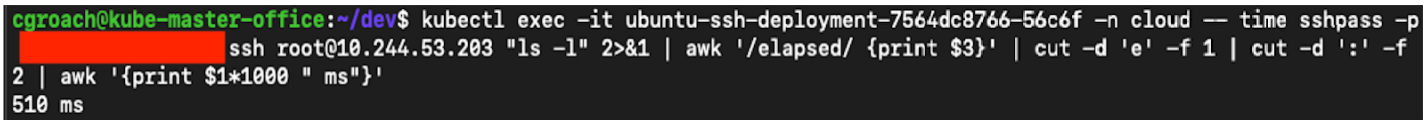
In Table 3, the inter-cluster communication traceroute commands had mixed success rates depending on the origin node. Originating from the cloud node, it reached 86.7% of the time which is considered significant with a p-value below <0.05. The office-node originating requests were successful 100% of the time. Since there was no variance in the success rates therefore no standard deviation, a p-value could not be calculated. This inconsistency with the routing through a home network as packets were frequently dropped within that environment. Regarding latency, traffic originating from the cloud node averaged 101ms round-trip, and the traffic originating from the office node averaged 59ms round-trip. Both are below the RQ2 threshold of 200ms, and both are statistically significant with p-values below 0.0001. These increases over the intra-cluster routing scenarios can be attributed to the geographical distribution and overall added latency necessary to facilitate communications over the internet.

4.2. SSH Queries

The next test with ssh followed the same three scenarios observed when using traceroute: across pods within the on-premises cluster, across pods within the cloud cluster, and across pods in different clusters. The command used was ‘time sshpass -p *omitted* ssh root@*ip* "ls -l" 2>&1 | awk '/elapsed/ {print $3}' | cut -d 'e' -f 1 | cut -d ':' -f 2 | awk '{print $1*1000 " ms"}'’ where *ip* was replaced with the target IP address. A combination of commands was used to formulate this command. Starting with *time*, this was used to collect the real time passed for the command to return. *Sshpass* is a command line utility that allows a password to be used to the ssh command without interactivity for the purposes of accelerating continuous tests. Commands *awk* and *cut* where used to parse the real time values from the time command.

Figure 11

SSH Command Example



Note. ubuntu-ssh-deployment-7564dc8766-56c6f refers to the pod name of the ubuntu-ssh-deployment on the office-node-3 node. 10.244.53.203 refers to the IP address of the ubuntu-ssh-deployment pod on the gcp-node-1 node. The password used was omitted in the figure to not disclose any secrets.

For the ssh command, success was measured by the rate at which the command does not return an error. In Figure 11, the test is a success as it does not error. The time is collected across the 10 tests in both directions. This culminates in an average success rate, an average elapsed real time, the standard deviation, and the p-value collected from a single sample t-test.

Table 4

Table of SSH Results: gcp-node-1, gcp-node-2 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)
AVG	1.000	35.000	1.000	168.4
N (Count)	10	10	10	10
St Deviation	0	18.574	0	2.458
P-Value	-	<0.0001	-	<0.0001

In Table 4, the intra-cluster cloud traffic from the SSH command had success rates of 100% when originating from either direction. Both of these averages exceed the RQ1 threshold

of 95%; however, the lack of variance in the sample with a value of 0 for the standard deviation meant that a p-value could not be evaluated. In the latency measure, traffic originating from the first node averaged 35ms. Traffic originating from the second node averaged 168.4ms. Both of which are below the RQ2 threshold of 200ms, and both are significant with p-values below 0.0001. The difference in the two averages can be attributed to routing irregularities experienced when originating from the second node. The second node tended to have higher latency yet had lower variance.

Table 5

Table of SSH Results: office-node-3, office-node-4 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)
AVG	1.000	58	1.000	110
N (Count)	10	10	10	10
St Deviation	0	8.718	0	34.351
P-Value	-	<0.0001	-	<0.0001

In Table 5, the intra-cluster office traffic from the SSH command had success rates of 100% when originating from either direction. Both of these averages exceed the RQ1 threshold of 95%; however, the lack of variance in the sample with a value of 0 for the standard deviation meant that a p-value could not be evaluated. In latency testing, traffic originating from office-node-3 averaged 58ms, and traffic originating from office-node-4 averaged 110ms. Both of these values fall below the RQ2 threshold of 200ms, and both are statistically significant with p-values below 0.0001.

Table 6

Table of SSH Results: gcp-node-1, office-node-3 Asset Pair

	TO		FROM	
	Success Rate	Time (ms)	Success Rate	Time (ms)
AVG	1.000	291	1.000	583
N (Count)	10	10	10	10
St Deviation	0	76.609	0	83.193
P-Value	-	<0.0001	-	<0.0001

In Table 6, the inter-cluster traffic from the SSH command had success rates of 100% when originating from either direction. Both of these averages exceed the RQ1 threshold of 95%; however, the lack of variance in the sample with a value of 0 for the standard deviation meant that a p-value could not be evaluated. In latency testing, traffic originating from the cloud node averaged times of 291ms round-trip, and traffic originating from the office node averaged 583ms. Both exceeded the RQ2 threshold of 200ms, and both were statistically significant. Similar to observations in Table 3, the increase in latency overall can be attributed to communications occurring between geographically displaced assets; however, the difference in direction can be attributed to routing differences when originating from one node over the other.

4.3. HTTP Queries

Finally with the HTTP command, two tests were done. Both on-premises endpoints performed a GET request on their corresponding endpoint in the GCP environment. The command used was “time curl http://*ip*:5000/hello | awk '/elapsed/ {print $3}' | cut -d 'e' -f 1 | cut -d ':' -f 2 | awk '{print $1*1000 " ms"}'” where *ip* was replaced with the target IP address.

Similar to the SSH command, the HTTP test takes advantage of the installed time, cut, and awk commands to calculate and parse the real time value for the command's completion.

Figure 12

HTTP Curl Command Example

```
cgroach@kube-master-office:~/dev$ kubectl exec -it ubuntu-ssh-deployment-7564dc8766-t5jt4 -n cloud -- time curl http:
//10.244.174.144:5000/hello | awk '/elapsed/ {print $3}' | cut -d 'e' -f 1 | cut -d ':' -f 2 | awk '{print $1*1000 "
ms"}'
230 ms
```

Note. ubuntu-ssh-deployment-7564dc8766-t5jt4 refers to the pod name of the ubuntu-ssh-deployment on the office-node-4 node. 10.244.174.144 refers to the IP address of the ubuntu-ssh-deployment pod on the gcp-node-2 node.

For the HTTP command, success was measured by the rate at which a request returns without an error. In Figure 12, the success rate for the command would be 100% as the command successfully returned. The total elapsed time for the request to return in milliseconds.

Table 7

Table of HTTP Curl Results: gcp-node-1, office-node-3 Asset Pair

	TO	
	Success Rate	Time (ms)
AVG	1.000	327.5
N (Count)	20	20
St Deviation	0	61.804
P-Value	-	<0.0001

In Table 7, the inter-cluster cloud traffic from the HTTP command involving gcp-node-1 and office-node-3 had success rates of 100%. This average exceeds the RQ1 threshold of 95%; however, the lack of variance in the sample with a value of 0 for the standard deviation meant

that a p-value could not be evaluated. Latency testing originating from the office node destined for the cloud node averaged 328ms round-trip. With a p-value below 0.0001, this is considered statistically significant. The value exceeds the RQ2 threshold of 200ms. Reasoning for exceeding the threshold can be attributed to routing between geographically displaced assets.

Table 8

Table of HTTP Curl Results: gcp-node-2, office-node-4 Asset Pair

	TO	
	Success Rate	Time (ms)
AVG	1.000	279
N (Count)	20	20
St Deviation	0	62.230
P-Value	-	<0.0001

In Table 8, the inter-cluster cloud traffic from the HTTP command involving gcp-node-2 and office-node-4 had success rates of 100%. This average exceeds the RQ1 threshold of 95%; however, the lack of variance in the sample with a value of 0 for the standard deviation meant that a p-value could not be evaluated. Latency in the second HTTP test, averaged 279ms round-trip. With a p-value below 0.0001, this is considered statistically significant. Additionally, the average exceeds the RQ2 threshold of 200ms. The average latency consistency of the two scenarios in the HTTP test is a positive sign of consistent routing between the two clusters.

With the collection of Traceroute, SSH, and HTTP metrics from pods to pods, the overall effectiveness of the multi-cluster Zero Trust topology can be measured. In Chapter 5. Discussion, the significance of the eight scenarios will be reviewed to compare the effectiveness of this Zero

Trust approach to other theoretical and practical implementations of Zero Trust architectures spanning hybrid environments.

CHAPTER 5. DISCUSSION

From the results in Chapter 4, the usage of Kubernetes and Istio can certainly be used to orchestrate a network overlay of an IT infrastructure; however, its efficiency in doing so is a weakness of this implementation. Chapter 5. Discussion takes a rigorous analysis of the intra-cluster and inter-cluster queries to compare the efficiency of this overlay to other examples.

5.1. RQ1

For the policy engine, Istio's service mesh namespace segmentation and an Istio Authorization Policy were implemented to segment the services and apply rules on inter service communication. In the connectivity tests involving the Traceroute command (see Table 1, Table 2, and Table 3), intra-cluster communications in the cloud cluster exceeded the threshold of a 95% success rate; however, intra-cluster communications in the office cluster and inter-cluster communications between clusters were not as efficient. This is reasoned to be as a fault of the reliability of the consumer router used to host the office cluster as there was high variance in the latencies and success rates to all traceroute queries that had at least one resource under that router's domain. Additionally, requests made by the traceroute command have a limited time to live and differentiates between each retry as a method to measure consistency. The findings of the traceroute tests provide an indicator that this hybrid network topology is prone to dropping packets and requests in local environments.

In the test involving the SSH and HTTP commands, the outcome was a success across all tests performed. There were no variances in the success rate in intra-cluster traffic and inter-cluster traffic showing that Kubernetes and Istio can be used to manage TCP traffic in a method that is consistent with topology segmentation and authentication policy objects that were configured.

5.2. RQ2

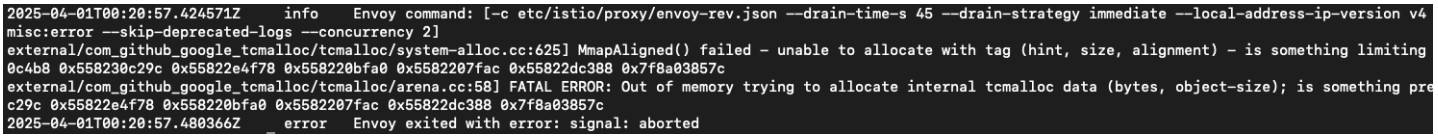
Tests measuring latency had a far more revealing outcome as a majority of the tests conducted and all tests that passed traffic far exceeded the threshold set for a healthy network used to orchestrate real-time dependent communications. In the traceroute tests, intra-node-communications were well below the threshold of 200ms with the cloud environment average response time in both directions being less than 1ms. In the office environment, similar observations were made with the averages in both directions being less than 200ms. Inter-cluster communication saw much higher values as the average traffic latency originating from the cloud cluster exceeded 100ms. This is an indicator that Istio may be lighter on resource constraints such as CPU and memory; however, it has a heavy reliance on networking resources, and Istio's success correlates with the network resources made available. The results from the Traceroute command had a high variance with values ranging from 47ms to 188ms. These variations came as a result of the internet control message protocol, the protocol Traceroute operates on, not being a high priority protocol compared to the transmission control protocol or the user datagram protocol. In the SSH and HTTP tests, there are similar findings that the network overhead of Istio increases the latency of the requests as intra-cluster communications tests; however, they remain under the threshold of 200ms. In inter-cluster communications, the SSH commands frequently exceeded 300ms with some exceeding 500ms. Even for small commands as the SSH tests passed the list command would have human noticeable latencies. The inter-cluster HTTP requests average around 300ms. Istio can be used to effectively monitor intra-cluster communications and make appropriate security and authentication actions; however, it adds network latency overhead that diminishes its overall effectiveness as a replacement of other Zero Trust topology implementations. For applications that exist on a single site, Istio serves as an excellent option

for an endpoint service mesh; however, environments that span multiple sights may not meet the tight tolerances that real time services may require for efficient operation.

Additionally, several observations were made irrespective of RQ1 and RQ2. This approach was tested predominantly on machines that possessed an Intel chip. Devices with ARM chips were tested. Notably, several varieties of the Raspberry Pi 4 with memory capacities varying from 4gb to 8gb. Although these devices could be added to the cluster, not all ARM chips were compatible with the Istio Envoy proxy which Istio uses to route container traffic in-between pods on the same node and neighboring nodes. An example of the error message generated during the installation of the Istio Envoy proxy is shown in Figure 13. These machines would frequently error out with Out of Memory flags; however, their total available memory would exceed that of similar worker nodes with Intel chips in GCP. Not only does Istio have limitations regarding its network resource usage, but it also has restrictions on what devices are compatible with its technology.

Figure 13

Istio Envoy Proxy Error Message on an ARM-hosted Pod



Kubernetes and Istio being the backbone of cloud based network topologies has some viability in its method to manage traffic between the devices that are within its domain; however, it has drawbacks targeting its network efficiently, notably in inter-cluster hybrid communications, and device compatibility.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

This study has shown that Istio and Kubernetes can be used to efficiently manage a dynamic overlay supporting multiple endpoints in an “out of the box” Zero Trust security architecture. Through experimentation without the reliance on commercial vendors, Kubernetes and Istio was able to manage secure communications at a low cost. Within the scope and the methods observed by this study, several challenges to Kubernetes and Istio adoption as a Zero Trust overlay have been identified including restrictions on devices supported and the network overhead generated by the forwarders. The issues of restricted device support and networking resource overhead could have a negative impact on latency sensitive applications. Kubernetes and Istio are both cloud native products and have been optimized for cloud environments. Kubernetes and Istio together can sufficiently implement a Zero Trust framework confirming the alternative hypothesis of RQ1; however, its poor latency over a multi-cluster environment affirms that the null hypothesis of RQ2, that the introduction of these two technologies will cause communications to exceed 200ms of latency, to be true. The latency and chipset architecture issues point to the need to explore optimizations beyond cloud infrastructures that could make an Istio based Zero Trust solution a very effective alternative to managing hosts and servers in a geographically distributed environment. This is a natural extension of the Istio functionality, particularly in the context of organizations moving more into hybrid cloud IT environments. The evaluation of the success of Kubernetes and Istio as a Zero Trust framework for endpoint devices addresses the gap of service mesh platform support for hybrid endpoint focused network topologies.

Future work will investigate Istio’s limitations to support ARM based architectures since the Google dependency tcmalloc was not compatible with the chips tested in this study. Several

potential work arounds to this constraint could involve the compilation of a modified operating system. Additionally, forwarding performance optimizations will be evaluated through a more extensive test suite and end-point resources tuning. A simple dumbbell topology was evaluated in this study; however, that may not be the best model for organizations that exist in more than one cloud provider or that operate out of multiple satellite offices. Additional work is needed to investigate other topologies, particularly those of multi-site organizations using an Internet-based backbone. Finally, an Istio based Zero Trust architecture lends itself well to enabling flexible, scalable, dynamic allocation of Network Functions and Security Functions to support security policy monitoring and enforcement. Future work will include the investigation of such orchestration of functions complementing the Zero Trust overlays.

REFERENCES

Alboqmi, R., Jahan, S., & Gamble, R. F. (2023). A runtime trust evaluation mechanism in the service mesh architecture. *23 10th International Conference on Future Internet of Things and Cloud (FiCloud)*, 242–249. https://doi.org/10.1109/FiCloud58648.2023.00043

Arora, S., & Hastings, J. (2024). Microsegmented Cloud Network Architecture Using Open-Source Tools for a Zero Trust Foundation. *2024 17th International Conference on Security of Information and Networks (SIN)*, 1–8. https://doi.org/10.1109/sin63213.2024.10871361

Berenberg, A., & Calder, B. (2022). Deployment archetypes for cloud applications. *ACM Comput. Surv.*, *55*, 3. https://doi.org/10.1145/3498336

Bicakci, K., Uzunay, Y., & Khan, M. (2021). Towards Zero Trust: The Design and Implementation of a Secure End-Point Device for Remote Working. *2021 International Conference on Information Security and Cryptology (ISCTURKEY)*. https://doi.org/10.1109/iscturkey53027.2021.9654298

Billawa, P., Bambhore Tukaram, Anusha, Ferreyra, D., Steghöfer, J.-P., Scandariato, R., & Simhandl, G. (2022). SoK: Security of microservice applications: A practitioners' perspective on challenges and best practices. *Proceedings of the 17th International Conference on Availability, Reliability and Security*. https://doi.org/10.1145/3538969.3538986

Brazhuk, A., & Fernandez, E. B. (2023). An abstract security pattern for zero trust access control. *Proceedings of the 29th Conference on Pattern Languages of Programs*.

D'Silva, D., & Ambawade, D. D. (2021). Building A Zero Trust Architecture Using Kubernetes. *2021 6th International Conference for Convergence in Technology (I2CT)*, 1–8. https://doi.org/10.1109/I2CT51068.2021.9418203

Donenfeld, J. A. (2016). *WireGuard: fast, modern, secure VPN tunnel*. Wireguard.com. https://www.wireguard.com/

Goel, A., & Thangaraju, B. (2022). Authenticating distributed systems using SPIRE over kubernetes cluster. *2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 1–6. https://doi.org/10.1109/CONECCT55679.2022.9865835

Istio Authors. (2025). *Authorization Policy*. Istio. https://istio.io/latest/docs/reference/config/security/authorization-policy/#Source

Jiang, X., Shokri-Ghadikolaei, H., Fodor, G., Modiano, E., Pang, Z., Zorzi, M., & Fischione, C. (2019). Low-Latency Networking: Where Latency Lurks and How to Tame It. *Proceedings of the IEEE*, *107*(2), 280–306. https://doi.org/10.1109/JPROC.2018.2863960

Kiali. (2024). *Kiali*. Kiali.io. https://kiali.io/

Kurpad, S., BT, S., Vijaykumar, S., Jain, S., & Kalambur, S. (2023). Microarchitectural Analysis and Characterization of Performance Overheads in Service Meshes with Kubernetes. *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*, 1–6. https://doi.org/10.1109/asiancon58793.2023.10270428

Liu, J., Xiong, H., & Yan, C. (2025). An Open Interconnection System for Computing Power Based on Service Mesh. *2025 IEEE 5th International Conference on Power, Electronics*

and Computer Applications (ICPECA), 266–269. https://doi.org/10.1109/icpeca63937.2025.10928796

Liu, Z., Qian, Z., & Li, N. (2023). An East-West-Traffic Governance System Based on eBPF and Centralized Gateway. *2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*, 1366–1370. https://doi.org/10.1109/icsece58870.2023.10263324

Maia, T., & Correia, F. (2023). Service mesh patterns. *Proceedings of the 27th European Conference on Pattern Languages of Programs*. https://doi.org/10.1145/3551902.3551962

Nathaniel, L., Perdana, G. V., Hadiana, M. R., Negara, R. M., & Hertiana, S. N. (2023). Istio API Gateway Impact to Reduce Microservice Latency and Resource Usage on Kubernetes. *2023 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 43–47. https://doi.org/10.1109/isitia59021.2023.10221035

Pizzato, F., Bringhenti, D., Sisto, R., & Valenza, F. (2024). An intent-based solution for network isolation in Kubernetes. *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. https://doi.org/10.1109/netsoft60951.2024.10588939

Rey, P., & Kieth, R. (2022). Designing secure and scalable end-to-end private network connections between sites via overlay tunnels. *Proceedings of the 2022 5th International Conference on Electronics, Communications and Control Engineering*, 50–56. https://doi.org/10.1145/3531028.3531036

Rodigari, S., O'Shea, D., McCarthy, P., McCarry, M., & McSweeney, S. (2021). Performance analysis of zero-trust multi-cloud. *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 730–732. https://doi.org/10.1109/CLOUD53861.2021.00097

Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero Trust Architecture. *Zero Trust Architecture*, *800-207*(800-207). https://doi.org/10.6028/nist.sp.800-207

Sheikh, O., Dikaleh, S., Mistry, D., Pape, D., & Felix, C. (2018). Modernize digital applications with microservices management using the istio service mesh. *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, 359–360.

Shen, Q., & Shen, Y. (2024). Endpoint security reinforcement via integrated zero-trust systems: A collaborative approach. *Computers & Security*, *136*, 103537. https://doi.org/10.1016/j.cose.2023.103537

Singh, S., Muntean, C. H., & Gupta, S. (2024). Boosting Microservice Resilience: An Evaluation of Istio's Impact on Kubernetes Clusters Under Chaos. *2024 9th International Conference on Fog and Mobile Edge Computing (FMEC)*, 245–252. https://doi.org/10.1109/fmec62297.2024.10710237

The Kubernetes Authors. (2022, October 24). *Kubernetes Components*. Kubernetes. https://kubernetes.io/docs/concepts/overview/components/

Varvello, M., Azurmendi, Iñigo Querejeta, Nappa, A., Papadopoulos, P., Pestana, G., & Livshits, B. (2021). VPN-Zero: A privacy-preserving decentralized virtual private network. *2021 IFIP Networking Conference (IFIP Networking)},* 1–6. https://doi.org/10.23919/IFIPNetworking52078.2021.9472843

Ward, R., & Beyer, B. (2014). BeyondCorp: A new approach to enterprise security. *;Login:*, *Vol. 39, No. 6*, 6–11.

Yao, Q., Wang, Q., Zhang, X., & Fei, J. (2021). Dynamic access control and authorization system based on zero-trust architecture. *Proceedings of the 2020 1st International*

Conference on Control, Robotics and Intelligent System, 123–127.

https://doi.org/10.1145/3437802.3437824

APPENDIX A

Code for Node Setup: node_setup.sh

```
#!/bin/bash

architecture=$(uname -m)

# Check the architecture and set the variable accordingly
if [[ "$architecture" == "x86_64" ]]; then
    architecture="amd64"
    sudo sed -i '/^GRUB_CMDLINE_LINUX_DEFAULT/ s/"$/cgroup_enable=memory
cgroup_memory=1"/' /etc/default/grub
    sudo update-grub
    sudo su -c 'modprobe br_netfilter'
elif [[ "$architecture" == "aarch64" ]]; then
    architecture="arm64"
else
    echo "Unknown architecture: $architecture. Please check the system."
    exit 1
fi

mkdir dev
cd dev

sudo apt-get install -y curl
curl -sSL https://get.docker.com | sh
sudo usermod -aG docker $USER

VER=$(curl -s https://api.github.com/repos/Mirantis/cri-dockerd/releases/latest|grep
tag_name | cut -d '"' -f 4|sed 's/v//g')
wget https://github.com/Mirantis/cri-dockerd/releases/download/v${VER}/cri-dockerd-
${VER}.${architecture}.tgz
tar xvf cri-dockerd-${VER}.${architecture}.tgz
sudo mv cri-dockerd/cri-dockerd /usr/local/bin/

wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-
docker.service
wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-
docker.socket
sudo mv cri-docker.socket cri-docker.service /etc/systemd/system/
sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-
docker.service

sudo systemctl daemon-reload
```

```
    sudo systemctl enable cri-docker.service
    sudo systemctl enable --now cri-docker.socket

    sudo apt-get update && \
      sudo apt-get install -y dphys-swapfile && \
      sudo dphys-swapfile swapoff && \
      sudo dphys-swapfile uninstall && \
      sudo systemctl disable dphys-swapfile

    sudo apt-get update
    sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

    curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
    sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg

    echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
    sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list

    sudo apt-get update
    sudo apt-get install -y kubelet=1.30.3-1.1 kubeadm=1.30.3-1.1 kubectl=1.30.3-1.1
    sudo apt-mark hold kubelet kubeadm kubectl
```

APPENDIX B

Code for Master Node Setup: master_setup.sh

```
#!/bin/bash

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.28.2/manifests/tigera-operator.yaml
kubectl apply -f calico-install-config.yaml

cd /usr/local/bin
sudo curl -L https://github.com/projectcalico/calico/releases/download/v3.29.0/calicoctl-
linux-amd64 -o kubectl-calico
sudo chmod +x kubectl-calico
cd ~/dev

# Needs to wait until kubectl pods -A is done running
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```