UiA University of Agder

Tapping network traffic in Kubernetes

Diving deeper into the footprint, impact and characterization of the sidecar method with dimensions of volume, scalability, load and stability

SIGBJØRN SKOLEM LEDAAL

SUPERVISOR
Sigurd Brinch & Roger Skjetlein

University of Agder, 2023
Faculty of Engineering and Science
Department of Engineering and Sciences

Master

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Preface

I would like to thank Roger Skjetlein from Telenor for introducing me to the idea and concept behind the thesis, and for being a helpful project supervisor, providing insightful comments along the course of the thesis. In addition, I would like to thank Telenor Security Operation Center for providing me the opportunity to write this thesis, as well as giving me access to the resources needed to conduct the research. I must also reach out a thank you to Sigurd K. Brinch who served as my supervisor from the University of Agder. Lastly, I send out a collective thank you to all the co-students who have helped me in times of need.

Sigbjørn Ledaal
Grimstad 14.05.2023

Abstract

The rapid increase in cloud usage among organizations has led to a shift in the cybersecurity industry. Whereas before, organizations wanted traditional security monitoring using statically placed IDS sensors within their data centers and networks, they now want dynamic security monitoring of their cloud solutions. As more and more organizations move their infrastructure and applications to the cloud the need for cybersecurity solutions that can adapt and transform to meet this new demand is increasing. Although many cloud providers, provide integrated security solutions, these are dependent on the correct configuration from the customers, which may rather want to pay a security firm instead. Telenor Security Operation Center is a long contender in the traditional cybersecurity firm space and is looking to move into IDS monitoring of cloud solutions, more specifically providing network IDS monitoring of traffic within managed Kubernetes clusters at cloud providers, such as Amazon Web Services Elastic Kubernetes Service. This is to be accomplished by providing all the desired pods within a cluster their own sidecar container, which acts as a network sniffer that sends the recorded traffic through vxlan to an external sensor also operating in the cloud. By doing this, traditional IDS monitoring suddenly becomes available in the cloud, and is covering a part that is often neglected in cloud environments, and that is monitoring the internal Kubernetes cluster traffic.

AWS EKS was used as a testing ground for a simulated Kubernetes cluster running sample applications monitored by the sidecar container. Which is essentially a Python script sniffing the localhost traffic of the shared network namespace of a Kubernetes pod. This infrastructure will be generated by a set of Terraform files for automated setup and reproducibility, as well as making use of the gitops tool Fluxcd for syncing Kubernetes manifests. The solution will also be monitored by a complete monitoring solution in the form of *kube-prometheus-stack* which will provide complete insight into performance metrics down at the container level, through Prometheus and Grafana. Finally, a series of performance tests will be conducted, using k6s and iperf, automated by Ansible, to gather the performance impact of the sidecar container.

A series of iperf and k6s tests were conducted against the sidecar container. The k6s test was run at a data rate of `3 Mb/s` and showed that the data rate needed to be higher to gather useful performance metrics. This is where iperf took over and tested the sidecar container at data rates of 50,100,250 and 500 `Mb/s` using a server at the University of Agder as base. These initial raw performance results showed a max CPU usage of 11.8% of the Kubernetes node's 2 vCPU's. Together with a max memory usage of `14 MB` this showed that the sidecar container does not consume a vast amount of resources. And has the potential as a scalable and efficient network tapping method in Kubernetes. However, some anomalies were discovered during the performance testing that revealed undiscovered issues with the method. One of which was packet anomalies between the number of packets at the sensor and the number of packets observed by the iperf server at the University of Agder. Due to the many layers involved in the networking stack for this method, there needs to

be conducted additional research into how these anomalies arise. While also considering alternative transport methods to vxlan.

Contents

List of Figures

List of Tables

Abbreviations

ARP Address Resolution Protocol. 10

AWS Amazon Web Services. iii, xi, 2, 12, 14, 20, 24–27, 47–49

CNI Container Network Interface. 12, 21

CSA Cloud Security Alliance. 6

DDoS Distributed Denial of Service. 1

EC2 Elastic Cloud Compute. 14, 20, 26, 31, 47, 49

EKS Elastic Kubernetes Service. iii, 2, 14, 24, 26, 47, 49

IaaS Infrastructure as a Service. 5, 7

IAM Identity Access Management. 14

IDS Intrusion Detection System. 1, 2, 15, 19–23, 47

PaaS Platform as a service. 5, 7

SaaS Software as a service. 5

TSOC Telenor Security Operation Center. iii, 1–4, 19, 24, 47, 48

VPC Virtual Private Cloud. 14, 20, 25–27, 47, 49

Chapter 1

Introduction

1.1 Motivation

Telenor Security Operation Center (TSOC) which is a part of Telenor offers cybersecurity services such as *intrusion detection systems* (IDS) and log analysis, as well as defense against *Distributed Denial of Service* (DDoS) attacks, to customers. Traditionally, IDS monitoring consists of having a passive monitoring device within a network, that uses signatures to match network traffic with known malicious patterns [1]. This deployment method assumes that a network monitoring device is placed inside a customer's datacenter and is setup to passively receive the customer's traffic. However, a problem arises when the customer's network is not limited to a single datacenter, but is located in the cloud. There arises a need for a solution that can provide IDS protection in cloud environments where it is not possible for security organizations such as TSOC to physically put a passive monitoring device within a customer's network.

The rise of cloud computing has shown an exponential growth in organizations utilizing cloud providers instead of maintaining their own servers and infrastructure. With surveys showing that up to 70% of organizations are using two or more cloud providers [2]. Which means that almost every organization is using at least one cloud provider in some extent [2]. This has made TSOC look into how IDS monitoring can be achieved in the ever growing cloud landscape. Nevertheless, the cloud is large, and some areas are higher priorities than others, with the container orchestration platform Kubernetes being the starting target. Within Kubernetes, there are pods that run on the nodes within a Kubernetes cluster. A pod is the smallest unit within Kubernetes, and can contain multiple containers that share storage and network resources and run in a shared context [3]. And it is within each of these pods that TSOC want to use a technique called the sidecar pattern to tap the network traffic from the application containers. The solution is to attach a sidecar container to the application container within a pod, this sidecar will passively record the network traffic and send it to an IDS sensor which could exist in another place in the cloud, outside the monitored Kubernetes cluster. The purpose of this research proposed by TSOC is to research the footprint, impact, and performance of the sidecar container with dimensions of load and stability.

1.2 Problem statement and Research questions

Each pod in a Kubernetes cluster will consume resources from the node it is deployed on. Best practices recommend that each pod is set with resource limits, which means that the pod will be stopped if it tries to consume more resources than are set in its limits. These limitations can also be set per container within a pod, such as on the network tapping sidecar container. What the resource limit of the sidecar container should be would be difficult to know without doing a rigorous performance test of the sidecar. Due to the operative nature of network monitoring setting a too low resource limit would halt the IDS monitoring during traffic spikes. From this background, the issue that must be addressed is:

> *What are the performance characteristics of a sidecar container tapping network traffic in Kubernetes?*

Given the plethora of different parameters that could play a part in this research question, it can be divided into a set of smaller goals to isolate parts of the problem:

1. Setup a lab environment simulating a Kubernetes cluster hosted at a cloud provider. Including configuration for easy repeatability.
2. Configuration of a monitoring solution that can monitor performance statistics such as CPU, memory, and network bandwidth of both the compute nodes and the sidecar container.
3. Conduct a series of performance tests of the sidecar container by generating network traffic against pods that are monitored.

Knowing the performance characteristics of the network tapping sidecar container puts TSOC in a better position to further develop IDS monitoring within Kubernetes in the cloud. The results of the performance test may lead to a change in strategy for the sidecar pattern.

1.3 Scope and limitations

Given the plethora of choices of cloud providers, this thesis will focus on one of the most widely used, Amazon Web Services (AWS). This is due to the fact that it is what TSOC had at hand, while also taking in the cost factor. It is cheaper to pay for a managed Kubernetes service in the cloud than to procure your own servers, while also eliminating the infrastructure management of servers. It is also beneficial for the thesis to utilize the same system that customers are using, such as AWS Elastic Kubernetes Service (EKS). Furthermore, this thesis is not about Kubernetes itself, and given the complexity of the Kubernetes ecosystem, choices will be made to ease the testing of the problem statement. And will not dive deep into Kubernetes internals that may affect the testing. That will be out of scope for this thesis. It must also be mentioned that AWS and Kubernetes are monumental and complex tools to learn and even harder to master. Therefore, a substantial amount of this thesis has been used to learn how to utilize and make use of AWS and Kubernetes to help research the problem statement.

1.4 Methodology

The main research goal of this thesis was to explore the performance impact of tapping network traffic in Kubernetes through the sidecar container pattern. The problem that warrants solving is how to efficiently monitor network traffic within Kubernetes in the cloud, and the engineered solution is to utilize sidecar containers. With this background, we are approaching the field of applied research methods. Applied research focuses on understanding the performance impact of a proposed engineered system that solves a preconceived problem [4, p. 74]. Within applied research, there are two different research paths: applied experimentation and applied observational study. While applied experimentation focuses on controlled tests to determine how well a system performs, applied observational studies observe the behavior of an engineered system in varying conditions [4, p. 74]. Given the preconceived problem stated in this thesis, the conditions of an applied observational study apply. The sidecar container pattern is the engineered solution, and we want to monitor how this system behaves under different conditions in the cloud. And given the number of parameters that exist in the cloud and between different cloud providers, it would be next to impossible to perform controlled experiments as described in applied experimentation.

Within the realm of applied observational studies, there are two subgroups: exploratory studies and descriptive studies [4, p. 301]. An exploratory study focuses on testing the behavior of a system, for example, firewall processing speed or cryptographic hashing performance. On the other hand, a descriptive study explores the results of applying foundational research [4, p. 301]. We want to explore the performance of a given system, therefore, an exploratory study aligns with our research goals.

Within applied exploratory studies, there are several ways to explore the behavior of a system, either through operational bounds testing or through sensitivity analysis [4, p. 301]. The objective of operational bounds testing is simply to test the performance boundaries of the given system. However, there are different methods of operational bounds testing, which are:

- **Performance testing**
- **Stress testing**
- **Load testing**

Performance testing is to ensure that the system can tolerate expected loads, while load testing tries to push the system to its limits. Lastly, stress testing goes beyond the expected limits of the system to observe how the system behaves at extreme load [4, p. 302]. Given that TSOC have developed a prototype of the system that will be researched in this thesis, there are some initial expectations of the performance of the sidecar container. With this in mind, performance testing seems like the most appropriate starting point to observe for this thesis. However, TSOC would also be interested in any data gathered on the maximum load and performance in the extremes for this system.

The testing will be conducted by deploying example applications to the Kubernetes cluster and attaching the network tap sidecar. Thereafter, using the load testing applications k6s and iperf, perform a set of tests that will generate traffic. In-depth monitoring of the sidecar will be observed to see the initial performance of the sidecar. These results will be used to determine if the sidecar will need to be changed or optimized before being used in a live operational environment.

1.5 Outline

Given the complicated nature of the cloud and its components, the thesis will begin Chapter 2 by giving an introduction to the technologies and systems used to conduct the research. Chapter 3 will explore relevant research about the thesis topic, but the research conducted on tapping network traffic from within Kubernetes is limited, hence why TSOC wants this research conducted. Afterwards, chapter 4 will explore the lab environment and the approach for testing the sidecar container. The results of the testing are shown in chapter 5, before being discussed in chapter 6 along with considerations. Lastly, chapter 7 concludes the results of the performance testing and explores future research that would need to be conducted in this area.

Chapter 2

Background

2.1 The Cloud

Earlier in this thesis, we were introduced to the term *the cloud* but what is being referenced when utilizing this term? Where and what is the cloud in general terms? The major cloud company, *Cloudflare* defines the cloud as *"Servers that are accessed over the Internet, and the software and databases that run on those servers"* [5]. This definition answers some questions but is not quite enough to answer everything. To elaborate further, *Cloudflare* states that the servers in the cloud are located in data centers around the globe. And the main advantage of this is that companies around the world do not have to manage these physical servers themselves. The data centers are operated by a cloud provider that manages all the infrastructure needed and rents out the servers [5]. Another major cloud provider, *Microsoft Azure*, describes the cloud as: *"The cloud is not a physical entity, but instead a vast network of remote servers around the world that are hooked together and meant to operate as a single ecosystem"* [6]. And this concept of a single ecosystem is what makes the cloud appealing to organizations. By providing the power of complex infrastructure around the globe through renting options, cloud computing has grown to be widely used among companies around the world. With market research showing that around 90% of enterprises are using a cloud provider in some form [7]. However, the cloud is not a single entity but a collection of different services, and there are a plethora of different terms and definitions of cloud services. Cloud services can be divided into a collection of major service models, though there are some industry terms that must be defined in order to understand the differences.

Below is a list of some of the major terms within cloud computing and cloud service models:

- **Infrastructure as a Service (IaaS)**: A customer can rent infrastructure through a cloud provider and is given direct access to servers without needing to worry about the management of physical servers themselves [5].
- **Platform as a service (PaaS)**: PaaS cloud providers offer a platform for building applications to customers. The customer will get everything provided, such as an operating system, database management, and development tools [5].
- **Software as a service (SaaS)**: In SaaS the cloud provider provides everything to the customer, even the application itself. The customer rents a piece of software directly and does not need to worry about the infrastructure behind it [5].

- **Private Cloud**: A private cloud deployment is a piece of infrastructure wholly dedicated to a single organization. The infrastructure could be a single server or a whole dedicated data center. For organizations that do want to share computing power or servers with other customers [8].
- **Public Cloud**: In a public cloud deployment, multiple customers may share the same physical server and its computing power. The cloud provider rents out different tiers of computing power, and the data centers share multiple organizations [8].
- **Hybrid Cloud**: A hybrid cloud deployment involves connecting multiple computing environments together. This could be connecting a public cloud to an on premise computing environment and configuring information sharing in such a way that some data will always be computed in the on premise environment. However, hybrid cloud deployments can also involve multiple public clouds connected together [8].

A graphical representation of the main cloud service models can be seen in the figure below:

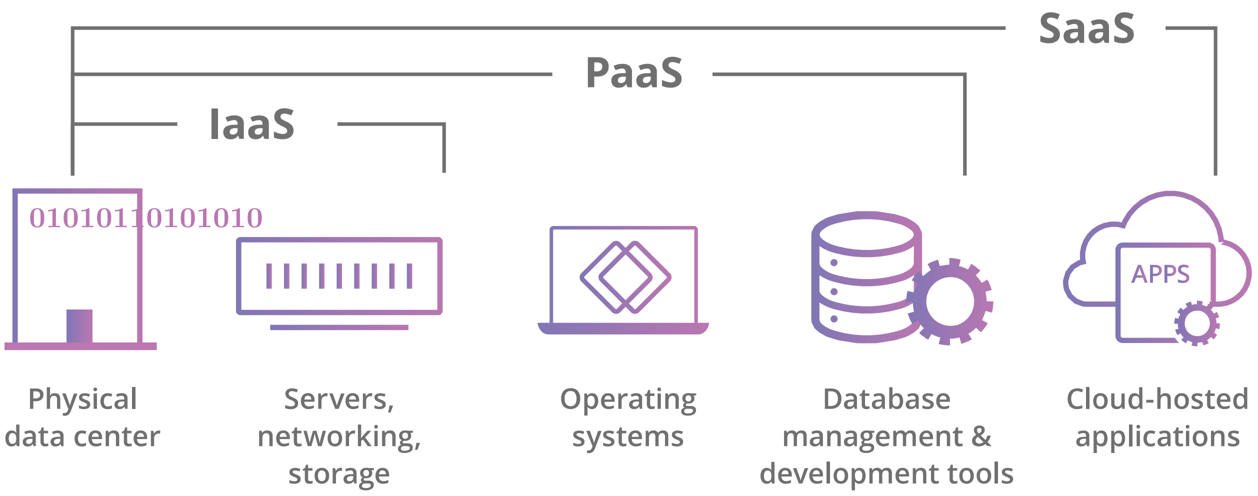

SaaS
PaaS
IaaS
010101110101010
APPS
Physical data center
Servers, networking, storage
Operating systems
Database management & development tools
Cloud-hosted applications

Figure 2.1: Overview of cloud service models [5]

2.1.1 Security and Monitoring

One of the rising issues with the widespread adoption of the cloud is the cybersecurity aspect. Many organizations moved their applications and infrastructure to the cloud without proper knowledge and expertise in how to utilize the security features of the cloud provider. This has led to a range of cybersecurity incidents where the cloud infrastructure of an organization has been compromised in some way. In the cloud industry, there is an organization called the Cloud Security Alliance (CSA) which is a global organization focused on improving and defining best practices for operating in the cloud computing environment [9]. The CSA collects expertise from industry leaders, governments, and researchers to provide a forum where information on how to protect the cloud can be shared in an efficient manner [9]. Further, the CSA releases reports on the top threats against cloud computing environments; one of these reports is called the *Pandemic Eleven* and contains the eleven largest threats against cloud computing [10]. Below is a table of the results from this report:

Survey Results Rank	Issue Name
1	Insufficient ID, Credential, Access and Key Mgt, Privileged Accounts
2	Insecure Interfaces and APIs
3	Misconfiguration and Inadequate Change Control
4	Lack of Cloud Security Architecture and Strategy
5	Insecure Software Development
6	Unsecure Third Party Resources
7	System Vulnerabilities
8	Accidental Cloud Data Disclosure/ Disclosure
9	Misconfiguration & Exploitation of Serverless & Container Workloads
10	Organized Crime/ Hackers/ APT
11	Cloud Storage Data Exfiltration

Table 2.1: Pandemic Eleven report top threats [10]

As seen in the table above, the top three threats against cloud computing environments are related to misconfiguration and management of security in the cloud. And these parameters are under the control of the customer renting the cloud and not the cloud provider themselves, although some cloud environments are harder to master than others. And taking into consideration that it is up to the cloud provider to give easy access to efficient management tools to properly configure against these vulnerabilities, It must also be noted that when utilizing IaaS or PaaS, the secure configuration of the applications within these services is up to the customer. And that involves configuring the appropriate security monitoring for both the cloud environment and the applications deployed within.

2.2 Kubernetes

Kubernetes is one of the main components of the research in this thesis and is a core component of many cloud computing environments. What Kubernetes offers is a *"portable, extensible, open sourced platform for managing containerized workloads and services."* [11]. Kubernetes was developed within Google before becoming open source in 2014 and continues to be the world's leading platform to run containerized applications, with over half of the world's container organizations using Kubernetes [12]. It is important to note that a full description of how Kubernetes works is way beyond the scope of this thesis; as such, the main focus will be on the core components related to the thesis. Next, to understand why Kubernetes is so popular and game-changing, there is a need to look back in time and look at the different deployment eras.

Below is an overview of the three main deployment eras as described by Kubernetes themselves [11]:

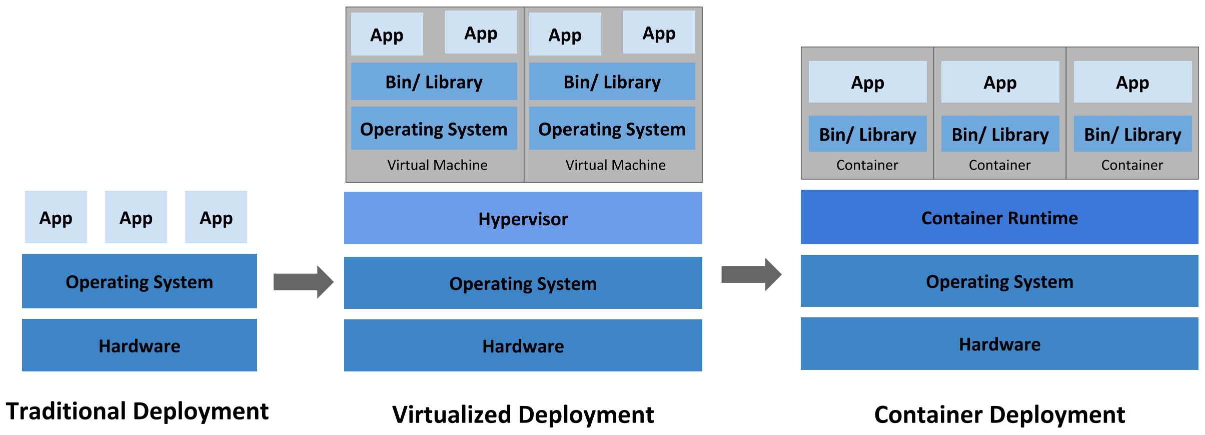


Figure 2.2: Overview of the main deployment era's [11]

The traditional deployment era consisted of running all required applications on the same physical server, running them all on the same operating system side by side. This type of deployment did not utilize the resources of the underlying hardware efficiently because a single application could use CPU and memory and make the other applications underperform. There are also issues of dependencies when different applications require different versions of the same software to be installed on the same operating system [11].

Then, researchers figured out a way to virtualize a machine within a machine by running multiple virtual servers within a single physical server. The virtual machines are given a piece of the underlying resources and are completely separate from each other. This eliminated many of the headaches of the traditional deployment, although these virtual machines produced a lot of overhead that required a significant portion of the resources of the physical server [11].

Lastly, in the container deployment era, a solution was found to make applications share operating systems without worrying about dependency headaches as in the traditional era. While also eliminating the overhead of an entire virtual machine. The solution was to create isolated containers that each have a share of the resources of the operating system but have their own filesystem. They are managed through a container runtime, which is more lightweight than an entire virtual machine [11].

Now that we have seen what containers are, the question that quickly arose was how to manage workloads when running hundreds and thousands of containers across several virtual machines in the cloud or on premise. This is where Kubernetes comes in as a container orchestration platform. The simplest example of Kubernetes is that if a container goes down, another one should start and take its place. With Kubernetes, such workflows can be configured and expanded upon. A starting point for some features that Kubernetes can offer is as follows [11]:

- Service Discovery and load balancing
- Storage provisioning
- Automated rollouts and rollbacks
- Self-healing

These features and more can be provided through the large community of Kubernetes addons that enhance the standard Kubernetes experience.

To understand how a Kubernetes cluster operates, it may be helpful to have a graphical representation. An overview of Kubernetes standard components can be viewed in the following figure [13]:

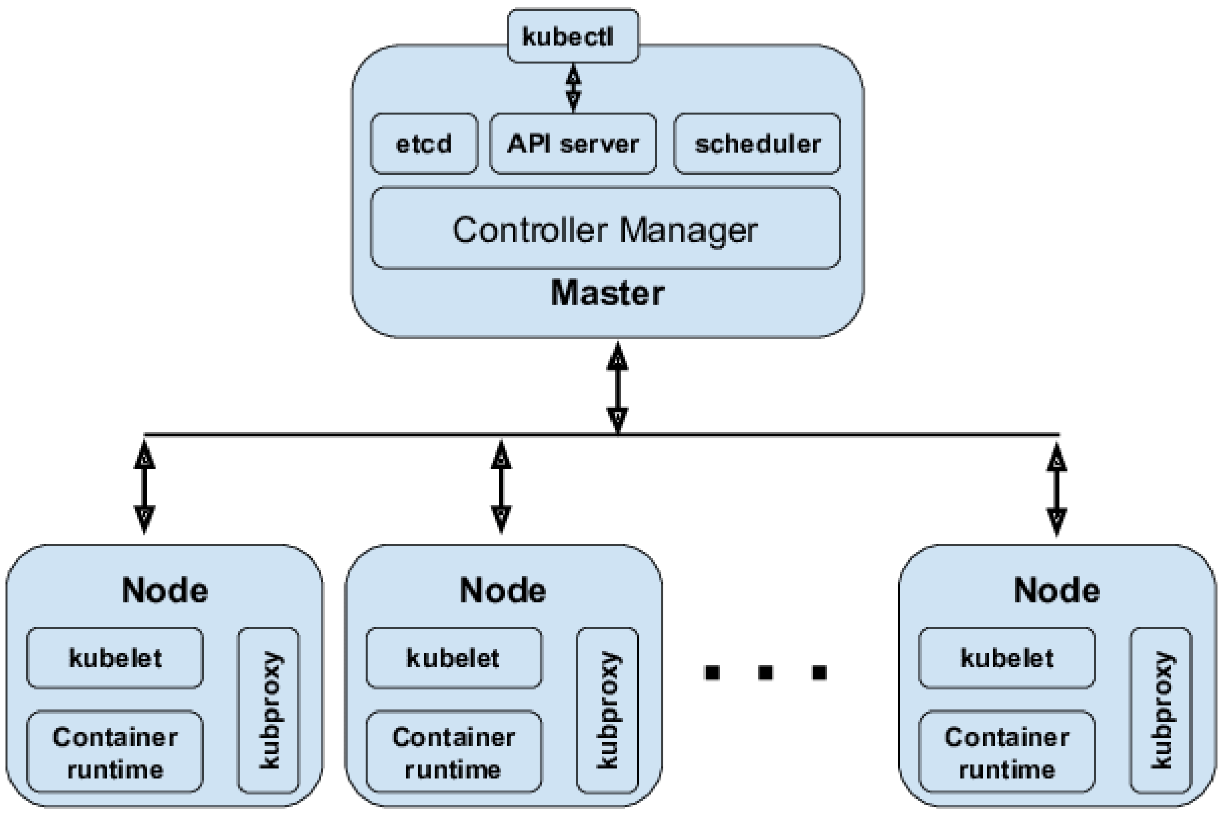


Figure 2.3: Kubernetes components [13]

There are a variety of unknown terms in the figure above, but each plays a part in the Kubernetes platform. Firstly, the master node is, as the name suggests, the master of the cluster; it is responsible for managing the resources and scheduling where workloads should be run [11]. It is also called the control plane and is where all the control plane components of Kubernetes live. The control plane consists of:

- **kube-apiserver**: Exposes the Kubernetes API and is the main frontend for Kubernetes [11].
- **etcd**: Key value store for all clustered data [11].
- **kube-scheduler**: Selects where workloads should be ran based on a variety of parameters [11].
- **kube-controller-manager**: Contains different controllers such as: Node controller, Job controller, ServiceAccount controller and more [11].
- **kubectl**: CLI tool to communicate with the API server [11].

There can be multiple master nodes in a cluster, and it is encouraged to have multiple masters synced in case a master node goes down. The nodes in Figure 2.3 are the compute nodes where containers are run, although in Kubernetes the smallest component is called a pod. A pod is a group of containers with shared resources, including the configuration of how the containers should be run [3]. Some pods only have one container, while others can have multiple. Often there is a set "main" container, and the others are sidecar containers offering extended functionality to the main container. In addition, each compute node will need a container runtime, as explained in Figure 2.2. The kubelet in Figure 2.3 is responsible for running the containers within a pod [11]. Lastly, kubeproxy is the component that allows network communication within the Kubernetes cluster; it maintains network rules on each node [11].

2.2.1 Networking

The research conducted in this thesis will be directly impacted by how networking works within a Kubernetes cluster. As with the main Kubernetes components shown in Figure 2.3, Kubernetes networking is complex and consists of multiple components working in unison. The following figure gives a broad overview of container-to-container and pod-to-pod networking [14]:

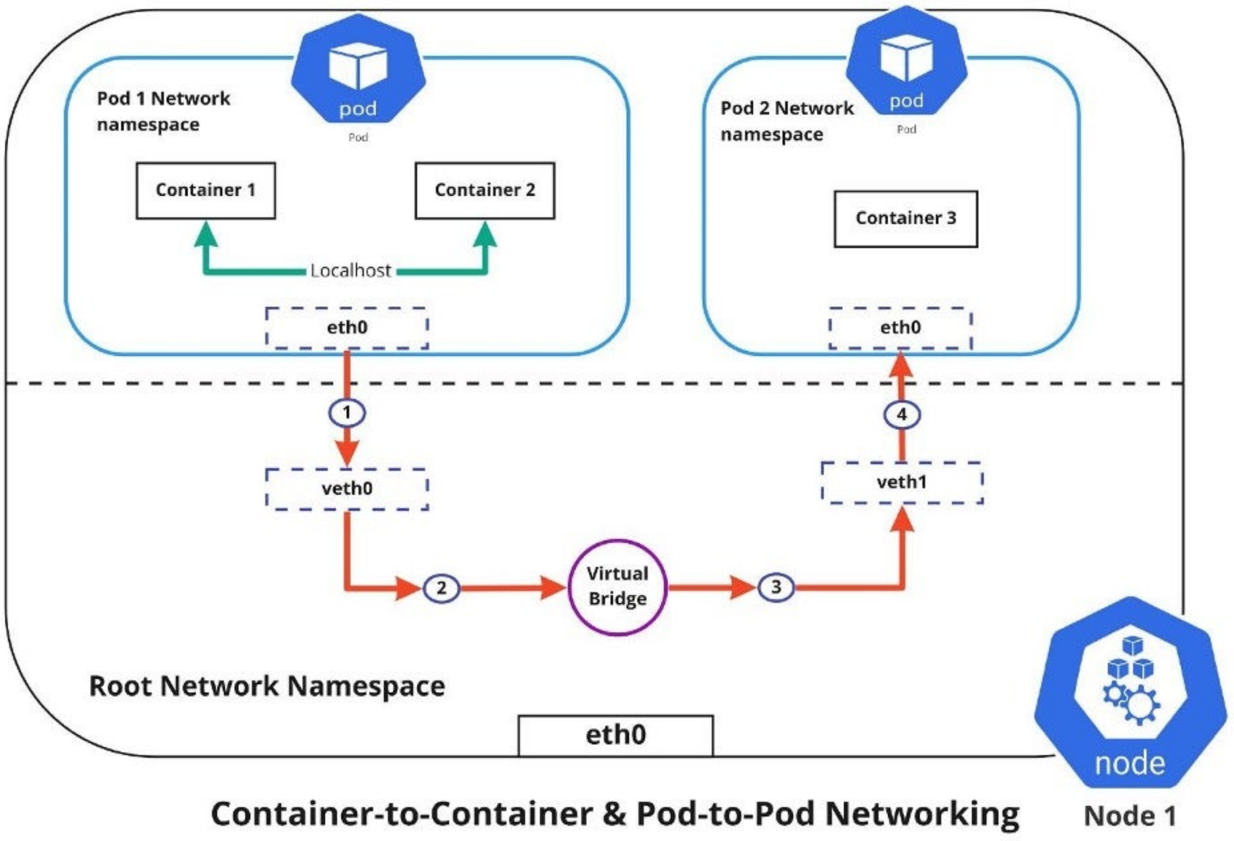


Figure 2.4: Container-to-container and pod-to-pod networking [14]

In Figure 2.4, there is an overview of a single compute node containing two pods. Pod 1 contains two containers, while pod 2 contains a single container. The figure shows the networking components of how container 1 communicated with container 2 and from pod 1 to pod 2. Inter-container communication is straightforward due to the shared network namespace attribute of a pod. All containers within a pod share the same localhost due to being in the same network namespace [15]. On the other hand, in order for pod 1 to communicate with pod 2, the traffic must reach the root network namespace of the compute node and have a way to know where pod 2 exists. This is achieved by each node receiving an IP range that it can allocate to pods [15]. Each pod will have a unique IP address. To route traffic through the root network namespace, a virtual network interface is connected to each pod from the root network namespace. This allows traffic to flow between the virtual interfaces of pod 1 and pod 2 using the Address Resolution Protocol (ARP). Through the virtual interfaces, eth0 of pod 1 can reach eth0 of pod 2 [15].

Pod to service networking

In order to continue the learning journey of Kubernetes networking, there is a new concept that must be introduced. The service object is a construct in Kubernetes to expose an application running in a single or multiple pods through a single IP address [15]. Given the volatile nature of pods, where they are created and destroyed following the load of the cluster, there needs to be a way for applications to not worry about the changing IP addresses of pods. This is where services come in. The Kubernetes documentation describes a service as a way to expose a group of pods over the network [15]. Below is an illustration of pod-to-service networking [14]:

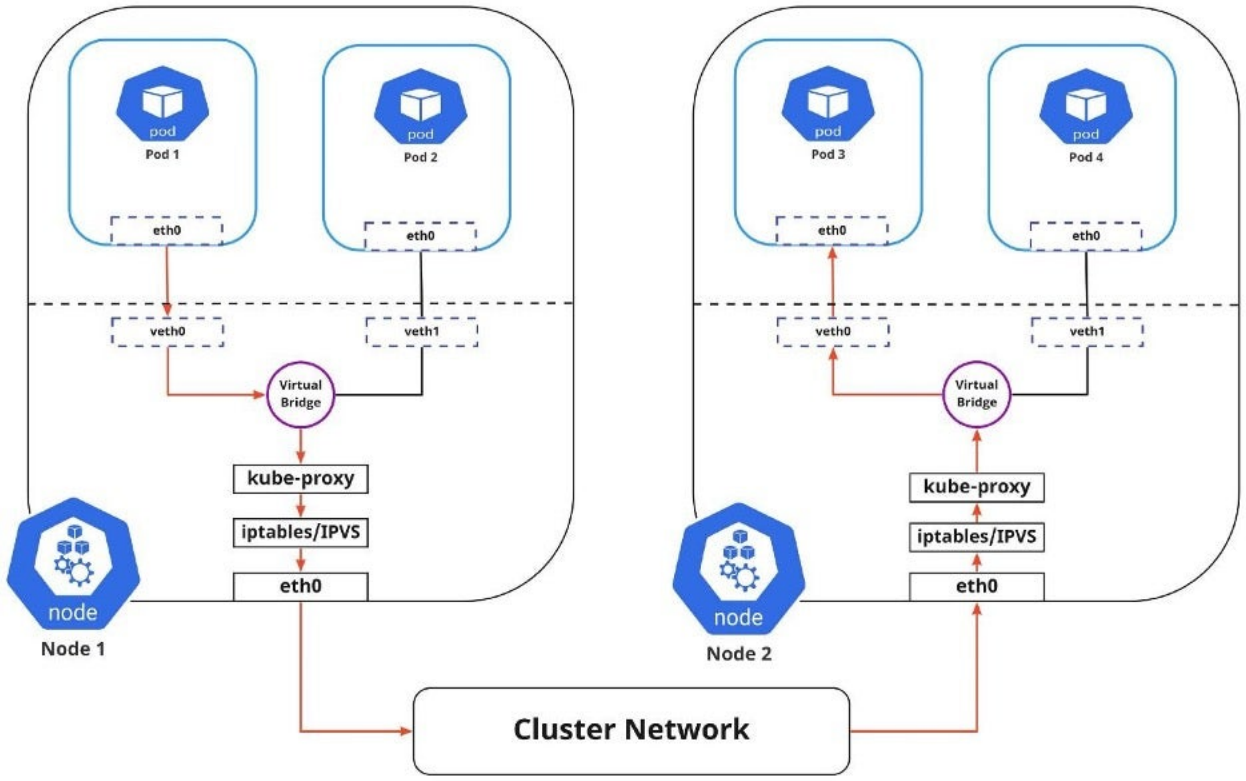


Figure 2.5: Pod to service networking [14]

In Figure 2.5, there are two compute nodes running two pods each. A service has been defined that groups pod 3 and pod 4 together as a backend for a virtual IP address in the service definition. When pod 1 wants to communicate with the virtual IP address for the service, traffic first reaches the virtual bridge which does not know about the service, then traffic is filtered by iptables rules created by the kube-proxy agent. These rules tell the traffic where to find the service with the given virtual IP. The traffic then travels to node 2 and another set of iptables rules load balance the traffic between the two backend pods 3 and 4. The kube-proxy agent maintains the list of where to send traffic for each service [15]. The reason IPVS is listed besides iptables, is that the cluster owner can choose which implementation to use. IPVS stands for IP virtual server and is a better performing version of iptables operating in the kernel space [14].

Internet to service networking

The next component of Kubernetes networking is how to route traffic from outside the cluster to applications running within. There are different service types available to expose a Kubernetes service, depending on how you want to publish the service. These include *ClusterIP*, *Loadbalancer* and *Nodeport* [15]. A service of type ClusterIP is published on an IP address only accessible within the cluster; this is the default behavior of a Kubernetes service. This service can then be accessed from the outside using an *ingress* object, which creates an entrypoint to the cluster with specific routing rules [15]. A more visual overview of the different service types can be seen in the figure below [14]:

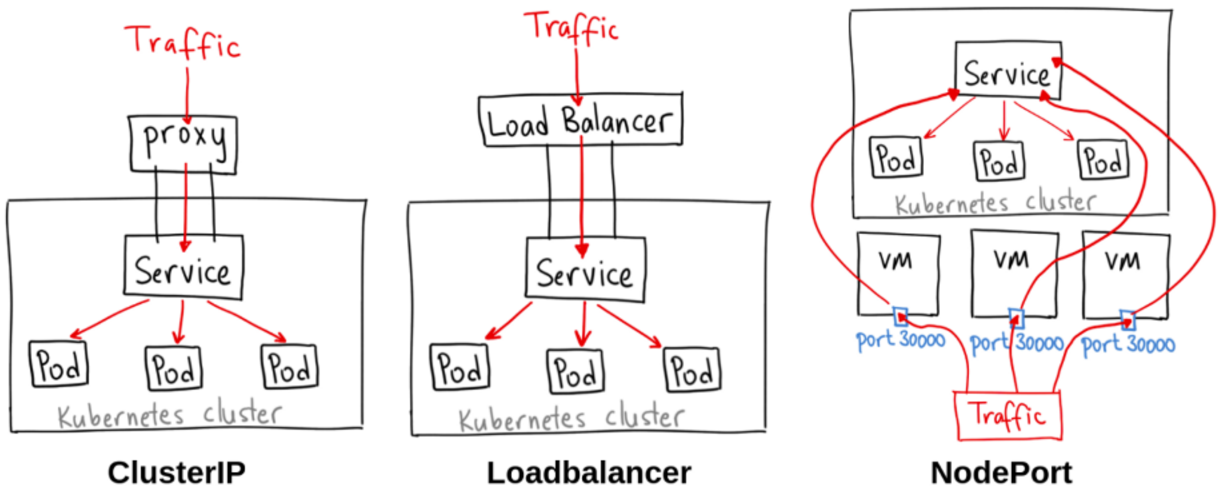


Figure 2.6: Service types of a Kubernetes service object [14]

As mentioned, a service of type ClusterIP will need another object, like an ingress proxy, for outside traffic to reach it. A service of type Loadbalancer will automatically provision a load balancer from your cloud provider. For example, a Kubernetes cluster in Amazon Web Services (AWS) will provision an AWS load balancer for the service. Lastly, a service of type NodePort will create a mapping between a service and a physical port on each Kubernetes node [15]. There are other ways to expose Kubernetes applications, but they utilize external addons that help in routing.

CNI Plugins

The Container Network Interface (CNI) is an integral part of networking in Kubernetes and is responsible for inserting network interfaces into the network namespace of containers [16]. The CNI project is made by the *Cloud Native Computing Foundation* and contains a variety of specifications and libraries for managing network interfaces in Windows and Linux containers [17]. There exists a large collection of CNI plugins created by the community based on the CNI specifications, which can be used in Kubernetes to extend or modify CNI functionality [16]. Even though many organizations utilize these CNI plugins for extended functionality, it is out of scope for this thesis, given that the network tapping sidecar container is made to work with any Kubernetes network plugin.

2.2.2 Monitoring

Kubernetes itself offers no recommendations for specifics on how to monitor the Kubernetes cluster and its resources, but recommends users investigate what type of monitoring solution is appropriate for their use case [18].

Prometheus and Grafana

Prometheus is one of the leading open-source applications for monitoring and alerting, working particularly well in unison with microservices and containers [19]. It uses a real-time time-series database to perform powerful queries and real-time alerting and pairs well with another open-source monitoring solution called Grafana, which can visualize the data from Prometheus in powerful dashboards [19]. Prometheus has good support for integrating with Kubernetes and supports monitoring everything from each compute node to individual containers within pods [19]. Prometheus does this by using a variety of exporters who gather metrics and expose them as an http endpoint that Prometheus scrapes on a set interval.

The Prometheus community maintains a project called the Kubernetes Prometheus Stack which includes an all-you-need package for a complete monitoring solution for Kubernetes. Including Prometheus, Alertmanager, Grafana and exporters, and pre-defined configuration and dashboards for Grafana visualizations [19]. The project allows for the simple deployment of a complete monitoring solution without deep knowledge or configuration of each component. An example of a Grafana visualization for the CPU usage of a single Kubernetes compute node can be viewed below:

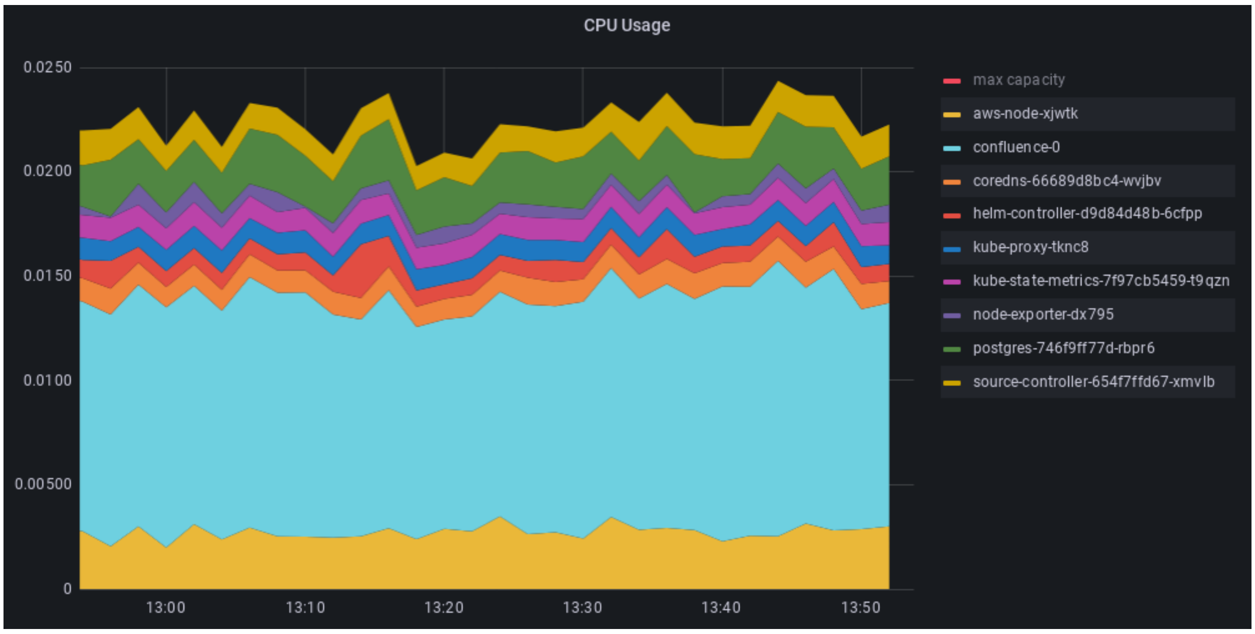


Figure 2.7: Grafana dashboard cpu usage of Kubernetes node

The labels on the right of the graph are the names of Kubernetes pods running on this particular node. Some other metrics that can be viewed with the monitoring stack are memory, network, storage, and Kubernetes internal metrics.

2.2.3 Resource allocation and limitation

The Kubernetes documentation recommends that all containers that are deployed to the cluster contain specifications about resource requests and limits. These two parameters tell Kubernetes how many resources the container needs to start and what limitations it should enforce [20]. These specifications will be used to ensure that the network tapping sidecar container will not use up the resources of other containers running on the same node.

2.3 Amazon Web Services

Amazon Web Services (AWS) started as a side company to the then e-commerce company Amazon in 2002 and has since grown to be the world market leader in the cloud market [21]. According to recent statistics AWS holds a 34% of the global cloud market, with industry competitor Microsoft Azure following behind with 21% [22]. A reason for this market dominance could be the plethora of different services that AWS offers through its cloud services, with over *"200 fully featured services from data centers globally"* [23]. This makes AWS have more services than any other cloud provider and makes their system extremely flexible and suited for complex use cases. Though this highly flexible environment makes for steep learning curves for people not familiar with cloud services and secure configuration of them.

2.3.1 Elastic Kubernetes Service

One of the services that AWS provides is the Elastic Kubernetes Service (EKS) which provides a managed Kubernetes cluster where customers do not need to worry about maintaining the infrastructure behind the cluster such as nodes and networking [24]. Utilizing the vast network infrastructure of AWS, an EKS cluster can be scaled to any size and maintain an availability that is hard to match with on-premise infrastructure. Although the EKS service utilizes another AWS service to provide the compute nodes for the Kubernetes cluster. AWS Elastic Cloud Compute (EC2) is used as virtual machines for the nodes. EC2 provides simple and scalable virtual machines in the cloud, and these are managed as cluster nodes through EKS. In addition, all these componenets are connected through yet another AWS service, AWS Virtual Private Cloud (VPC). AWS VPC gives the customer full control of a virtual private network in the cloud, where subnets can be created in different availability zones [25]. Then, EC2 instances can be connected to the subnets within the VPC. Next, rules can be defined for how the different subnets communicate with each other across availability zones or regions [25]. Detailed information about how these services are configured and used is beyond the scope of this thesis, and could even be an entire thesis in itself.

2.3.2 Identity Access Management

An essential part that touches everything managed within AWS is the Identity Access Management (IAM) service. IAM allows users to specify access control rules for every resource within AWS. Including who can access what resources [26].

2.4 IDS monitoring

An Intrusion Detection System's (IDS) objective is to monitor and detect malicious activity. There are different kinds of IDSs, some are network-based and others are anomaly-based [1]. An illustrative figure of a network-based IDS can be viewed below [27]:

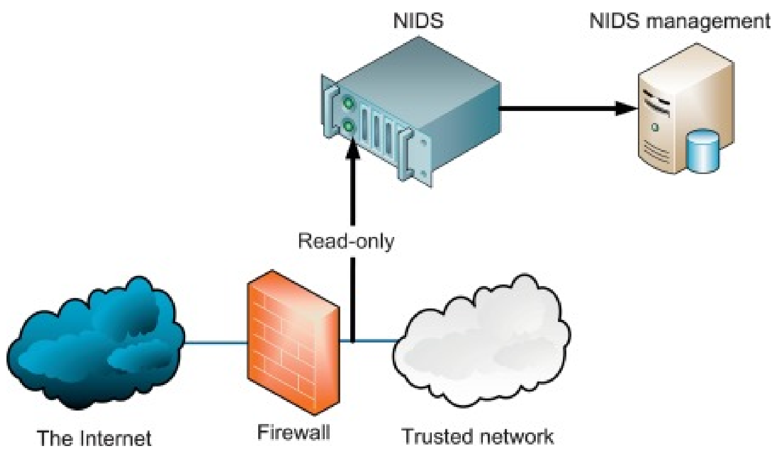


Figure 2.8: Overview of a network-based intrusion detection system [27]

The purpose of the IDS device is to passively monitor the network traffic without impacting the system being monitored. It will then generate alerts when malicious traffic is matched, it does this by using signatures [28]. A collection of known malicious traffic is analyzed and sorted into signatures that can be deployed in IDS devices to match similar traffic, but this requires someone else to make a signature first [28]. Therefore, this method of traffic analysis does not work well against new undetected threats. Though, some IDS systems can be set up to capture and store all the network traffic that it analyzes, even if there is no match. This is useful when performing incident response after an attack because old traffic can be analyzed to detect where an attacker might have had lateral movement before being found. This thesis will not focus on how the IDS is set up, but more on how the IDS device will receive the traffic when inside Kubernetes in a cloud environment. Where it is impossible to physically place your own device within the cloud provider's network.

2.5 VXLAN

VXLAN stands for Virtual eXtensible Local Area Network and is used to encapsulate network traffic at layer 2 in the OSI model over a layer 3 network [29]. Typically, layer 2 traffic only travels within a given Local Area Network (LAN) and is therefore restricted. However, VXLAN gives the ability to tunnel this layer 2 traffic on top of the already physical LAN network [29]. It was created out of the need for data center providers to have rapidly scaling network segmentation that quickly outgrew the 4096 number limit of traditional Virtual Local Area Networks (VLAN). With the VXLAN technology it is possible to create up to 16 million VXLAN's [29]. In detail, the layer 2 ethernet frames from the LAN are encapsulated as layer 3 UDP packets and are tagged with a VXLAN identifier (VNI) to segment the traffic. Lastly, VXLAN can be implemented in both hardware and software depending on the device's capabilities that will be communicating over the VXLAN.

2.6 Infrastructure

To conduct the research of this thesis there was a need for a way to manage and configure the testing infrastructure in a manageable way. The applications chosen for this task were chosen based on feedback and advice from the thesis supervisors and co-students.

2.6.1 Terraform

Terraform is a tool that allows automated configuration of infrastructure resources using a JSON-like language called the Hashicorp Configuration Language (HCL) [30]. It is an *"infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share."* [30]. This is achieved through the APIs of the infrastructure that is going to be managed, though the APIs are not accessed directly, but through a provider. A provider defines how the Terraform configuration is going to be translated into API calls to generate the defined resources [30]. An overview of the Terraform workflow can be viewed below [30]:

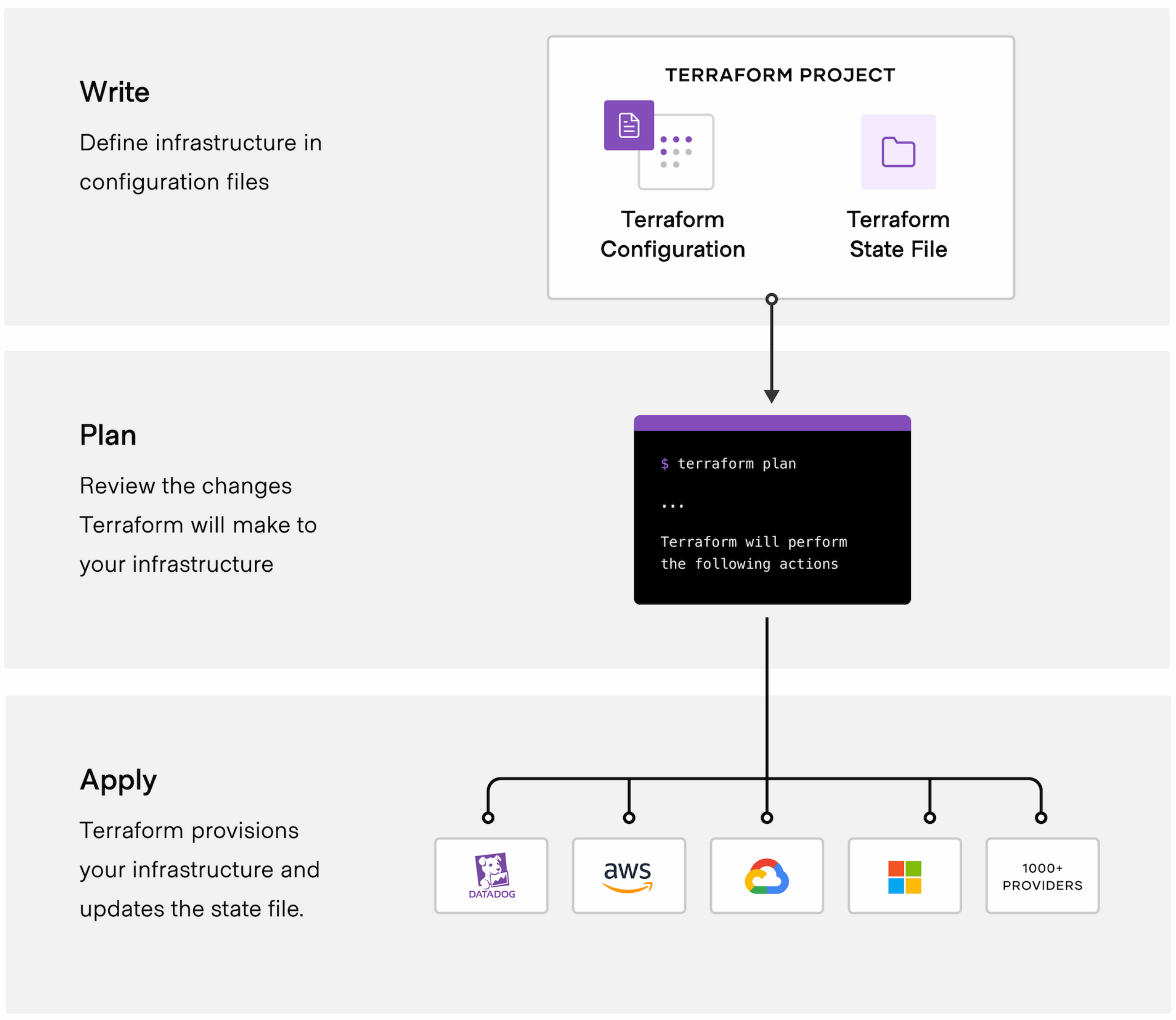

Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
$ terraform plan
...
Terraform will perform the following actions
Apply
Terraform provisions your infrastructure and updates the state file.
DATADOG
aws
1000+ PROVIDERS

Figure 2.9: Overview of Terraform workflow [30]

The desired resources are defined in Terraform configuration files before executing the *plan* command which utilize the chosen provider and generates a list of what resources are going to be created, added, or destroyed [30]. This plan ensures that the user knows what the configuration will do. The *apply* command executes the plan and generates the configured resources using the associated provider. The Terraform community maintains a list of over 1000 providers for different cloud providers and on-prem services [30].

2.6.2 Fluxcd

As a part of the process to automate much of the lab setup for the thesis, Fluxcd was chosen as a gitops tool to ease deployment of resources to the Kubernetes cluster. Fluxcd is a gitops tool made to sync Kubernetes manifests from git sources and apply them to a Kubernetes cluster [31]. In this way, manifests can be checked into version control and users will always know what version of manifests are deployed to their cluster.

The core concepts of Fluxcd are as follows:

Concept	**Description**
Gitops	A specialized way of organizing and managing your infrastructure through the use of declarative configuration and version control [32]. Infrastructure and applications are defined in a structured manner and pushed to a Git repository.Then an automated process ensures that this declared state is applied to the target environment [32].
Sources	A source is, as the name implies, a source for a repository containing declarative configuration files [32]. As well as the credentials needed to access the given source repository. These sources are checked on an interval for changes to be applied to the target environment [32]. An example of such a source file can be viewed in listing 21 in appendix B.
Reconciliation	Reconciliation is the action of applying the state declared in a ”source” to the target environment, and making sure that the state stays synchronized between the live environment and the declared environment [32].
Kustomization	The flux kustomize resource is a Kubernetes custom resource that acts as a collection of Kubernetes resources, that is to be applied to the cluster [32]. This also runs on an interval, meaning that changes are reverted to what is declared in the ”sources” every 5 min, but you can pause the reconciliation to stop this [32].

Table 2.2: Core concepts in Fluxcd [32]

Fluxcd includes a variety of extra features, but describing all of those is out of the scope of this thesis.

2.6.3 Ansible

Ansible is an application that can be used to automate or configure just about anything [33]. It is open-source and made by Red Hat. Some examples of what Ansible can do are configuration of systems, deployment of software, and orchestration of multi-step deployments and system updates [33]. The application uses SSH for communication with managed systems and uses a syntax similar to YAML, making it easy for humans to read. One of the main use cases with Ansible is to create playbooks which is a collection of tasks that are to be carried out by a given set of hosts. These tasks are module based and can do anything from a simple command to carrying out multi-stage deployment of advanced applications [33]. In Ansible, there exists a control node and a managed node or multiple. The control node holds the configuration of what Ansible is to do, in playbooks. The only setup needed is that the control node needs SSH access to the managed nodes [33]. An example task list can be viewed below:

```yaml
- name: Test on ec2
  ansible.builtin.command:
    cmd: touch gg.txt
  register: iperf_out
  changed_when: iperf_out.rc != 0
  delegate_to: ec2-pcap
  remote_user: ubuntu
```

Listing 1: Example Ansible task

The configuration above will execute the ”command” module on the host ec2-pcap with the parameter of `touch gg.txt` which will create the file `gg.txt`. The remote_user variable tells Ansible what user has SSH access to the remote host.

Chapter 3

Related Work

3.1 Traditional IDS monitoring

It is not in the scope of this thesis to go into depth on research done for traditional IDS monitoring, but the solution tested in this research is built upon the experience of TSOC as an IDS service provider.

3.2 IDS in the cloud

Several proposals of IDS monitoring in the cloud have been proposed over the years, using IDS monitoring together with anomaly-based detections to enable network-based threat detection in cloud environments [34]. Many of these proposals focus on implementing a solution that can monitor the entire cloud environment, and not just on a component such as Kubernetes. The bigger picture cloud threat model includes a substantial amount of parameters on top of an already complex Kubernetes system. An example model of the different components in a cloud environment can be viewed in the figure below [34]:

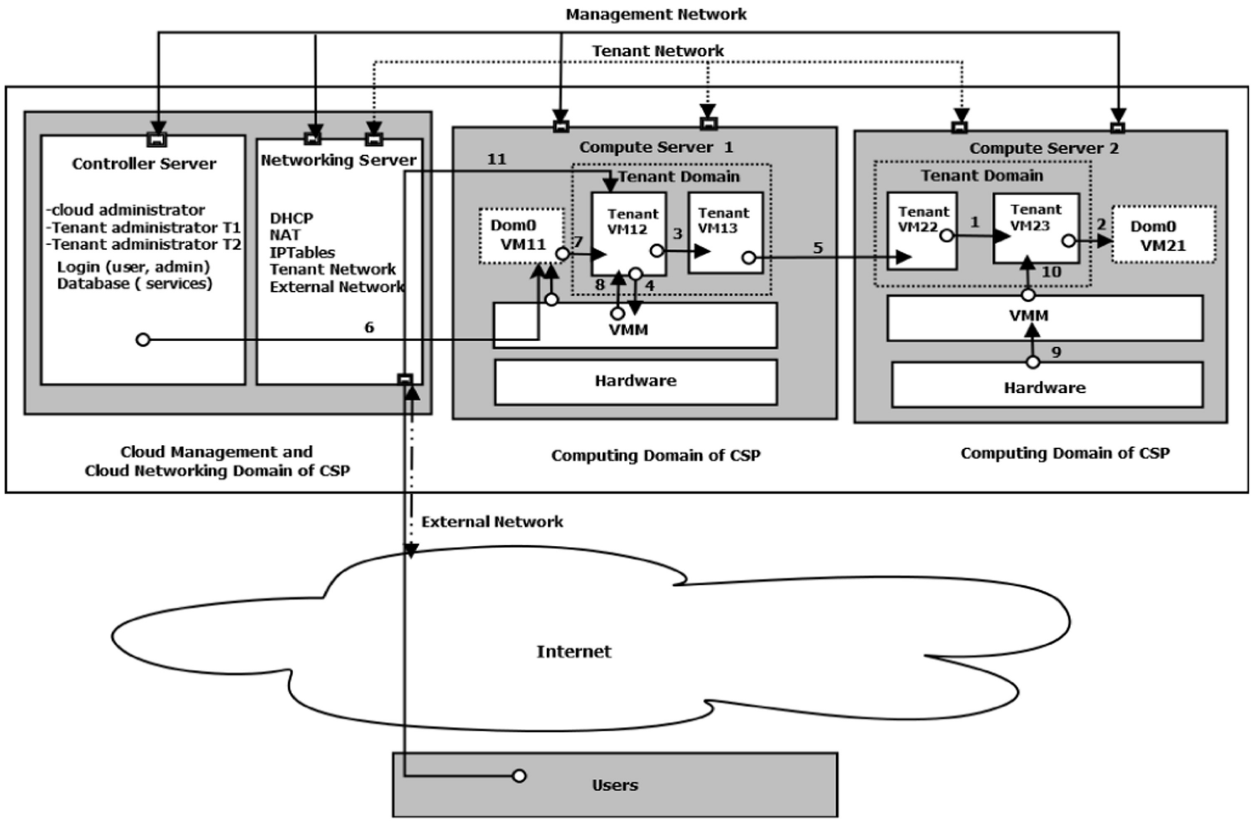


Figure 3.1: Example threat model of a standard cloud environment [34]

In figure 3.1 the tenant network is the network of a customer of a cloud service provider. What is called as the VPC when using AWS. Next, the virtual machines within the compute servers can be seen as the nodes in the Kubernetes cluster which are represented as AWS EC2 instances. The management network, controller and compute components are in the hands of the cloud service provider and are not something the tenant has the capacity to protect [34].

In cloud environments there are different deployment methods of IDS based on where the IDS device is placed. The IDS device can be placed within tenant virtual machines using either anomaly or rule based approaches. These are called knowledge-based approaches, because they are built upon our already gathered knowledge of attack patterns to detect. In addition, there exists techniques such as virtual machine introspection and hypervisor based introspection, they are outside the reach of a cloud tenant [34].

An overview of how a rule based, also called misuse detection, IDS system works can be viewed below [34]:

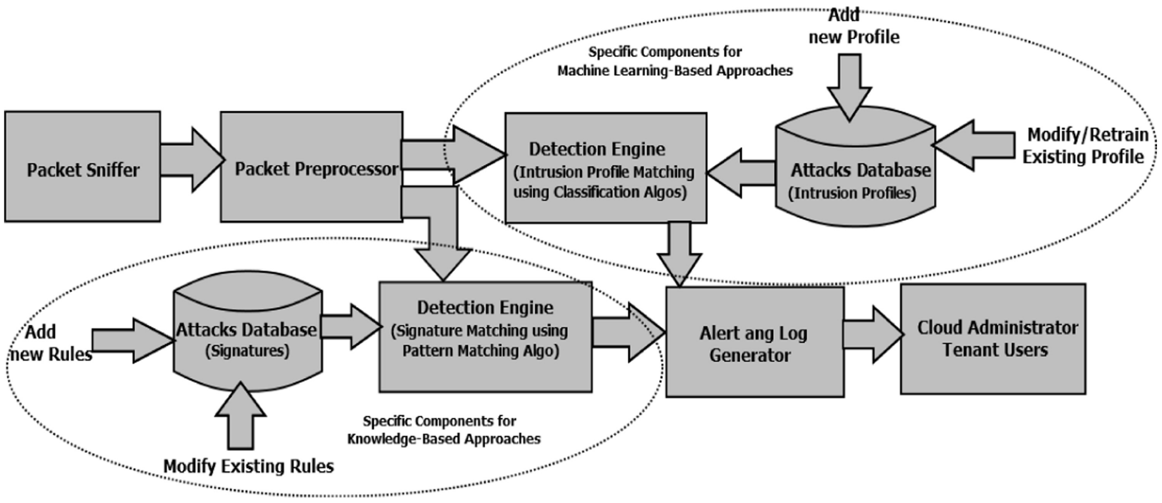

Add new Profile
Specific Components for Machine Learning-Based Approaches
Packet Sniffer
Packet Preprocessor
Detection Engine (Intrusion Profile Matching using Classification Algos)
Attacks Database (Intrusion Profiles)
Modify/Retrain Existing Profile
Add new Rules
Attacks Database (Signatures)
Detection Engine (Signature Matching using Pattern Matching Algo)
Alert ang Log Generator
Cloud Administrator Tenant Users
Modify Existing Rules
Specific Components for Knowledge-Based Approaches

Figure 3.2: Overview of misuse detection in networks [34]

In figure 3.2, the packet sniffer can be equated with the sidecar container which is used to sniff traffic within Kubernetes. The traffic is sniffed and then sent to a detection engine somewhere else which does the processing of the packets.

3.2.1 Sidecars for Network Monitoring in Kubernetes

The main focus area of this thesis is how to achieve the network monitoring mentioned previously inside containers running in Kubernetes. Achieving passive network monitoring within containers can be a challenging task, however the security solution company *Corelight* have published a series of posts exploring the different options organizations have to setup passive network monitoring of container workloads [35]. And it must be mentioned that without intra-container observability you will be left vulnerable to attacks such as remote code executions, command-and-control communication, lateral movement or file exfiltration [35]. These threats could be detected by a continuous detection system such as an IDS device.

The first problem that is encountered when trying to achieve network monitoring of containers is how to mirror the traffic to a suitable IDS sensor. In Kubernetes there are a few different ways to get access to the network traffic, which are listed below in rising difficulty [35]:

- **Container Network Interface (CNI)**: If you are using a CNI or network overlay that has native support for traffic mirroring its trivial to mirror traffic to a chosen destination. However, not all implementations of the CNI support traffic mirroring [35]. And forcing users to change their CNI or network overlay to a supported one is not a suitable alternative.
- **Container sidecars**: Container sidecars are small lightweight containers that are deployed in the same pod and exist in the same network namespace as each container [35]. This gives it easy access to all packets in and out of the container. Although this method requires creating a suitable sniffing program within the sidecar container. Another positive side of this solution is that it is completely agnostic to the Kubernetes environment such as CNI's and network overlays since it operates at the container level [35].

- **Host agents**: Host agents are deployed directly on the Kubernetes node and tap into the virtual network interface created for each namespace. This method also has observability down to the pod-to-pod level but not inside containers [35]. The downside to this method is that it requires special access to the node itself and some Kubernetes cloud deployments may not allow this.

A visualized overview of the above-mentioned methods can be viewed in the figure below [35]:

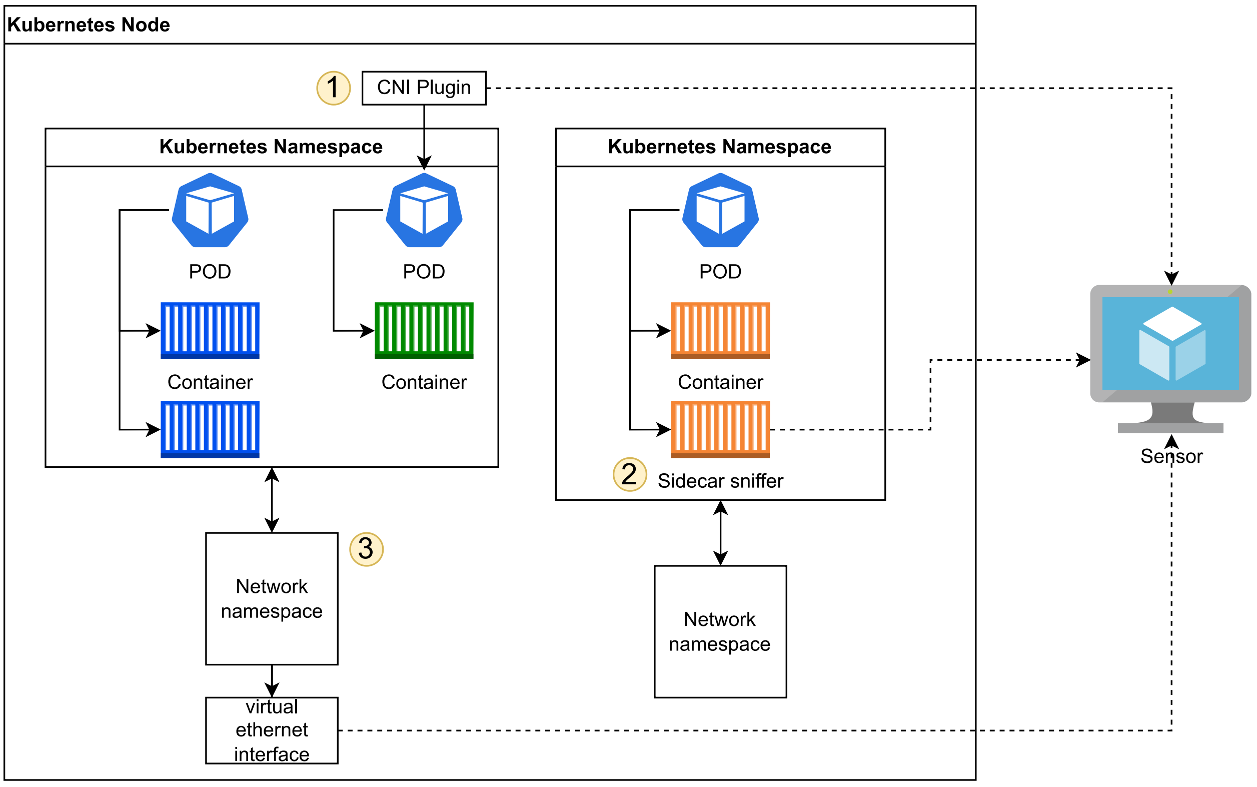


Figure 3.3: Overview of methods to monitor network traffic in Kubernetes

This thesis will focus on method 2, using container sidecars, though the Corelight sensor can be exchanged for any IDS capable device. A more detailed overview of how the container sidecar method works can be viewed in the figure below [36]:

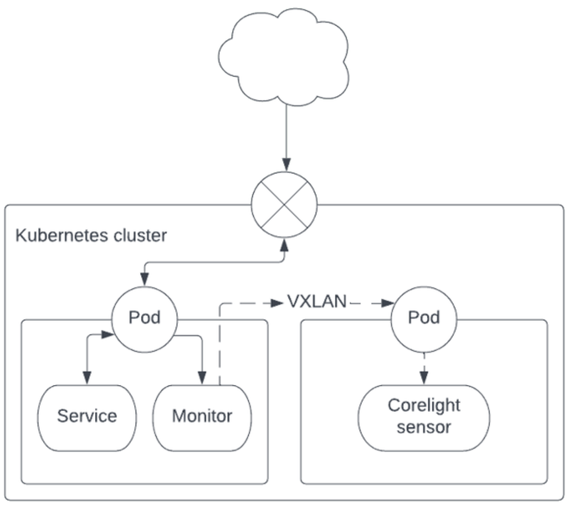


Figure 3.4: Overview of container sidecar [36]

It must be mentioned that the IDS sensor does not have to exist in a pod in the Kubernetes cluster, it can exist anywhere that the container sidecar can send the vxlan packets.

Chapter 4

Lab environment and testing approach

This chapter will give an overview of the cloud lab environment used to conduct the testing needed for answering the thesis questions. While also diving into how, and what type of tests are needed for the results to be relevant for TSOC's use case.

4.1 AWS

The lab environment will consist of a managed AWS EKS cluster that will run example applications with the sidecar container, as well as a collection of monitoring applications for the gathering of metrics. In addition, there must exist a machine that can act as the *sensor* which will receive the captured network traffic. Lastly, there must be a way to generate network traffic to watch the impact it will have on the sidecar container. A simple overview of the general setup can be viewed in the figure below:

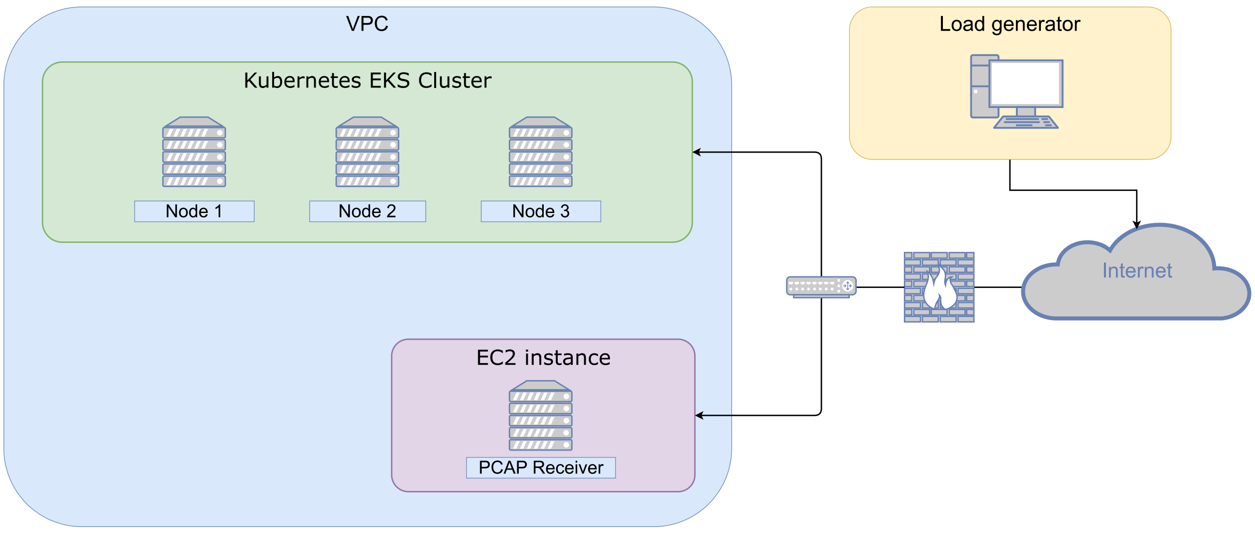


Figure 4.1: Overview of the AWS lab environment

In figure 4.1 there is a simplified overview of the components within the lab environment. However, to better understand the workflow that is happening beneath will require a venture deeper into the details. The figure shown below shows the individual components that are a part of the general workflow during usage of the sidecar container, though it does not reflect every use case.

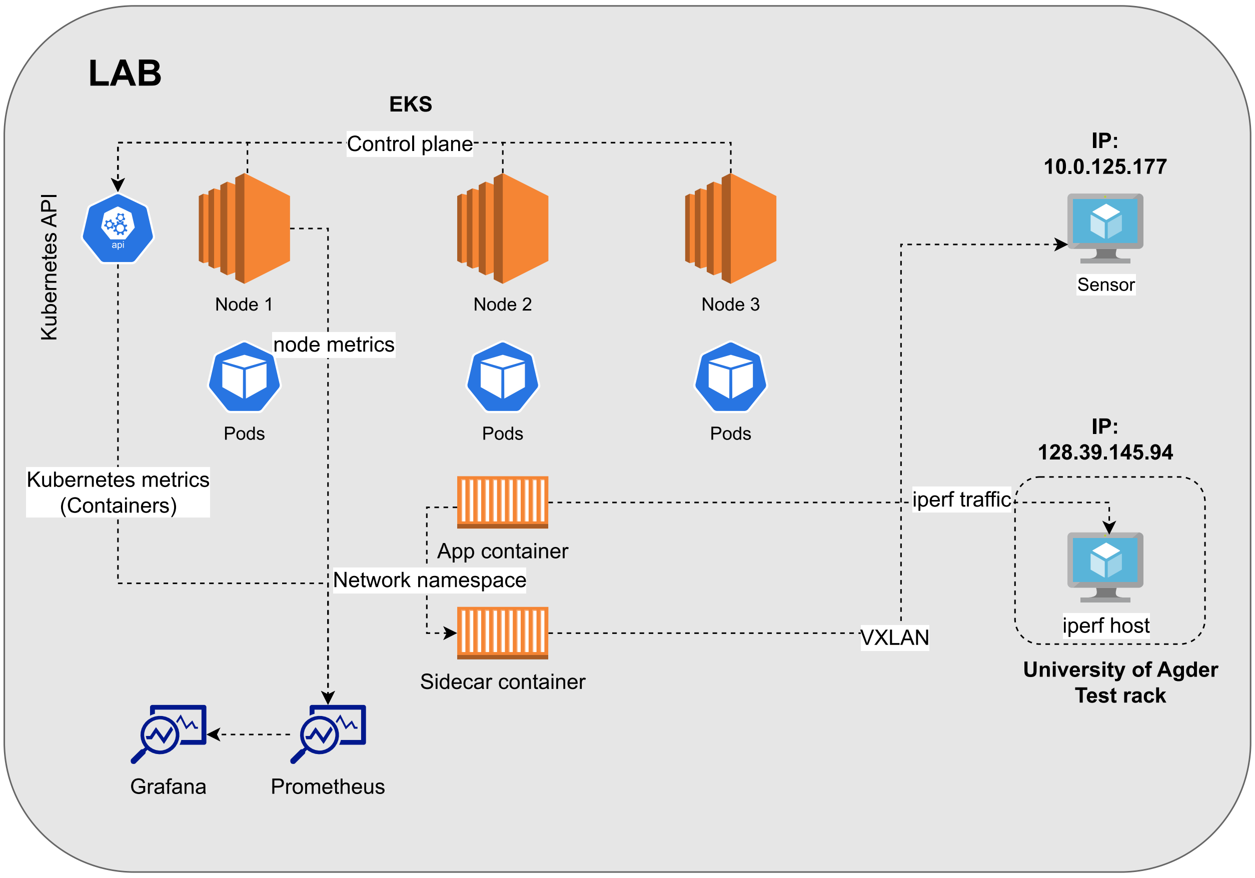


Figure 4.2: Detailed overview of lab environment

In the figure above most of the components of the lab are illustrated, although it must be mentioned that Grafana and Prometheus are themselves running as pods within the Kubernetes cluster. But the figure shows how Prometheus gathers performance metrics from both the Kubernetes API and directly from the compute nodes, allowing for a granular view of metrics in the associated Grafana dashboards. The figure 4.2 also depicts the traffic simulation between an application container and the `iperf host` which is just a normal virtual machine residing in a rack at University of Agder. This connection will simulate traffic load using iperf, which will be explained later. Lastly, the overview shows the VXLAN flow from the sidecar container to the sensor. Every component except for the `iperf host` exists within the AWS VPC.

Moreover, the hardware specification of the nodes used can be viewed in the table below:

EKS Node	
Instance Type:	t3.large
OS:	Amazon Linux 2
CPU:	2 vCPU Up to 3.1 GHz Intel Xeon Scalable processor
Memory:	8 GB
Network performance:	Up to 5 Gbps
Storage:	Amazon Elastic Block Storage

Table 4.1: t3.large node hardware specifications

The instance type viewed in the table is one of several instance types that can be chosen for compute nodes in a managed EKS cluster. The different types are for different needs and can vary in size from 1 GB of memory to several 100 GBs based on customer needs. Below is the same hardware specification table but for the UiA server:

UiA Machine	
OS:	Ubuntu Server
CPU:	2xIntel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Memory:	230 GB
Network Performance:	4x10GB SFP+
Storage:	10 TB Raid 5

Table 4.2: Hardware specifications of the UiA machine

As viewed above the hardware specification on the UiA machine are a huge overkill for the task, but it is sure that it will not be the bottleneck of the lab environment.

4.1.1 Terraform

Most of the infrastructure used in the lab environment was provisioned using Terraform to enable an easily reproducible setup and for easier management of taking the lab up and down when needed. The Terraform files used can be found in appendix A and contain the necessary configuration to generate a standard 3 node Kubernetes EKS cluster, and a standalone EC2 instance within the same VPC. The files also configure IP address whitelisting restricting ssh access of the EC2 instance from the outside world. Furthermore, the configuration was made easier by taking use of premade AWS modules for Terraform which are driven by community contribution and significantly simplifies how much Terraform code is needed to get a working EKS cluster. It is in listing 18 in appendix A that the most important parameters are set such as:

- **cluster_version: 1.23** Defines what version of Kubernetes should be used.
- **eks_managed_node_groups**: Object that contains information such as:
  - **instance_types: ["t3.large"]** What AWS instance type should be used for the nodes in the cluster.
  - **min_size, max_size, desired_size**: Specifies minimum and maximum number of nodes for scaling purposes, while also setting baseline number of nodes.
  - **ami_type: AL2** What operating system or *Amazon Machine Image* should be used on the nodes

Another important part of using Terraform is to manage the state files which tells Terraform what is the current status of the controlled environment. If changing computers and state files are not reachable, the new computer could make unexpected changes to the controlled environment. Therefore state files should be in an universally reachable place, defined through a backend such as AWS s3 block storage. The following lines from listing 14 in appendix A configures Terraform to store the state files inside AWS s3 and is reachable from anywhere, given that you have the needed AWS api access keys:

```
    backend "s3" {
    bucket = "ssl-tf-states"
    key    = "new/terraform.tfstate"
    region = "eu-west-1"
  }
```

Listing 2: Terraform AWS s3 state backend configuration

4.2 Kubernetes

As mentioned in the previous section the Kubernetes cluster consists of 3 compute nodes seperated into three different AWS availability zones in their *eu-west-1* data center in Ireland. This is for redundancy incase something occurs to a single AWS zone, and which means that the AWS VPC has the following configuration:

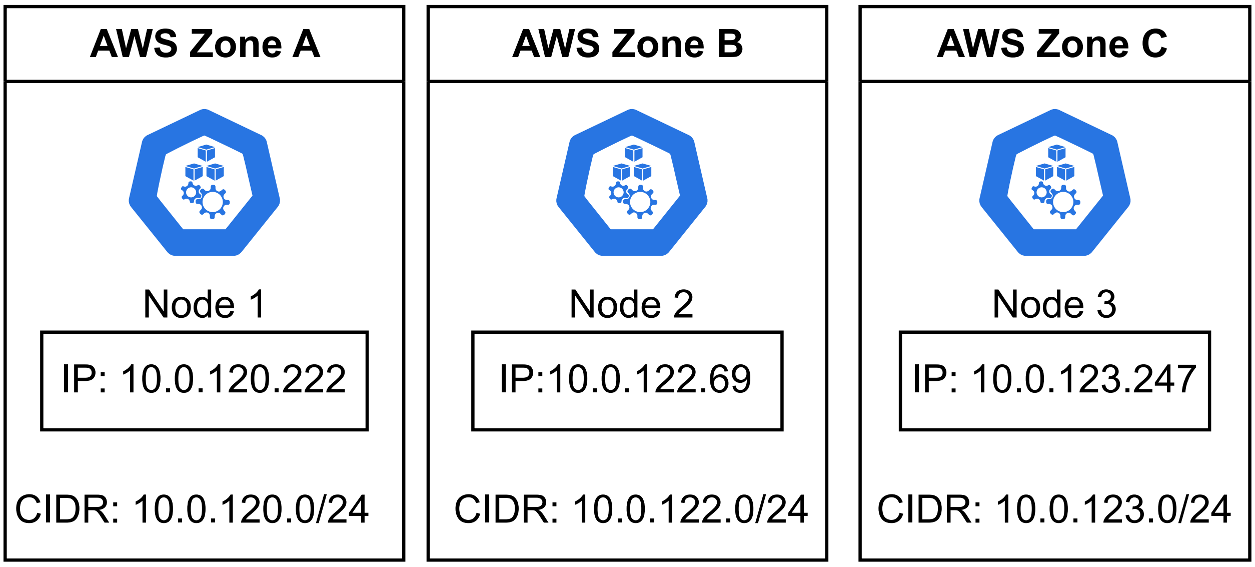


Figure 4.3: Overview of the Kubernetes nodes

The further configuration of the Kubernetes cluster is done through Fluxcd and its ability to orchestrate the deployment of manifests to the Kubernetes cluster. The manifests can be viewed in B.

In general application pods within Kubernetes are exposed first through a Kubernetes service, then through an ingress controller which allows the outside world to ask for applications running within the pods. This is the same process as described in figure 2.6 and is not shown as components in figure 4.2. The actual service type and ingress used can be wildly different from cluster to cluster and is dependent on the choice of the cluster owner, which in this case is the Nginx ingress controller. The initial test done in the lab environment is done using a Confluence instance and k6s, with data flow as follows:

There are many components involved in the data flow, but given that the sidecar container operates within the pod level it makes it possible for us to partially ignore what service type and ingress is used given that the sidecar will see the traffic either way.

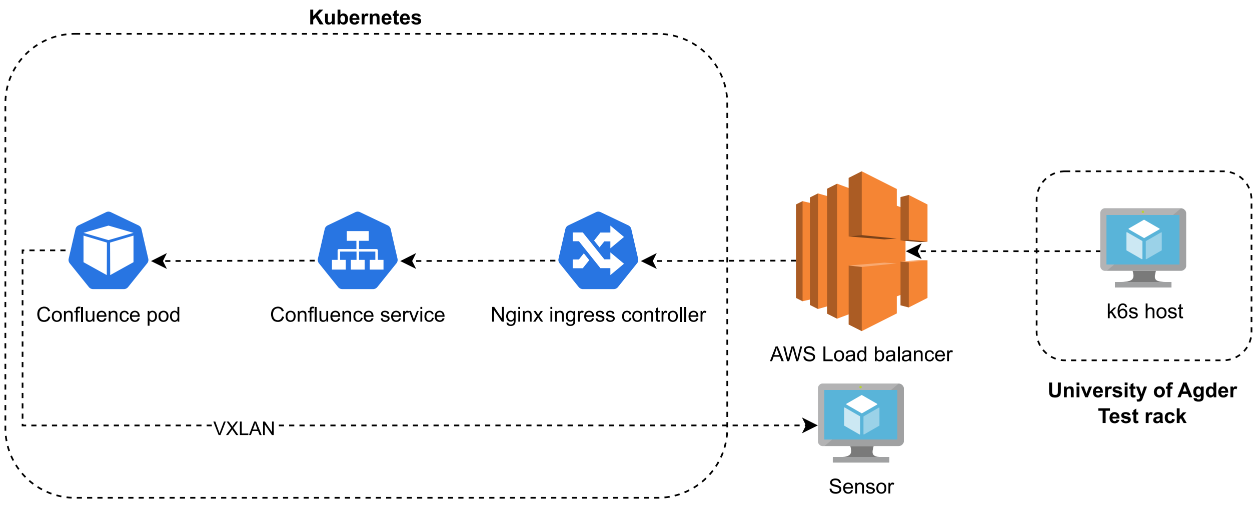


Figure 4.4: Data flow of k6s to Confluence pod

4.2.1 Fluxcd

The manifests used to configure Fluxcd can be found in appendix B and contains the configuration of where Fluxcd can locate Kubernetes manifests and sync them to the Kubernetes cluster. A kustomization in Fluxcd is simply a pointer to a folder where a collection of manifests can be found. One of the main components of this lab setup is the monitoring component, which was implemented through the *kube prometheus stack* project [37]. The project contains a premade collection of manifests for a complete monitoring solution within Kubernetes based on Prometheus and Grafana. Since the project has created a helm chart for installation, it was quite easy to use Fluxcd to sync the chart into the cluster. A helm chart is a recipe for installing a collection of Kubernetes manifests [38]. The main Fluxcd components used is the *HelmRelease* and *HelmRepository* files below, both of which are also found in appendix B:

```yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: prometheus-community
spec:
  interval: 120m
  type: default
  url: https://prometheus-community.github.io/helm-charts
```

Listing 3: Helm repository for kube prometheus stack

```yaml
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: kube-prometheus-stack
spec:
  interval: 5m
  chart:
    spec:
      version: "45.6.x"
      chart: kube-prometheus-stack
      sourceRef:
        kind: HelmRepository
        name: prometheus-community
      interval: 60m
  install:
    crds: Create
    remediation:
      retries: 2
  upgrade:
    crds: CreateReplace
  valuesFrom:
    - kind: ConfigMap
      name: prom-values
```

Listing 4: Helm release for kube prometheus stack

The *HelmRepository* just tells Fluxcd where to find the specified helm chart and how often to look for updates. Next, the HelmRelease specifies how Fluxcd should install the specified helm chart, with parameters such as version, update strategy and update interval. By default helm cannot update custom resources definitions, but Fluxcd can using *crds: Create*. At the bottom of the HelmRelease is also specification on what values should be applied to the helm chart. The full values can be viewed in appendix B.

Even though Fluxcd is mainly CLI based, there exists a UI interface that can be deployed to get a more visual approach to what is happening inside Fluxcd and the reconciliation status of the different applications. The files to configure this UI can be viewed in appendix B in listing 31.

4.2.2 The sidecar

The sidecar container that is to be capturing network traffic and sending it to the sensor is simply a python script written using socket programming. It opens a socket that listens to all traffic within the local network namespace of the pod that it is attached to and does some intelligent filtering. The filtering selects which packets should be packed in vxlan and sent to sensor and which packets to drop. For example, the sidecar should not record its own traffic to the sensor. The current iteration of vxlan.py in appendix C in listing 34 is based on corelight's example vxlan.py from their Kubernetes sidecar monitoring blog posts [36].

An example Kubernetes manifest running with the sidecar container can look as follows:

```yaml
        apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
      - image:
↪   108759891166.dkr.ecr.eu-west-1.amazonaws.com/sidecartap:vxlan
        name: vxlan
        imagePullPolicy: "Always"
        env:
        - name: VNI
          value: "499"
        - name: INTERFACE
          value: eth0
        - name: SENSOR
          value: 10.0.125.177 # 10.0.125.177 is gwlb endpoint
```

The above manifest will create a basic nginx container with an attached sidecar container that will capture the traffic and send it to the IP address set at *SENSOR* value. When this manifest has been deployed, the data flow will look as follows:

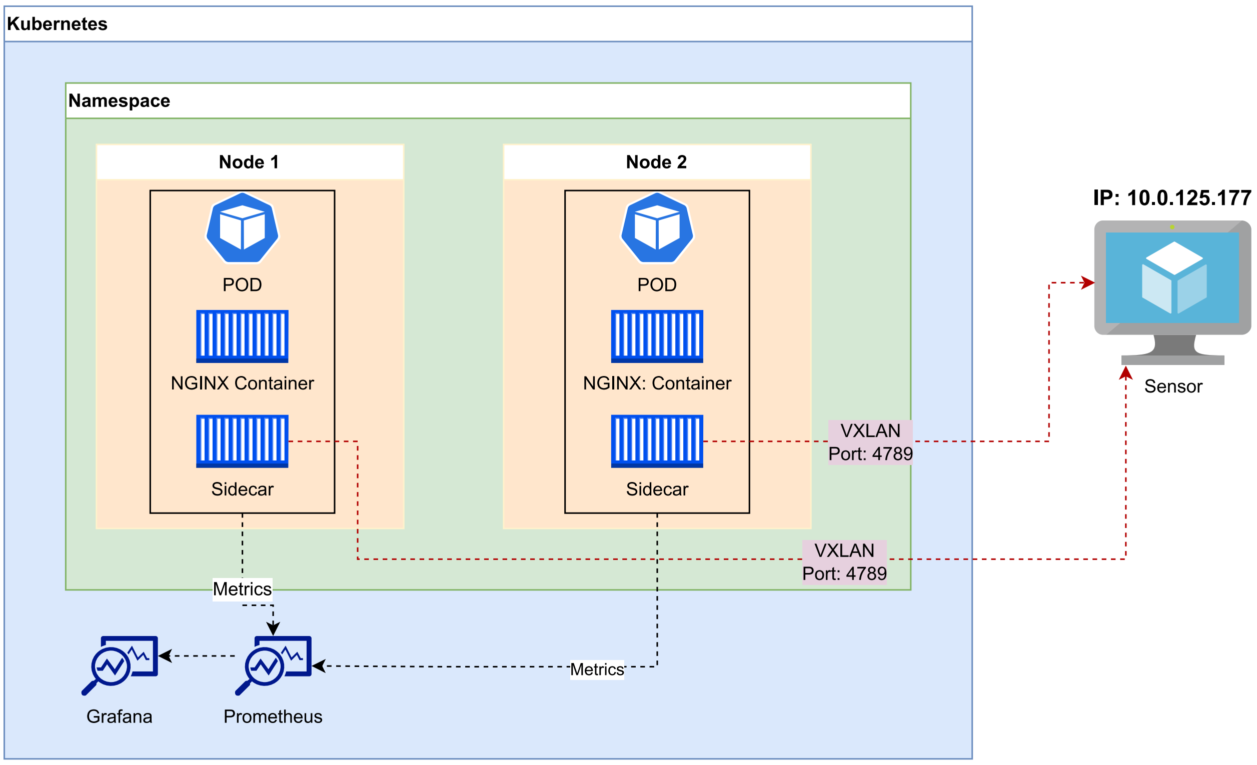


Figure 4.5: Overview of data flow in lab environment

The image above shows that the manifest has created two replicas of the given pod across two nodes, Node 1 and Node 2. In addition, the sidecar container has been set up to send captured traffic to IP `10.0.125.177` on port 4789 (default VXLAN port) which is a standalone EC2 instance. This data flow will be similar to other monitored containers, as the sidecar container is just an extra container within the pod. In addition, the metrics of both the containers, the pod and the node itself are monitored by Prometheus and Grafana.

vxlan.py

This section will describe how vxlan is used in the sidecar container, the script running inside the container is mostly based on Corelight's work as described in section 3. The following lines of python code is all that is needed to setup VXLAN:

```python
vxlan = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
...
# VXLAN header
vxlanHeader  = struct.pack('!L', 0x08000000)
vxlanHeader += struct.pack('!L', vni << 8)
...
vxlan.sendto(vxlanHeader+data, (sensorAddr, 4789))
```

Listing 5: VXLAN setup inside vxlan.py

The code above creates a new UDP socket that can be used to send packets through python. Since VXLAN is just a normal IP packet encapsulated with a header, an IP header with a given identifier is created to encapsulate the captured network data. A seen in the last line of code, the data is just appended to the VXLAN header and sent to the sensor at port 4789.

Next, the lines of code that do the actual network tapping is seen below:

```python
sniff = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3)) #
 ↪  ETH_P_ALL
sniff.bind((interface, 0))
...
while True:
    (data, _) = sniff.recvfrom(65535)

    sensorAddr = sensor.ip()
    if not sensorAddr:
        continue

    ipPacket = data[14:]
    ipHeaderLength = (struct.unpack('B', bytes([data[0]]))[0] & 0x0f) * 4

    srcIP = ipPacket[12:16]
    dstIP = ipPacket[16:20]
    protocol = ipPacket[9]

    if protocol in [6, 17] and len(ipPacket) >= ipHeaderLength+3:
        srcPort = struct.unpack('!H',
         ↪  ipPacket[ipHeaderLength+0:ipHeaderLength+2])[0]
        dstPort = struct.unpack('!H',
         ↪  ipPacket[ipHeaderLength+2:ipHeaderLength+4])[0]

        # Not our own traffic on UDP/4789 to sensor
        if protocol == 17 and dstPort == 4789 and dstIP == sensorAddr:
            continue

        if protocol == 17:
            data = ipPacket[ipHeaderLength+8:]
            if data[:len(vxlanHeader)] == vxlanHeader:
                continue

        if protocol == 17:
            continue

        print('Got %s byte%s from %s:%s to %s:%s proto %s (VXLAN %s->%s)' %
        ↪  (len(data), len(data) != 1 and 's' or '',
        ↪  socket.inet_ntoa(srcIP), srcPort, socket.inet_ntoa(dstIP),
        ↪  dstPort, protocol, socket.inet_ntoa(ipAddr), sensorAddr))
```

Listing 6: Python code to bind a sniffing interface socket

This code also creates a python socket, listening for any raw IP packets at the given interface. Given that the sidecar container shares a network namespace with the other containers in the pod, the above sniffing interface will observe all the traffic to the other containers. The rest of the python code is the running operation of the script. It operates in an eternal while loop and receives 65535 bytes before it unpacks the raw bytes and does some simple filtering before sending the packets to the sensor. For example, the code ignores its own UDP packets to the sensor. Lastly, the print statement is there for debugging purposes and to show current status of the script.

4.3 iperf and k6s

iperf and *k6s* are the tools that will be used to generate traffic that will test the sidecar container. There is no setup needed to utilize the iperf tool, it just needs to be installed on the *client* and a *server*. In this lab environment, the *client* will be an Ubuntu container running inside Kubernetes, that is monitored by the sidecar container. The *server* will be the a virtual machine that is placed in a rack at the University of Agder (UiA). The commands needed to get started with iperf is as simple as:

```
iperf3 -s # On the server, the UiA machine
iperf3 -c 128.39.145.94 # On the client, Ip of the UiA machine
```

This will generate a TCP connection between the server and the client that will try to utilize all the available bandwidth to test IP performance. However, it is unlikely that a real life application will utilize the entire bandwidth, therefore iperf allows us to specify the data rate that it will use in the test. By using `iperf -b bitrate/s` it lets us choose the data rate of the simulated network traffic. Then view the Grafana dashboards to see the performance metrics of the vxlan container. As well doing packet capture at the UiA machine to make comparisons between the observed traffic at sensor and what the UiA machine received.

In the other end k6s is a tool built to create performance test against web applications. In this lab it will be used to simulate users interacting with an Atlassian Confluence instance. And it is as simple to setup as iperf, it just needs to be installed then it can run custom Javascript files using the k6s binary. The code used can be viewed in appendix D in listing 36.

Ansible

The system explained above with iperf and packet capture has been automated using Ansible to minimize human error and for making it easier to reproduce. An Ansible playbook called `deploy_iperf.yml` has been created that manages everything from deploying the Kubernetes components to starting the iperf test on the UiA machine. The code for the playbook can be viewed below:

```yaml
---

- name: Ansible k8s playbook
  hosts: localhost
  roles:
    - iperf

- name: Start pcap
  hosts: ec2-pcap
  remote_user: ubuntu
  vars:
    cap_file: pcap_{{ ansible_date_time['epoch'] }}.pcap
    dest_folder: "/tmp"
    pod_ip: "{{ hostvars['localhost']['iperf_pod_list']['resources'][0]
['status']['podIP'] }}"
    pod_name: "{{ hostvars['localhost']['iperf_pod_list']['resources'][0]
['metadata']['name'] }}"
    dur_in_sec: 12
  roles:
    - tcpdump
```

Listing 7: Ansible playbook for load testing

The above playbook will first execute the iperf role which can be viewed in appendix F in listing 41. This role interacts with Kubernetes and deploys the Ubuntu container, along with the sidecar container, and installs the iperf application. Next, the pcap playbook is run with the role of tcpdump, which can be viewed in appendix F in listing 42. This role starts the iperf test with a given data rate and time against the UiA machine that is running an iperf server with `iperf -s`. The role also starts packet capture on the sensor and pulls the captured traffic file down for further analysis. Which also can be compared to the captured traffic from the UiA machine.

4.4 Performance monitoring

Performance monitoring is done by using Prometheus to collect metrics and visualizing them in Grafana dashboards for analysis. As mentioned in the problem statement one of the pain points that needs investigating is the cpu and memory footprint of the vxlan container when traffic hits the monitored container. If the sidecar container consumes too much resources of the Kubernetes node it may not be a viable network monitor alternative. Within the Kube Prometheus stack project there exists two premade dashboards that contain relevant visualizations for these parameters. And that is *Compute Resources / Node (Pods)* and *Compute Resources / Pod*. The first dashboard shows resource usage on a single node by pod, showing what pods are currently using in CPU and memory. The second dashboard shows resource usage by containers within a single pod, while also including network bandwidth and I/O operations. Below is an example image of CPU usage from each dashboard:

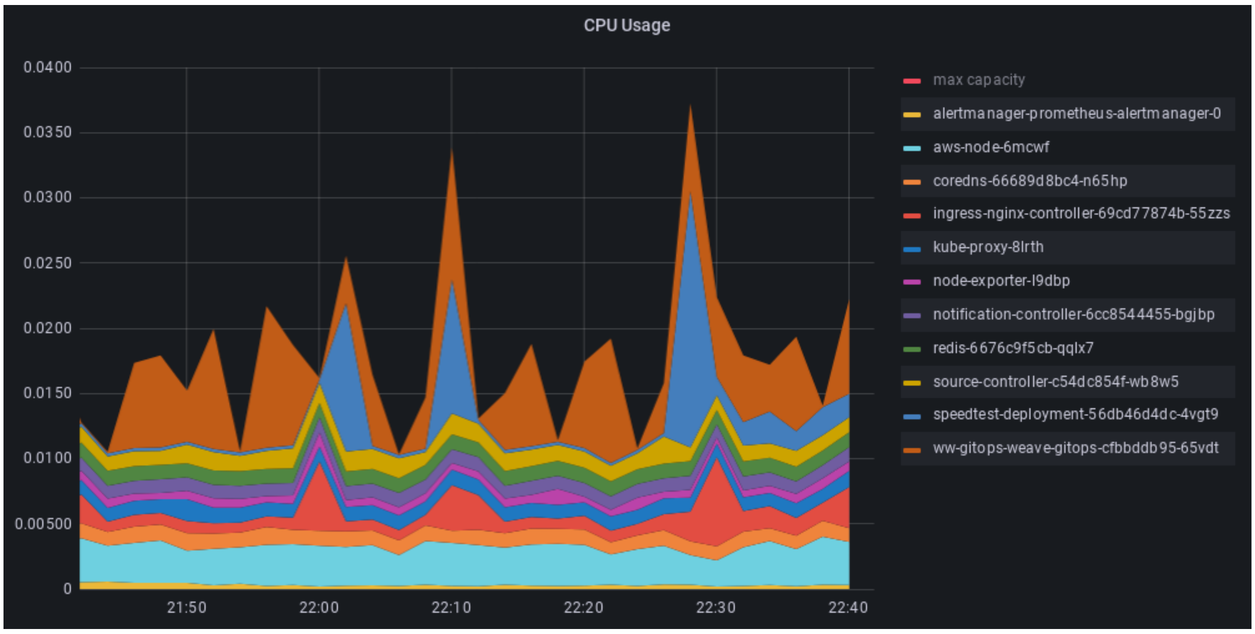


Figure 4.6: CPU usage per pod on a given node

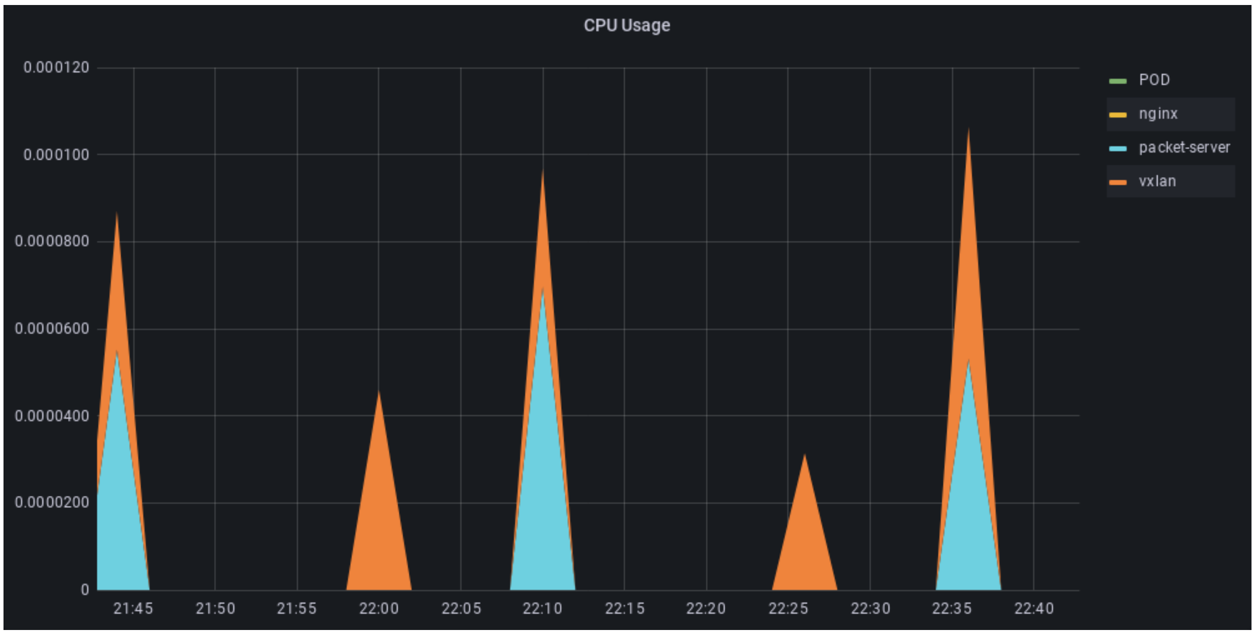


Figure 4.7: CPU usage per container within a single pod

Metric collection

The metric collection in this lab environment is facilitated by the Kube Prometheus stack, and is therefore built upon Prometheus gathering metrics both from the Kubernetes API and the compute nodes themselves. Therefore will the configuration values of Prometheus play a part in how the metrics are to be interpreted in Grafana. Since Grafana can only show what data that Prometheus provides. The configuration values of the Prometheus instance can be viewed below:

```yaml
prometheus:
  enabled: true
  prometheusSpec:
    replicas: 1
    replicaExternalLabelName: "replica"
    ruleSelectorNilUsesHelmValues: false
    serviceMonitorSelectorNilUsesHelmValues: false
    podMonitorSelectorNilUsesHelmValues: false
    probeSelectorNilUsesHelmValues: false
    retention: 7d
    scrapeInterval: 10s
    enableAdminAPI: true
    walCompression: true
```

Listing 8: Configuration values of Prometheus in Kube Prometheus Stack helm chart

The important parameter from this configuration is scrape interval since this number describes what will be the granularity of the data viewed in the Grafana dashboards. It means that every 10 seconds, Prometheus will scrape its configured targets such as the Kubelet API metrics, which expose the metrics of the running containers, such as the sidecar container.

By default, Grafana does not need any additional configuration when used in this deployment as it comes with a detailed set of dashboards covering all the metrics that are of interest in this project. Although, one must vary of staleness in Grafana, which is where Grafana will try to fill out a graph if it has not gotten data from Prometheus for some time.

4.5 Testing methodology

As previously mentioned in the introduction, there will be conducted a series of performance test against the sidecar to gather a grasp on the computational performance. The testing will consist of an initial k6s load test as well as a series of tests using iperf. The performance tests using iperf have been automated using Ansible and provides a stable testing platform with repeatability. Which gives time to observe CPU and memory metrics within the Grafana dashboards. Also, the use of iperf allows us to generate a connection to a remote host from a monitored container and simulate traffic at a given data rate and observe the sidecar metrics.

Chapter 5

Results

This chapter will summarize the findings of the performance tests done on the sidecar container. By looking at the sidecar performance both in speed tests and in simulated user traffic environments, we will get an initial grasp on the performance impact and footprint of the sidecar.

5.1 Initial performance test with k6s

5.1.1 k6s: Confluence

The configured Confluence instance in Kubernetes can be viewed in the manifest files in appendix B, which contains the necessary manifests to get Confluence running in Kubernetes and synced through Fluxcd. The first k6s test done against Confluence can be viewed below:

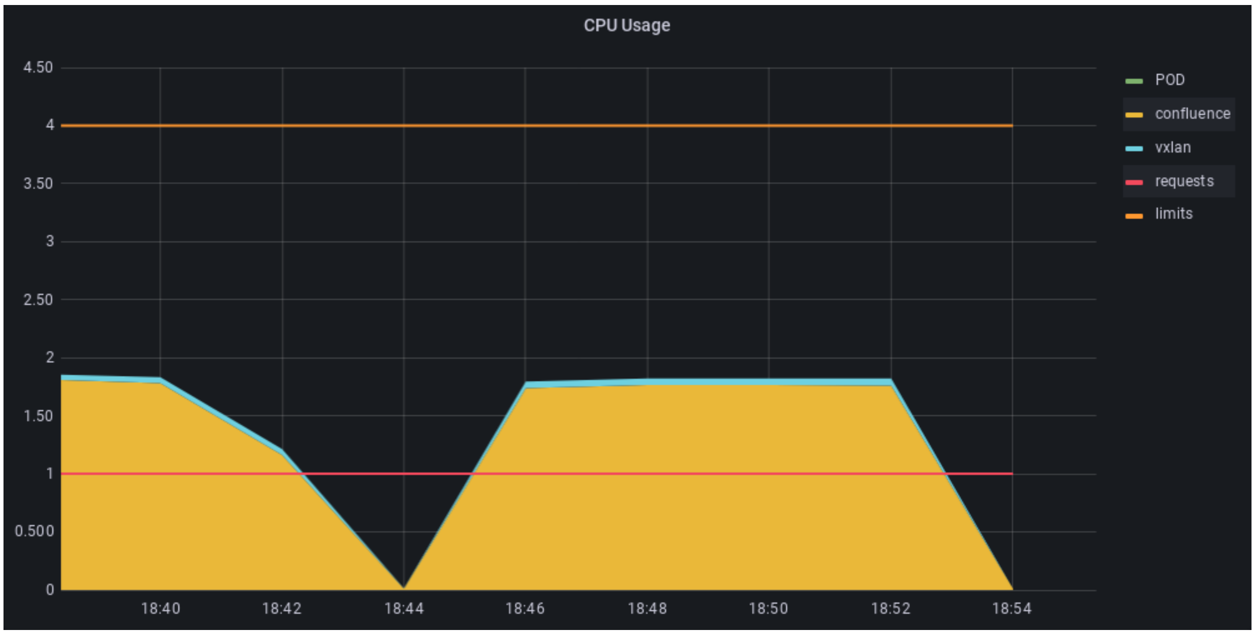


Figure 5.1: CPU usage within the Confluence pod

The graph shows the CPU usage within the Confluence pod, which contains two containers, vxlan and Confluence itself. It is the vxlan container CPU usage that is of interest here since it shows how it was impacted by the influx of user traffic generated by the k6s test. We can also take a look at the usage graphs of the Kubernetes node itself:

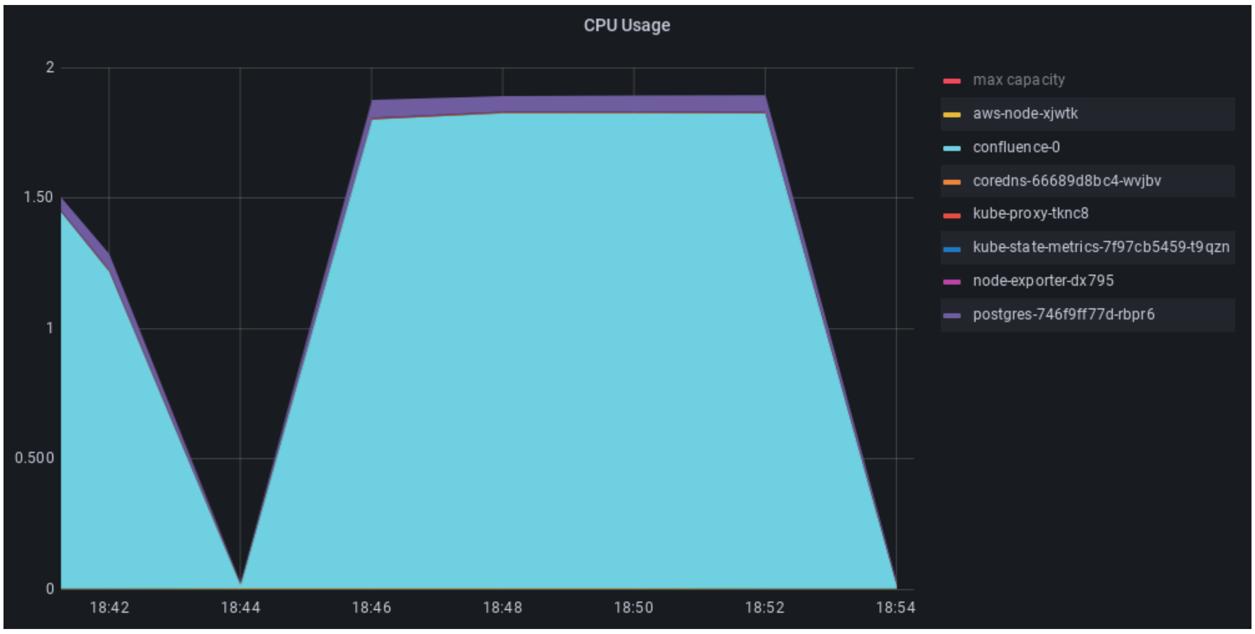


Figure 5.2: CPU usage viewed from the compute node

As seen in the images above, its the Confluence container itself that is consuming the most CPU, and the usage of the sidecar container is marginal in comparison. However, this data does not say much if we do not take a look at how much traffic was actually generated. Below is a graph of the network bandwidth usage during the test:

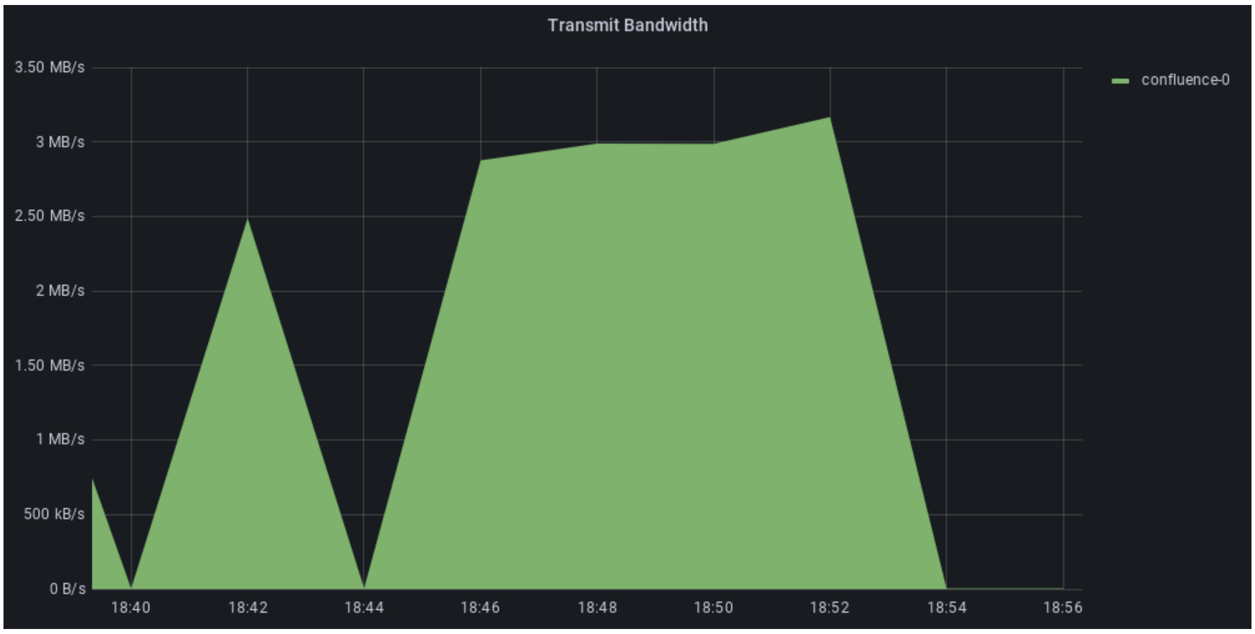


Figure 5.3: Network bandwidth transferred from Confluence pod

The traffic peaked at around `3 MB/s` which is a relatively small amount of traffic, and may give reason as to why the vxlan container did not struggle to capture this traffic. Below is also a graph of the memory usage of the Confluence pod:

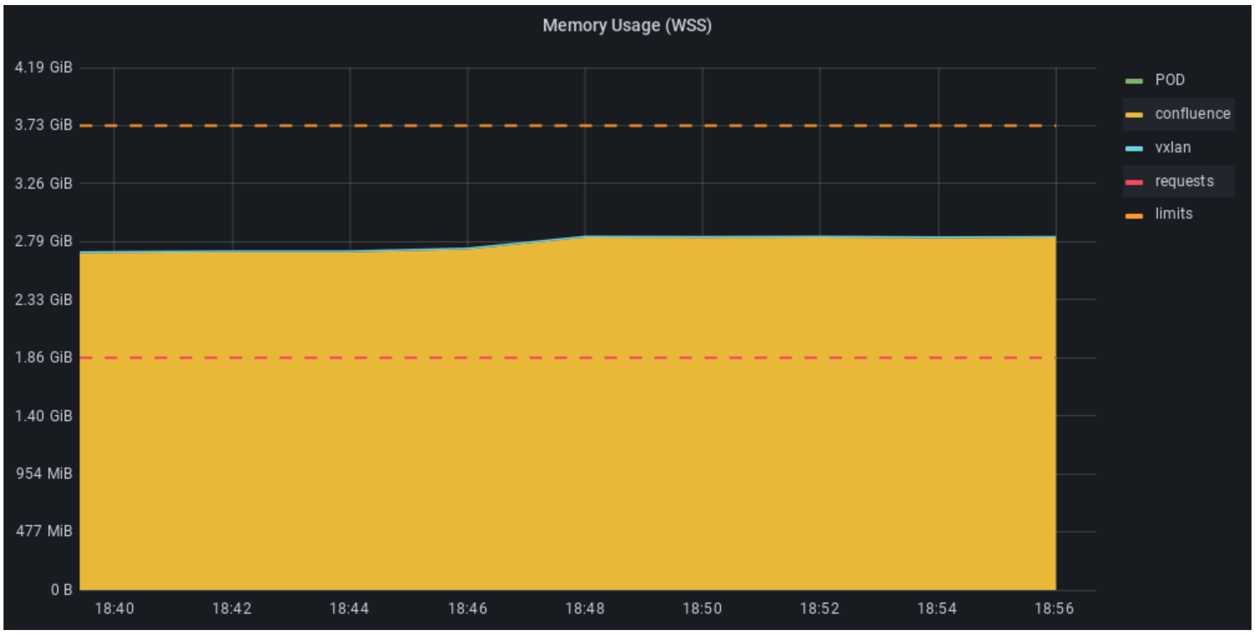
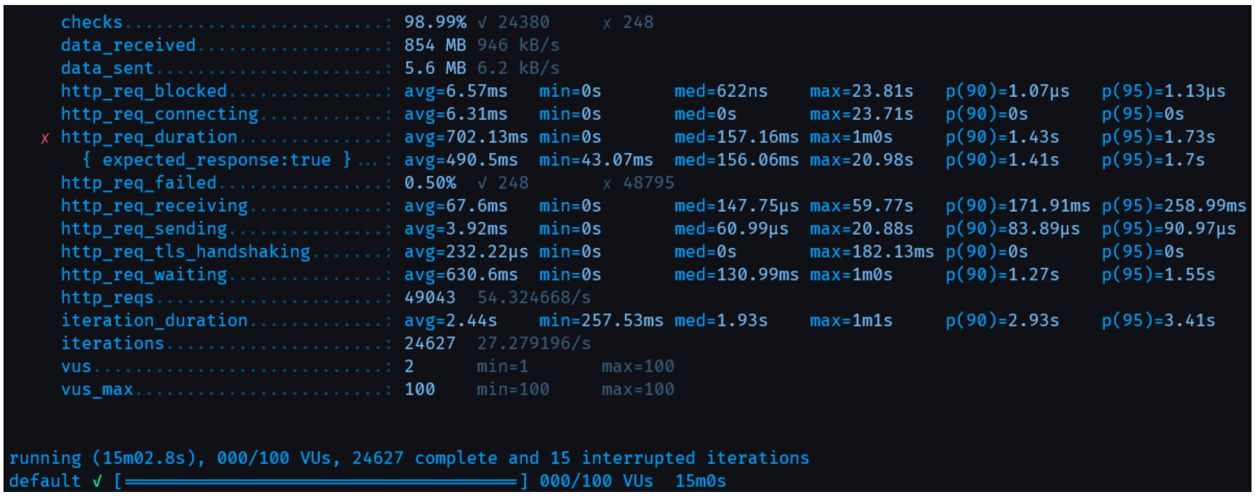


Figure 5.4: Caption

Given that Confluence is a Java application it uses quite a bit more memory than the vxlan container. The tiny blue line showing the vxlan memory usage peaks at around `15 MB` which is nothing compared to the almost `3 GB` used by Confluence. Lastly, the following image shows the result of the k6s test itself:



```
     checks.........................: 98.99% √ 24380      ✗ 248
     data_received..................: 854 MB 946 kB/s
     data_sent......................: 5.6 MB 6.2 kB/s
     http_req_blocked...............: avg=6.57ms    min=0s       med=622ns    max=23.81s    p(90)=1.07µs    p(95)=1.13µs
     http_req_connecting............: avg=6.31ms    min=0s       med=0s       max=23.71s    p(90)=0s        p(95)=0s
   ✗ http_req_duration..............: avg=702.13ms  min=0s       med=157.16ms max=1m0s      p(90)=1.43s     p(95)=1.73s
       { expected_response:true }...: avg=490.5ms   min=43.07ms  med=156.06ms max=20.98s    p(90)=1.41s     p(95)=1.7s
     http_req_failed................: 0.50%  √ 248        ✗ 48795
     http_req_receiving.............: avg=67.6ms    min=0s       med=147.75µs max=59.77s    p(90)=171.91ms  p(95)=258.99ms
     http_req_sending...............: avg=3.92ms    min=0s       med=60.99µs  max=20.88s    p(90)=83.89µs   p(95)=90.97µs
     http_req_tls_handshaking.......: avg=232.22µs  min=0s       med=0s       max=182.13ms  p(90)=0s        p(95)=0s
     http_req_waiting...............: avg=630.6ms   min=0s       med=130.99ms max=1m0s      p(90)=1.27s     p(95)=1.55s
     http_reqs......................: 49043  54.324668/s
     iteration_duration.............: avg=2.44s     min=257.53ms med=1.93s    max=1m1s      p(90)=2.93s     p(95)=3.41s
     iterations.....................: 24627  27.279196/s
     vus............................: 2      min=1        max=100
     vus_max........................: 100    min=100      max=100


running (15m02.8s), 000/100 VUs, 24627 complete and 15 interrupted iterations
default √ [======================================] 000/100 VUs  15m0s
```

Figure 5.5: Caption

The interesting stats here are that the test lasted for 15 minutes and that k6 received around `854 MB` of data. This can be compared to the data received by the sensor, which received also received around `850 MB` of traffic. Meaning that the user traffic was captured and sent to the sensor.

Preliminary summary

Based on the data above there is the following summary for the vxlan container after the initial k6s test:

Data Rate	CPU usage	MEM usage
3 MB/s	0,06	15 MB

Table 5.1: Sidecar container performance at 3 MB/s datarate

5.2 Speed test with iperf

The initial test with k6s showed that there is a need to generate more user traffic for the vxlan container to capture. This is where iperf comes in and allows for testing at different data rates. The testing will be conducted at the following data rates: `50Mb/s`, `100 Mb/s`, `250Mb/s` and `500Mb/s`. The time that the data rate will stay at the given data rate is set to 2 minutes to allow the metrics to populate Prometheus so that the Grafana dashboard will provide an appropriate estimation of the sidecar container metrics. The Grafana graphs below were gathered by viewing the pod that the Ansible automation deployed and the following output from Ansible:

```
TASK [tcpdump : Debug info] ok: [ec2-pcap] => {
    "hostvars['localhost']['iperf_pod_list']['resources'][0]['status']
['podIP']": "10.0.123.138"
}

TASK [tcpdump : Debug info2] ok: [ec2-pcap] => {
    "hostvars['localhost']['iperf_pod_list']['resources'][0]
['metadata']['name']": "iperf-test-1-6fbbbc9d68-wtgtc"
}
```

Listing 9: Ansible output for viewing pod

5.2.1 Data rate: 50Mb/s

Below is a summary given from the iperf application after the `50Mb/s` speed test is done:

```
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec   715 MBytes  50.0 Mbits/sec   28             sender
[  6]   0.00-120.04 sec   715 MBytes  50.0 Mbits/sec                  receiver
```

Listing 10: iperf results for 50mbs

This shows that `715MB` was transferred during the test at `50Mb/s`. The performance metrics of the vxlan container during this test can be viewed in the image below:

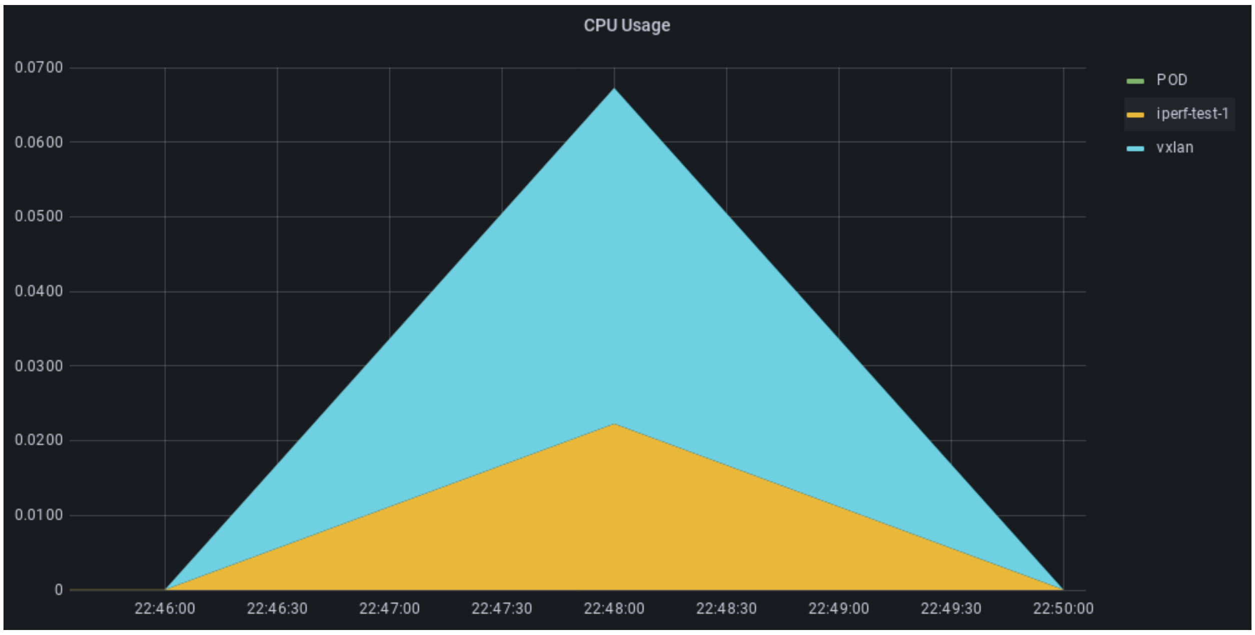


Figure 5.6: CPU usage of vxlan container during 50 Mbs

The CPU usage graph peaks at a CPU usage of `0.045`, this number is a relative number related to the number of logical cores on the compute node. Given that these compute nodes have 2 logical cores, a CPU usage of `0.045` is the same as $\frac{0.045}{2} = 2.25\%$ CPU usage. If we look at figure 5.7 it shows that the memory usage of the vxlan container stays at a steady `14MB` during the entire test and is outclassed by the standard Ubuntu container.

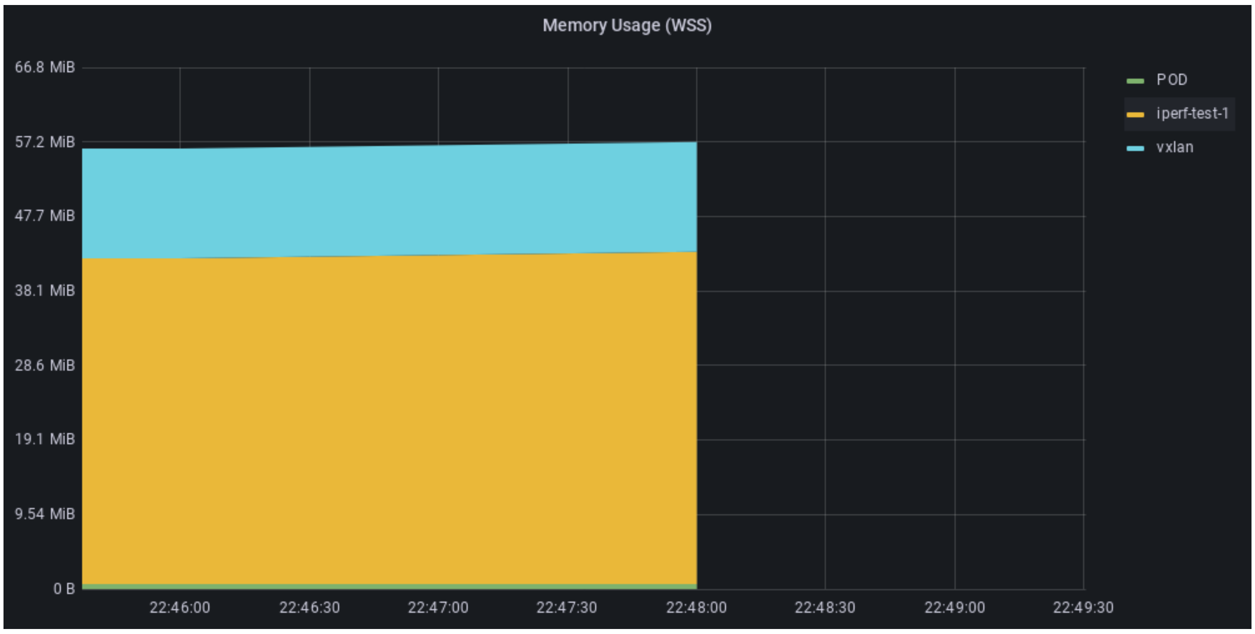


Figure 5.7: Memory usage of vxlan container during 50 Mbs

5.2.2 Data rate: 100Mb/s

Below is the summary from iperf for the `100Mb/s` speed test:

```
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  1.40 GBytes   100 Mbits/sec    0            sender
[  6]   0.00-120.04 sec  1.40 GBytes   100 Mbits/sec                 receiver
```

Listing 11: iperf results for 100mbs

This shows that `1.4GB` of data was transferred during the `100Mb/s` test. As with the `50Mb/s` we take a look at the performance metrics of the vxlan container:

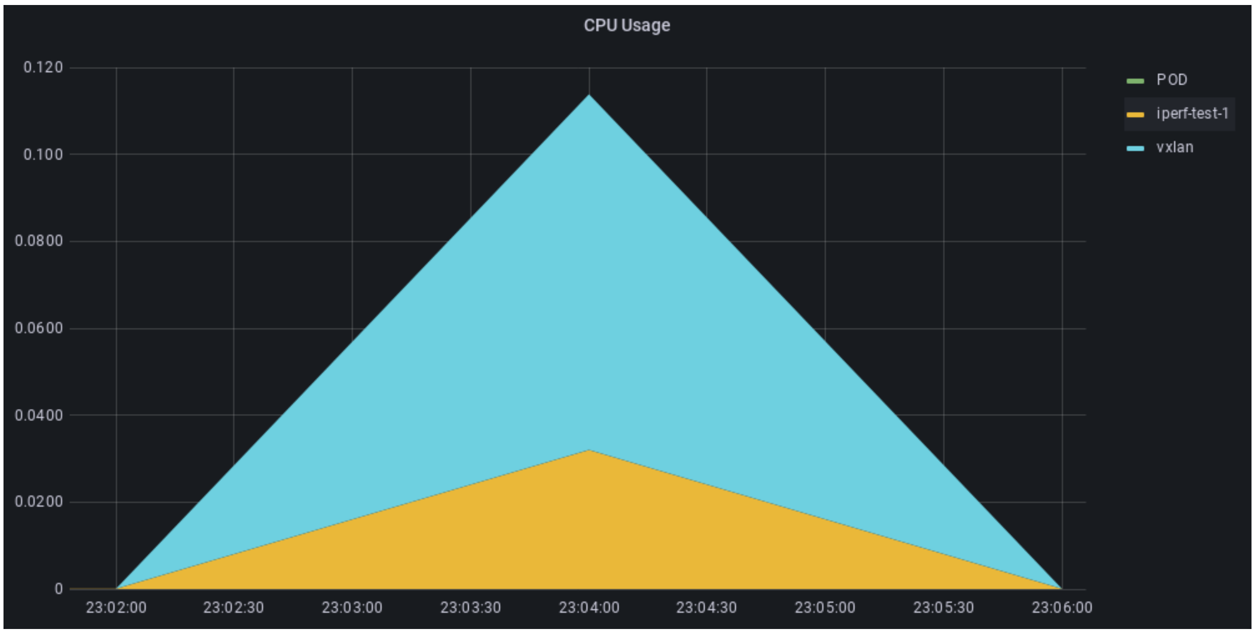


Figure 5.8: CPU usage of vxlan container during 100 Mbs

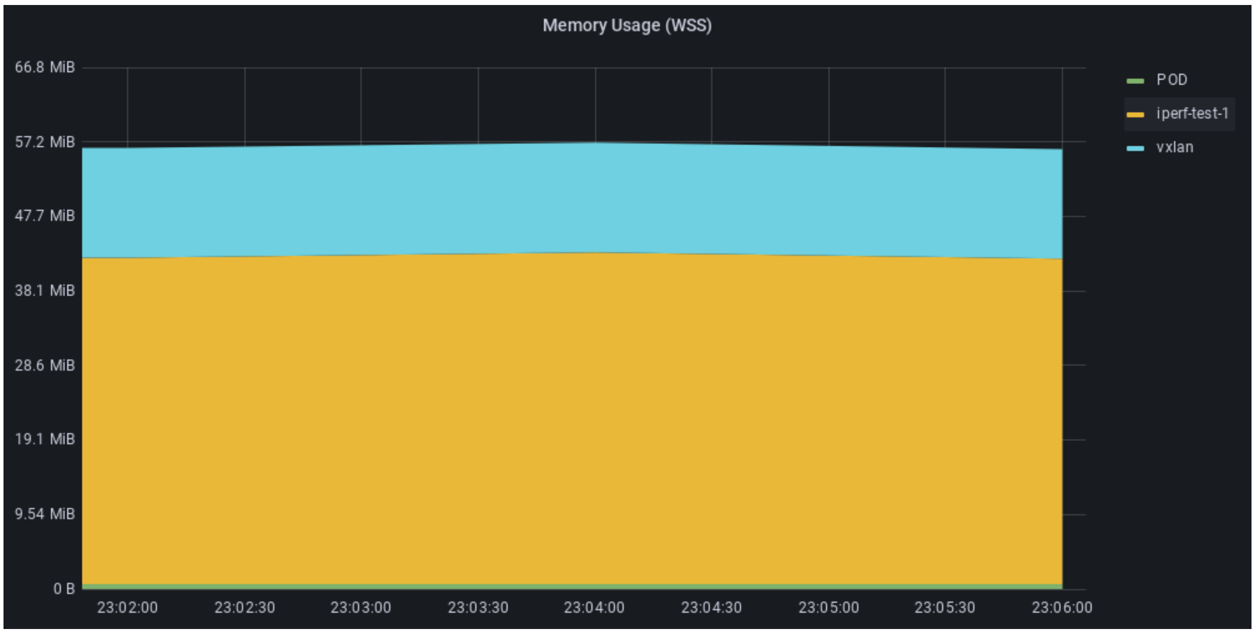


Figure 5.9: Memory usage of vxlan container during 100 Mbs

The CPU usage graph peaks at a value of `0.081` which can be translated into a percentage as such $\frac{0.081}{2} = 4.05\%$. It is approximately a doubling in CPU usage while the data rate was doubled, which makes an interesting connection. In addition, the memory usage graph shows around the same value as during the `50Mb/s` test, with the vxlan container staying at around `14MB` of memory usage during the entire test.

5.2.3 Data rate: 250Mb/s

Below is the summary from iperf for the `250Mb/s` speed test:

```
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  3.49 GBytes   250 Mbits/sec    0            sender
[  6]   0.00-120.04 sec  3.49 GBytes   250 Mbits/sec                 receiver
```

Listing 12: iperf results for 250mbs

The results show that around `3.5GB` of data was transferred during the 2 min long test duration at `250Mb/s`. Furthermore, below is the performance metrics of the vxlan container at `250Mb/s`:

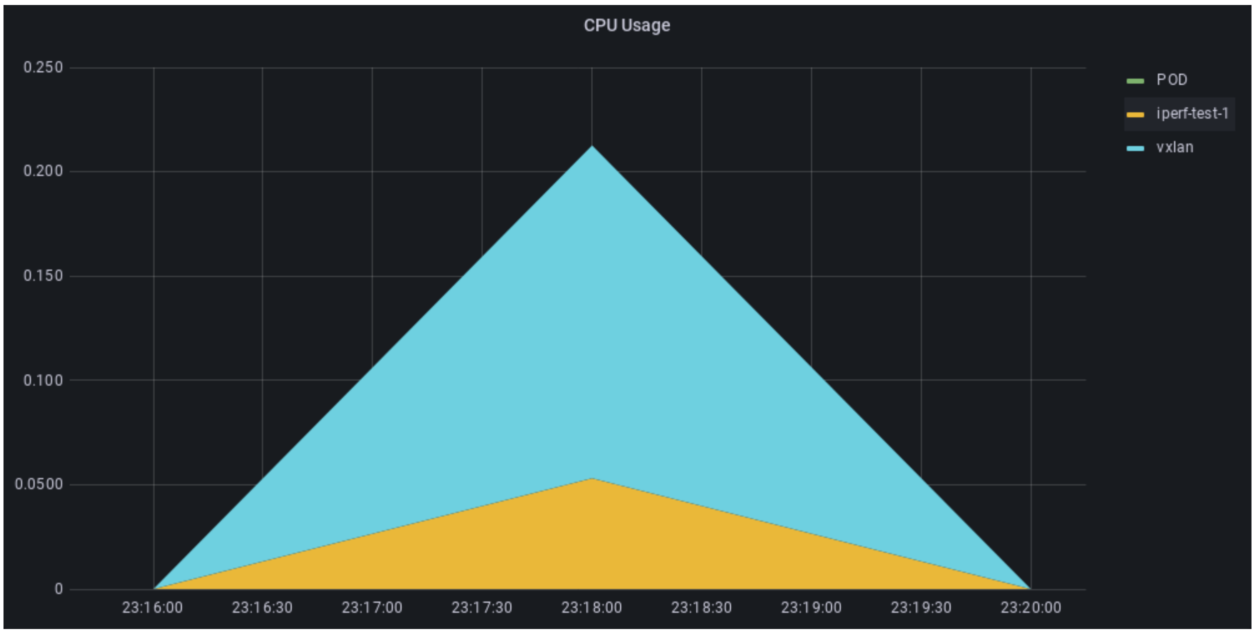


Figure 5.10: CPU usage of vxlan container during 250 Mbs

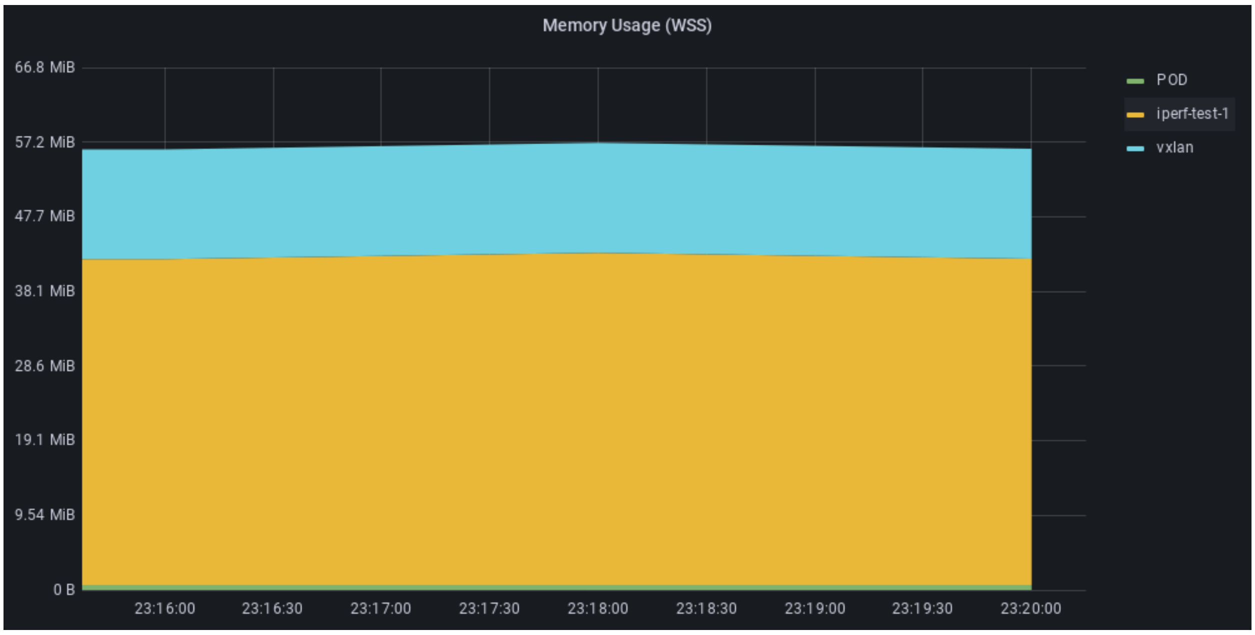


Figure 5.11: Memory usage of vxlan container during 250 Mbs

The CPU usage peaked at `0.16` which can be translated to $\frac{0.16}{2} = 8\%$ of CPU usage. Again, a doubling of CPU usage by doubling the data rate. And going the same direction as the last two data rates, the memory usage stays at a comfortable `14MB`.

5.2.4 Data rate: 500Mb/s

Last is the iperf results of the `500Mb/s` data rate test:

```
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  6.98 GBytes   500 Mbits/sec   15            sender
[  6]   0.00-120.04 sec  6.98 GBytes   500 Mbits/sec                 receiver
```

Listing 13: iperf results for 500mbs

During this test, a total of `7GB` was transferred at `500Mb/s`. The figures below show the performance metrics of the vxlan container at this data rate:

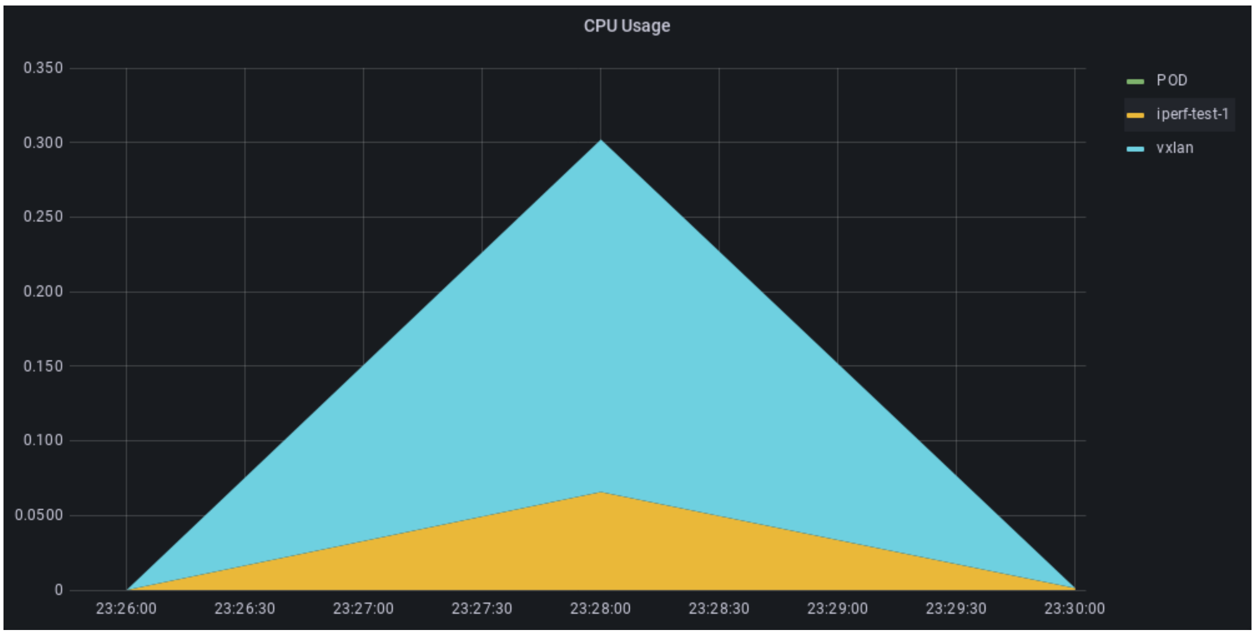


Figure 5.12: CPU usage of vxlan container during 500 Mbs

In this last speed test the CPU usage peaked at a value of `0.236` which can be viewed as $\frac{0.236}{2} = 11.8\%$ of CPU usage. This time the data rate doubled but the CPU usage only climbed by `4\%`. Which is a `50\%` increase. And following the rest of the results, the memory usage stayed at the mark of `14MB`.

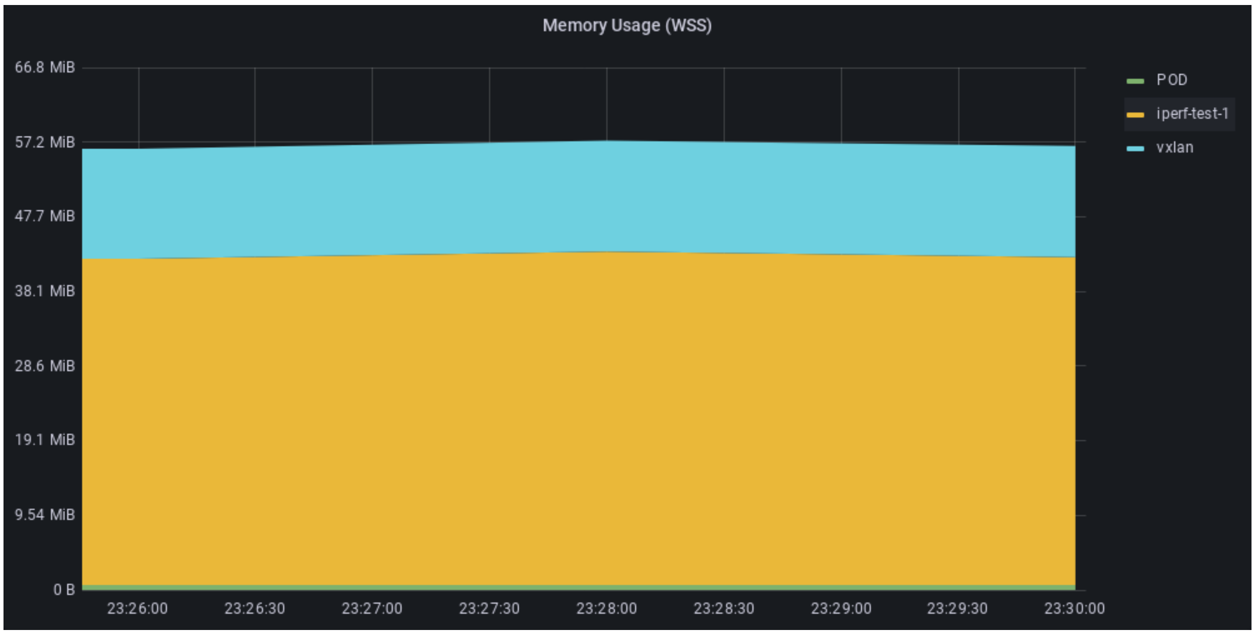


Figure 5.13: Memory usage of vxlan container during 500 Mbs

Chapter 6

Discussions

The discussion chapter will bring up the finding of the results chapter, and discuss what these results may indicate about the problem statement. As well as addressing issues with the results, and choices done in this lab environment that may have impacted the research.

6.1 Lab environment results

6.1.1 Performance

The initial user traffic test with confluence showed that the vxlan container operates at a low usage of both CPU and memory when traffic is low and does not compete with larger web applications for the resources. Though, the initial k6 test with Confluence showed that there was a need for larger traffic volume to get more sensible performance metrics of the vxlan container. The performance results with iperf showed that, even at speeds such as `500 Mb/s` the vxlan container did not consume more than $\frac{0.236}{2} = 11.8\%$ of the CPU of the compute node (2 vCPU). In addition, memory usage stayed at a comfortable 14 MB regardless of data rate. However, this CPU usage may be too high for some depending on how strict the resource limits are, as to not disturb existing applications. The conclusion that can be drawn from the performance testing alone is that the sidecar container has potential in terms of CPU and memory footprint to scale in Kubernetes.

6.1.2 Anomalies

An troubling question that arose from TSOC towards the end of the thesis was to find if there was any indications of potential packet loss in the sidecar monitoring solution, given that the vxlan traffic is sent using the UDP protocol. Which does not guarantee that packets reach their destination. If the solution shows to have too large of a packet loss, it will not be a reliable source for IDS monitoring.

As this was introduced towards the end of the project, a simple solution was tested. By comparing the captured network traffic at the UiA machine with the one captured at the sensor. One could possibly look at the number of packets transferred and see if the two hosts have observed an equal amount of packets.

Using the `50Mb/s` test as an example we have the following numbers:

- Sensor: 62 687 packets
- UiA machine: 66 217 packets

In addition, the numbers from the `100Mb/s` test:

- Sensor: 105 241 packets
- UiA machine: 121 581 packets

The numbers from the `50Mb/s` test gives an estimated packet loss of `5.3\%` while the other gives an estimated `13\%`. For an IDS solution these numbers are a on the high end and could lead to missing important malicious traffic. However, it is not for certain that this packet loss is real and to be taken as is, it is not even certain that there is packet loss, it could be counting packets in different way on the other side. Just by summarizing how tall the network stack that is operating, gives an overhead that is on the ridiculous end. There is a python application being interpreted to machine code within a docker container within a network and processor namespace created by the Kubernetes kubelet. On top of this the compute node is an EC2 instance within AWS in its own VPC that is communicating out to the UiA machine. The number of layers between the vxlan.py python code and what is captured by the sensor is staggering. And it is outside the scope of this thesis to gather enough information to find out where the networking stack could have problems. This thesis has focused on the performance metrics and impact of the sidecar container. A final anecdote to this problem is that a person with enough knowledge to know every component of this stack would be the most wanted cloud engineer in the world.

6.2 Considerations

There is a plethora of decisions that had to be taken in this research to be able to test the problem statement, without drowning in research about decisions that were far away from the main problem. And these choices were based on previous experience with the products as well as input from various sources at TSOC and co-students. Though, some of the choices could have hidden implications on the testing given the complex nature of both AWS EKS and Kubernetes itself.

6.2.1 AWS

AWS was chosen due to being the testing platform used at TSOC, making it easier to compare results, as well as receiving help in case of struggles. However, learning AWS is no short feat for someone not experienced with cloud providers. Meaning that a substantial amount of the research time has gone into learning and experimenting with AWS and its EKS and EC2 services. As well as making Terraform and Ansible files to setup the entire infrastructure in a reproducible manner, making it easier for follow up testing.

6.2.2 Kubernetes

Helper applications

In addition to learningg AWS, the research also demanded a deep dive into Kubernetes networking to understand how the sidecar container was going to achieve network capture. Though, Kubernetes is not just a single application, it is a tool for container orchestration with a plethora of plugins for additional features, such as monitoring and DevOps. And those plugins must be chosen by the cluster owner. In this research, the solution was to use *kube-prometheus-stack* given that Prometheus and Grafana are one of the most widely used monitoring solutions for Kubernetes and the stack provides a full monitoring setup with sane default configuration. In addition, Fluxcd was chosen as a gitops tool to create a streamlined and automated process of deploying the Kubernetes manifests that would become the testing environment. Lastly, Ansible was chosen due to being a familiar tool, as well as being used at TSOC.

6.3 Final thoughts

The end of the results showed that the performance impact of the sidecar container as a tapping method is not in the extremes. With a max CPU usage of `11\%` at a data rate of `500Mb/s`, the sidecar method has potential, based only on performance metrics, to scale as a network tapping method within Kubernetes. However, anomalies that appeared during testing showed that is it easier said than done to efficiently send packets through a networking stack taller than most. This has revealed a need for more in-depth research into how the traffic traverses the network stack before it ends up at the sensor. Given the discrepancy between the sensor and the UiA machine in packet numbers, among other signs.

Chapter 7

Conclusions

The first goal of the thesis was to establish a lab environment simulating a Kubernetes cluster hosted at a cloud provider. While also including configuration for easy repeatability for the setup. This has been achieved by utilizing AWS's managed Kubernetes service, Elastic Kubernetes Service. Together with a set of Terraform files which can be viewed in appendix A. The Terraform files configure a Virtual Private Cloud with a 3 node Kubernetes cluster, while also creating a standalone EC2 instance to act as the network sensor. Since the Terraform files utilize premade AWS Terraform modules, the infrastructure is created using sane defaults and removes our need to fine-tune every parameter. Next, there was a need for a complete monitoring solution for the now-created Kubernetes cluster. This was achieved through the *kube-prometheus-stack* project, which is a collection of Kubernetes manifests for a setup containing Prometheus, Grafana, Alertmanager, and Node exporter, with sane configuration defaults. To further match the goal of easy repeatability, Fluxcd was used for automatic syncing of Kubernetes manifests from a Git repository into the cluster. Which made it easy to deploy all applications to any cluster where Fluxcd was configured.

The next part was to create Ansible automation playbooks that could perform the performance tests against the sidecar container in an efficient and productive manner. By utilizing premade Ansible modules to interact with Kubernetes, playbooks were made to deploy an Ubuntu container with the iperf speed test application. While also starting packet capture at the sensor and the speed test target. Which was a machine located at the University of Agder. The playbooks finished by gathering the captured traffic and the iperf log in a local directory for further analysis. The initial performance test used k6s against a Confluence instance which showed a need for higher data rates. Where iperf came in as a perfect solution to test data rates up to `500Mb/s`.

In conclusion, there has been created a reproducible infrastructure in AWS using Terraform, together with automatic deployment of Kubernetes manifests using Fluxcd. While providing a full monitoring solution through the *kube-prometheus-stack*. Then, k6s and iperf, and Ansible were used to conduct a series of performance tests against the vxlan container, showing that CPU and memory usage stayed at sane levels and are not the bottleneck of the current solution. Furthermore, the performance testing showed that there exist anomalies in the solution that needs further investigation before this solution is to be brought into a production environment.

7.1 Future research

During the research a couple of issues arose that needs to be investigated further before the current sidecar container solution is to be trusted and reliable in production. Firstly, the issue of packet anomalies between the traffic observed at the sensor and the traffic observed by the UiA machine. As mentioned in the previous section, this will require a larger project given the complex nature of the networking stack. There should also be research looking into alternatives for vxlan.py as the sidecar container script, using more efficient languages such as Rust or C. Which may enable higher data rates or fewer anomalies. Lastly, it should be looked into whether there could be alternatives to using vxlan as the way to transport the network traffic out.

Bibliography

[1] *What is an Intrusion Detection System (IDS)?* Check Point Software. URL: https://www.checkpoint.com/cyber-hub/network-security/what-is-an-intrusion-detection-system-ids/ (visited on 04/06/2023).

[2] Checkpoint. "2022 Cloud Security Report". In: *Cybersecurity Insiders* (2022), p. 23. URL: https://pages.checkpoint.com/2022-cloud-security-report.html.

[3] *Pods*. Kubernetes. URL: https://kubernetes.io/docs/concepts/workloads/pods/ (visited on 04/06/2023).

[4] Thomas W. Edgar and David O. Manz. *Research methods for cyber security*. Cambridge, MA: Syngress, an imprint of Elsevier, 2017. 404 pp. ISBN: 978-0-12-805349-2.

[5] *What is the cloud? — Cloud definition*. Cloudflare. URL: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/ (visited on 12/10/2022).

[6] *What is the Cloud - Definition — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-the-cloud (visited on 04/07/2023).

[7] *451 Research - What's on the Mind of Cloud-Focused CTOs in 2018?* URL: https://go.451research.com/what-is-on-mind-of-cloud-focused-CTOs.html (visited on 12/10/2022).

[8] *Understanding cloud computing*. URL: https://www.redhat.com/en/topics/cloud (visited on 12/10/2022).

[9] Cloud Security Alliance. *About*. CSA. URL: https://cloudsecurityalliance.org/about/ (visited on 04/08/2023).

[10] Cloud Security Alliance. *Top Threats to Cloud Computing Pandemic Eleven*. 2022. URL: https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-pandemic-eleven/.

[11] Kubernetes. *Overview*. Kubernetes. URL: https://kubernetes.io/docs/concepts/overview/ (visited on 04/08/2023).

[12] Datadog. *9 insights on real world container use*. 9 insights on real world container use. 0. URL: https://www.datadoghq.com/container-report/ (visited on 12/10/2022).

[13] Andrew J. Younge. *Fig. 1: Kubernetes Components. The Kubernetes setup has at least three...* ResearchGate. URL: https://www.researchgate.net/figure/Kubernetes-Components-The-Kubernetes-setup-has-at-least-three-components-kublet-daemon_fig1_336889240 (visited on 04/08/2023).

[14] *A visual guide to Kubernetes networking fundamentals — Opensource.com*. URL: https://opensource.com/article/22/6/kubernetes-networking-fundamentals (visited on 12/11/2022).

[15] *Services, Load Balancing, and Networking*. Kubernetes. URL: https://kubernetes.io/docs/concepts/services-networking/ (visited on 04/08/2023).

[16] Kedar Vijay Kulkarni. *A brief overview of the Container Network Interface (CNI) in Kubernetes*. Enable Sysadmin. Publisher: Red Hat, Inc. Section: Enable Sysadmin. URL: https://www.redhat.com/sysadmin/cni-kubernetes (visited on 04/10/2023).

[17] Cloud Native Computing Foundation. *CNI*. CNI. URL: https://www.cni.dev/ (visited on 04/10/2023).

[18] *Tools for Monitoring Resources*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-usage-monitoring/ (visited on 04/10/2023).

[19] Tigera. *Prometheus Kubernetes*. Tigera. URL: https://www.tigera.io/learn/guides/prometheus-monitoring/prometheus-kubernetes/ (visited on 04/10/2023).

[20] *Resource Management for Pods and Containers*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ (visited on 04/10/2023).

[21] Ron Miller. *How AWS came to be*. TechCrunch. July 2, 2016. URL: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/ (visited on 04/10/2023).

[22] Rahul Kumar. *AWS Market Share 2023: How Far It Rules the Cloud Industry?* Section: Uncategorized. Sept. 20, 2022. URL: https://www.wpoven.com/blog/aws-market-share/ (visited on 04/10/2023).

[23] *What is AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is-aws/ (visited on 04/10/2023).

[24] *Managed Kubernetes Service – Amazon EKS – Amazon Web Services*. Amazon Web Services, Inc. URL: https://aws.amazon.com/eks/ (visited on 04/10/2023).

[25] *Logically Isolated Virtual Private Cloud—Amazon VPC – Amazon Web Services*. Amazon Web Services, Inc. URL: https://aws.amazon.com/vpc/ (visited on 04/10/2023).

[26] *AWS IAM — Identity and Access Management — Amazon Web Services*. Amazon Web Services, Inc. URL: https://aws.amazon.com/iam/ (visited on 04/10/2023).

[27] Eric Conrad, Seth Misenar, and Joshua Feldman. "Chapter 7 - Domain 7: Security operations". In: *Eleventh Hour CISSP® (Third Edition)*. Ed. by Eric Conrad, Seth Misenar, and Joshua Feldman. Syngress, Jan. 1, 2017, pp. 145–183. ISBN: 978-0-12-811248-9. DOI: 10.1016/B978-0-12-811248-9.00007-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128112489000073 (visited on 04/10/2023).

[28] *What is an Intrusion Detection System?* Palo Alto Networks. URL: https://origin-www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids (visited on 04/10/2023).

[29] Juniper. *What is VXLAN? — Juniper Networks US*. URL: https://www.juniper.net/us/en/research-topics/what-is-vxlan.html (visited on 06/01/2023).

[30] *What is Terraform — Terraform — HashiCorp Developer*. What is Terraform — Terraform — HashiCorp Developer. URL: https://developer.hashicorp.com/terraform/intro (visited on 04/10/2023).

[31] Flux. *Flux Documentation*. URL: https://fluxcd.io/flux/ (visited on 04/10/2023).

[32] Flux. *Core Concepts*. Core Concepts. Section: flux. URL: https://fluxcd.io/flux/concepts/ (visited on 05/09/2023).

[33] Red Hat. *How Ansible works*. How Ansible Works. URL: https://www.ansible.com/overview/how-ansible-works (visited on 06/01/2023).

[34] Preeti Mishra et al. "Intrusion detection techniques in cloud environment: A survey". In: *Journal of Network and Computer Applications* 77 (Jan. 1, 2017), pp. 18–47. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2016.10.015. URL: https://www.sciencedirect.com/science/article/pii/S1084804516302417 (visited on 04/23/2023).

[35] Vijit Nair. *Deeper visibility into Kubernetes environments with network monitoring*. Dec. 4, 2022. URL: https://corelight.com/blog/deeper-visibility-into-kubernetes-environments-with-network-monitoring (visited on 04/23/2023).

[36] AI Smith. *Sidecars for Network Monitoring in Kubernetes — Corelight*. Apr. 21, 2023. URL: https://corelight.com/blog/sidecars-for-network-monitoring (visited on 04/23/2023).

[37] Prometheus Community. *kube-prometheus-stack*. GitHub. URL: https://github.com/prometheus-community/helm-charts (visited on 05/10/2023).

[38] Helm. *Charts*. Charts. URL: https://helm.sh/docs/topics/charts/ (visited on 05/10/2023).

Appendix A

Terraform

This appendix includes the Terraform files used to provision the lab environment.

A.1 EKS Cluster

A.1.1 terraform.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.57.1"
    }

    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }

  }

  backend "s3" {
    bucket = "ssl-tf-states"
    key    = "new/terraform.tfstate"
    region = "eu-west-1"
  }

  required_version = "~> 1.4.0"

}
```

Listing 14: terraform.tf

A.1.2 main.tf

```
provider "kubernetes" {
  host                   = module.eks.cluster_endpoint
  cluster_ca_certificate =
↪  base64decode(module.eks.cluster_certificate_authority_data)
}

provider "aws" {
  region = var.region
}

provider "tls" {
  proxy {
    from_env = true
  }
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "dev-eks-${random_string.suffix.result}"
}

locals {
  tags = {
    Environment = "dev"
    Terraform   = "true"
    Owner = "ssl"
  }
}

resource "random_string" "suffix" {
  length  = 8
  special = false
}
```

Listing 15: main.tf

A.1.3 variables.tf

```
variable "region" {
  description = "AWS region"
  type        = string
  default     = "eu-west-1"
}

```

Listing 16: variables.tf

A.1.4 vpc.tf

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.14.2"

  name = local.cluster_name

  cidr = "10.0.0.0/16"
  azs  = slice(data.aws_availability_zones.available.names, 0, 3)

  private_subnets = ["10.0.120.0/24", "10.0.122.0/24", "10.0.123.0/24"]
  public_subnets  = ["10.0.124.0/24", "10.0.125.0/24", "10.0.126.0/24"]

  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = 1
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = 1
  }

  tags = local.tags
}
```

Listing 17: vpc.tf

A.1.5 eks-cluster.tf

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "18.30.2"

  cluster_name    = local.cluster_name
  cluster_version = "1.23"

  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets

  eks_managed_node_group_defaults = {
    ami_type = "AL2_x86_64"

    attach_cluster_primary_security_group = true

    # Disabling and using externally provided security groups
    create_security_group = false
  }

  cluster_endpoint_private_access = true
  cluster_endpoint_public_access  = true

  # Workaround for the multiple sg and the use of lb
  node_security_group_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = null
  }

  cluster_timeouts = {
    create = "60m"
  }

  ## IAM stuff fails in eks module due to apply failure

  #manage_aws_auth_configmap = true
  #create_aws_auth_configmap = true

  #aws_auth_users = [
  #  {
  #    userarn  = "arn:aws:iam::108759891166:user/fdfdf"
  #    username = "fdfdf"
  #    groups   = ["system:masters"]
  #  }
  #]

  #aws_auth_accounts = [ "108759891166" ]

  eks_managed_node_groups = {
    one = {
      name = "1-${local.cluster_name}"

      instance_types = ["t3.large"]

      min_size     = 2
      max_size     = 5
      desired_size = 3

      pre_bootstrap_user_data = <<EOF
      echo 'dev eks security tenant'
      echo -e "ssh-ed25519
↪   AAAAC3NzaC1lZDI1NTE5AAAAIAzFV4ibarFBjk6a2T18stB7Ldvl/G4jdEvGTyJkfDtu
↪   ubuntu@ip-10-0-125-177" >> /home/ec2-user/.ssh/authorized_keys
      EOF

      vpc_security_group_ids = [
        aws_security_group.node_group_one.id
      ]
    }
  }

  tags = local.tags

}
      echo 'dev eks security tenant'
      echo -e "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAzFV4ibarFBjk6a2T18stB7Ldvl/G4jdEvGTyJkfDtu ubuntu@ip-10-0-125-177" >> /home/ec2-user/.ssh/authorized_keys
      EOF

      vpc_security_group_ids = [
        aws_security_group.node_group_one.id
      ]
    }
  }

  tags = local.tags

}
```

Listing 18: eks-cluster.tf

A.1.6 ec2.tf

```
resource "aws_key_pair" "ssl" {
  key_name   = "ssl-key"
  public_key = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIG+3uD4MYXGGNaEbi7aNzUzM9MkDJ1CFamjcbSEREqKU ssl@DESKTOP-B8E1F30"
}


module "ec2_instance" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "~> 4.3"

  name = "ssl-pcaprecv"

  ami                    = "ami-015423a987dafce81"
  instance_type          = "t2.micro"
  key_name               = aws_key_pair.ssl.key_name
  monitoring             = true
  vpc_security_group_ids = [aws_security_group.ssl-pcaprecv.id, aws_security_group.vxlan.id]
  subnet_id              = module.vpc.public_subnets[1]

  associate_public_ip_address = true

  user_data = <<EOF
#!/bin/bash
echo "Copying the SSH Key Of work laptop to the server"
echo -e "ssh-ed25519
↪  AAAAC3NzaC1lZDI1NTE5AAAAIAsbx4n/ZJDQPg6jN9e4a8j7wmNFCWiWmuR3vUNFQCdZ
↪  user@master" >> /home/ubuntu/.ssh/authorized_keys
echo -e "ssh-ed25519
↪  AAAAC3NzaC1lZDI1NTE5AAAAIIc9538igE7sOQDoWmbpWQcNMv36v6WiC3/RZ9XNrm7U
↪  ssl@LAPTOP-4FN6FO9H" >> /home/ubuntu/.ssh/authorized_keys
  EOF

  tags = local.tags
}

# Allow ssh for debug
resource "aws_security_group" "ssl-pcaprecv" {
  name_prefix = "ssl-pcaprecv"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/8", "46.212.46.18/32", "212.4.46.194/32",
↪  "178.232.19.174/32"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = local.tags

}

# Allow vxlan for debug
resource "aws_security_group" "vxlan" {
  name_prefix = "ssl-pcaprecv-vxlan"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 4789
    to_port     = 4789
    protocol    = "udp"
    cidr_blocks = ["10.0.0.0/8"]
  }

  tags = local.tags
}
```

```


# resource "aws_security_group" "allow_ssh" {
#   name        = "allow_ssh"
#   description = "Allow ssh inbound traffic"

#   # using default VPC
#   vpc_id      = module.vpc.vpc_id

#   ingress {
#     description = "TLS from VPC"

#     # we should allow incoming and outoging
#     # TCP packets
#     from_port   = 22
#     to_port     = 22
#     protocol    = "tcp"

#     # allow all traffic
#     cidr_blocks = ["0.0.0.0/0"]
#   }
#   egress {
#     from_port = 0
#     to_port = 0
#     protocol = "-1"
#     cidr_blocks = ["0.0.0.0/0"]
#  }

#   tags = {
#     Name = "allow_ssh"
#   }
# }

# resource "aws_eip" "ip-test-env" {
#   instance = "${aws_instance.ssl_pcap_recv.id}"
#   vpc      = true
# }

# resource "aws_internet_gateway" "test-env-gw" {
#   vpc_id = "${module.vpc.vpc_id}"
#   tags {
#     Name = "test-env-gw"
#   }
# }

# resource "aws_route_table" "route-table-test-env" {
#   vpc_id = "${module.vpc.vpc_id}"
#   route {
#     cidr_block = "0.0.0.0/0"
#     gateway_id = "${aws_internet_gateway.test-env-gw.id}"
#   }
#   tags {
```

```
#     Name = "test-env-route-table"
#   }
# }

# resource "aws_route_table_association" "subnet-association" {
#   subnet_id      = "${module.vpc.private_subnets[2]}"
#   route_table_id = "${aws_route_table.route-table-test-env.id}"
# }

#!/bin/bash
echo "Copying the SSH Key Of work laptop to the server"
echo -e "ssh-ed25519
 ↪  AAAAC3NzaC1lZDI1NTE5AAAAIAsbx4n/ZJDQPg6jN9e4a8j7wmNFCWiWmuR3vUNFQCdZ
 ↪  user@master" >> /home/ubuntu/.ssh/authorized_keys
echo -e "ssh-ed25519
 ↪  AAAAC3NzaC1lZDI1NTE5AAAAIIc9538igE7sOQDoWmbpWQcNMv36v6WiC3/RZ9XNrm7U
 ↪  ssl@LAPTOP-4FN6FO9H" >> /home/ubuntu/.ssh/authorized_keys
  EOF

  tags = local.tags
}

# Allow ssh for debug
resource "aws_security_group" "ssl-pcaprecv" {
  name_prefix = "ssl-pcaprecv"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/8", "46.212.46.18/32", "212.4.46.194/32",
 ↪  "178.232.19.174/32"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = local.tags

}

# Allow vxlan for debug
resource "aws_security_group" "vxlan" {
  name_prefix = "ssl-pcaprecv-vxlan"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port   = 4789
    to_port     = 4789
    protocol    = "udp"
    cidr_blocks = ["10.0.0.0/8"]
  }

  tags = local.tags
}


# resource "aws_security_group" "allow_ssh" {
#   name        = "allow_ssh"
#   description = "Allow ssh inbound traffic"

#   # using default VPC
#   vpc_id      = module.vpc.vpc_id

#   ingress {
#     description = "TLS from VPC"

#     # we should allow incoming and outoging
#     # TCP packets
#     from_port   = 22
#     to_port     = 22
#     protocol    = "tcp"

#     # allow all traffic
#     cidr_blocks = ["0.0.0.0/0"]
#   }
#   egress {
#     from_port = 0
#     to_port = 0
#     protocol = "-1"
#     cidr_blocks = ["0.0.0.0/0"]
#  }

#   tags = {
#     Name = "allow_ssh"
#   }
# }

# resource "aws_eip" "ip-test-env" {
#   instance = "£{aws_instance.ssl_pcap_recv.id}"
#   vpc      = true
# }

# resource "aws_internet_gateway" "test-env-gw" {
#   vpc_id = "£{module.vpc.vpc_id}"
#   tags {
#     Name = "test-env-gw"
#   }
# }

# resource "aws_route_table" "route-table-test-env" {
#   vpc_id = "£{module.vpc.vpc_id}"
#   route {
#     cidr_block = "0.0.0.0/0"
#     gateway_id = "£{aws_internet_gateway.test-env-gw.id}"
#   }
#   tags {
#     Name = "test-env-route-table"
#   }
# }

# resource "aws_route_table_association" "subnet-association" {
#   subnet_id      = "£{module.vpc.private_subnets[2]}"
#   route_table_id = "£{aws_route_table.route-table-test-env.id}"
# }

```

Listing 19: ec2.tf

A.1.7 output.tf

```
output "cluster_id" {
  description = "EKS cluster ID"
  value       = module.eks.cluster_id
}

output "cluster_endpoint" {
  description = "Endpoint for EKS control plane"
  value       = module.eks.cluster_endpoint
}

output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control plane"
  value       = module.eks.cluster_security_group_id
}

output "region" {
  description = "AWS region"
  value       = var.region
}

output "cluster_name" {
  description = "Kubernetes Cluster Name"
  value       = local.cluster_name
}

output "aws_auth_configmap_yaml" {
  description = "Formatted yaml output for base aws-auth configmap
↪   containing roles used in cluster node groups/fargate profiles"
```

```
  value       = module.eks.aws_auth_configmap_yaml
}

output "vpc" {
  description = "VPC id"
  value       = module.vpc.vpc_id
}

output "public_ip" {
  description = "Public IP of ec2 instance"
  value       = module.ec2_instance.public_ip
}
```

Listing 20: output.tf

Appendix B

Fluxcd

This appendix includes the configuration and manifests for Fluxcd

B.1 Kustomizations

B.1.1 sync.yaml

```yaml
        # This manifest was generated by flux. DO NOT EDIT.
---
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 1m0s
  ref:
    branch: main
  secretRef:
    name: flux-system
  url: ssh://git@github.com/Siggert75/sidecar-manifests
---
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 10m0s
  path: ./aws-cluster
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
```

Listing 21: Baseline sync configuration generated by Flux bootstrap

B.1.2 monitoring.yaml

```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: confluence
  namespace: flux-system
spec:
  interval: 30m
  path: ./apps/confluence
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  timeout: 5m0s
  suspend: false
  wait: true
```

Listing 22: Kustomization for monitoring manifests

B.1.3 flux-dash.yaml

```yaml
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: fluxcd-dash
  namespace: flux-system
spec:
  interval: 30m
  path: ./apps/flux-dash
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  timeout: 5m0s
  suspend: false
  wait: true
```

Listing 23: Kustomization for Fluxcd dashboard manifests

B.1.4 nginx-controller.yaml

```yaml
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
```

```yaml
kind: Kustomization
metadata:
  name: nginx-controller
  namespace: flux-system
spec:
  interval: 30m
  path: ./apps/nginx-controller
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  timeout: 5m0s
  suspend: false
  wait: true
```

Listing 24: Kustomization for nginx ingress controller manifests

B.1.5 database.yaml

```yaml
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: postgresdb
  namespace: flux-system
spec:
  interval: 30m
  path: ./apps/database
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  timeout: 5m0s
  suspend: false
  wait: true
```

Listing 25: Kustomization for postgres database manifests

B.1.6 confluence.yaml

```yaml
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: confluence
  namespace: flux-system
spec:
  interval: 30m
  path: ./apps/confluence
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  timeout: 5m0s
  suspend: false
  wait: true
```

Listing 26: Kustomization for confluence manifests

B.2 Applications

B.2.1 Kube Prometheus Stack

kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: monitoring
resources:
  - namespace.yaml
  - repository.yaml
  - release.yaml
  - ingress.yaml
configMapGenerator:
  - name: prom-values
    files:
      - values.yaml=values.yaml
configurations:
  - kustomizeconfig.yaml
```

Listing 27: Kubernetes kustomization for kube prometheus stack

repository.yaml

```yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: prometheus-community
spec:
  interval: 120m
  type: default
  url: https://prometheus-community.github.io/helm-charts
  #type: oci
  #url: oci://ghcr.io/prometheus-community/charts
```

Listing 28: Fluxcd helm repository for kube prometheus stack

release.yaml

```yaml
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: kube-prometheus-stack
spec:
  interval: 5m
  chart:
    spec:
      version: "45.6.x"
      chart: kube-prometheus-stack
      sourceRef:
        kind: HelmRepository
        name: prometheus-community
      #verify:
      #  provider: cosign
      interval: 60m
  install:
    crds: Create
    remediation:
      retries: 2
  upgrade:
    crds: CreateReplace
  valuesFrom:
    - kind: ConfigMap
      name: prom-values
```

Listing 29: Fluxcd helm release for kube prometheus stack

values.yaml

```yaml
fullnameOverride: prometheus

defaultRules:
  create: true
  rules:
    alertmanager: true
    etcd: true
    configReloaders: true
    general: true
    k8s: true
    kubeApiserverAvailability: true
    kubeApiserverBurnrate: true
    kubeApiserverHistogram: true
    kubeApiserverSlos: true
    kubelet: true
    kubeProxy: true
    kubePrometheusGeneral: true
    kubePrometheusNodeRecording: true
    kubernetesApps: true
    kubernetesResources: true
    kubernetesStorage: true
    kubernetesSystem: true
    kubeScheduler: true
    kubeStateMetrics: true
    network: true
    node: true
    nodeExporterAlerting: true
    nodeExporterRecording: true
    prometheus: true
    prometheusOperator: true

alertmanager:
  fullnameOverride: alertmanager
  enabled: true
  ingress:
    enabled: false

grafana:
  enabled: true
  fullnameOverride: grafana
  forceDeployDatasources: false
  forceDeployDashboards: false
  defaultDashboardsEnabled: true
  defaultDashboardsTimezone: utc+1
  imageRenderer:
    enabled: true
  serviceMonitor:
    enabled: true
  admin:
    existingSecret: grafana-admin-credentials
    userKey: admin-user
    passwordKey: admin-password

kubeApiServer:
  enabled: true

kubelet:
  enabled: true
  serviceMonitor:
    metricRelabelings:
      - action: replace
        sourceLabels:
          - node
        targetLabel: instance

```

```yaml
kubeControllerManager:
  enabled: true
  endpoints: # ips of servers
    - 10.0.120.222
    - 10.0.122.69
    - 10.0.123.247

coreDns:
  enabled: true

kubeDns:
  enabled: false

kubeEtcd:
  enabled: true
  endpoints: # ips of servers
    - 10.0.120.222
    - 10.0.122.69
    - 10.0.123.247
  service:
    enabled: true
    port: 2381
    targetPort: 2381

kubeScheduler:
  enabled: true
  endpoints: # ips of servers
    - 10.0.120.222
    - 10.0.122.69
    - 10.0.123.247

kubeProxy:
  enabled: true
  endpoints: # ips of servers
    - 10.0.120.222
    - 10.0.122.69
    - 10.0.123.247

kubeStateMetrics:
  enabled: true

kube-state-metrics:
  fullnameOverride: kube-state-metrics
  selfMonitor:
    enabled: true
  prometheus:
    monitor:
      enabled: true
      relabelings:
        - action: replace
          regex: (.*)
          replacement: $1
          sourceLabels:
            - __meta_kubernetes_pod_node_name
          targetLabel: kubernetes_node

nodeExporter:
  enabled: true
  serviceMonitor:
    relabelings:
      - action: replace
        regex: (.*)
        replacement: $1
        sourceLabels:
          - __meta_kubernetes_pod_node_name
        targetLabel: kubernetes_node

prometheus-node-exporter:
  fullnameOverride: node-exporter
  podLabels:
    jobLabel: node-exporter
  extraArgs:
    - --collector.filesystem.mount-points-exclude=^/(dev|proc|sys|
    var/lib/docker/.+|var/lib/kubelet/.+)($|/)
    - --collector.filesystem.fs-types-exclude=^(autofs|binfmt_misc|bpf
↪   |cgroup2|configfs|debugfs|devpts|devtmpfs|fusectl|hugetlbfs|iso9660
↪   |mqueue|nsfs|overlay|proc|procfs|pstore|rpc_pipefs|securityfs|
↪   selinuxfs|squashfs|sysfs|tracefs)$
  service:
    portName: http-metrics
  prometheus:
    monitor:
      enabled: true
      relabelings:
        - action: replace
          regex: (.*)
          replacement: $1
          sourceLabels:
            - __meta_kubernetes_pod_node_name
          targetLabel: kubernetes_node
  resources:
    requests:
      memory: 512Mi
      cpu: 250m
    limits:
      memory: 2048Mi

prometheusOperator:
  enabled: true
  prometheusConfigReloader:
    resources:
      requests:
        cpu: 200m
        memory: 50Mi
      limits:
        memory: 100Mi

prometheus:
  enabled: true
  prometheusSpec:
    replicas: 1
    replicaExternalLabelName: "replica"
    ruleSelectorNilUsesHelmValues: false
    serviceMonitorSelectorNilUsesHelmValues: false
    podMonitorSelectorNilUsesHelmValues: false
    probeSelectorNilUsesHelmValues: false
    retention: 7d
    scrapeInterval: 10s
    enableAdminAPI: true
    walCompression: true

thanosRuler:
  enabled: false
```

Listing 30: Helm values for kube prometheus stack

B.2.2 Flux dashboard

HelmRepository and HelmRelease

```yaml
---
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  annotations:
    metadata.weave.works/description: This is the source location for the
 ↪  Weave GitOps
      Dashboard's helm chart.
  labels:
    app.kubernetes.io/component: ui
    app.kubernetes.io/created-by: weave-gitops-cli
    app.kubernetes.io/name: weave-gitops-dashboard
    app.kubernetes.io/part-of: weave-gitops
  name: ww-gitops
  namespace: flux-system
spec:
  interval: 1h0m0s
  type: oci
  url: oci://ghcr.io/weaveworks/charts
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  annotations:
    metadata.weave.works/description: This is the Weave GitOps Dashboard.
↪  It provides
      a simple way to get insights into your GitOps workloads.
  name: ww-gitops
  namespace: flux-system
spec:
  chart:
    spec:
      chart: weave-gitops
      sourceRef:
        kind: HelmRepository
        name: ww-gitops
  interval: 1h0m0s
  values:
    adminUser:
      create: true
      passwordHash:
↪  $2a$10$3U4Liyd4QOWIs6ev7q6C.OO5EOcnsjEsCJCZ6jUzMuKXyhl7FYQ1y
      username: admin


```

Listing 31: Fluxcd helm release and helm repository for the Fluxcd dashboard

B.2.3 Nginx ingress controller

deployment.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  minReadySeconds: 0
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/name: ingress-nginx
```

```yaml
  template:
    metadata:
      labels:
        app.kubernetes.io/component: controller
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/name: ingress-nginx
    spec:
      containers:
      - args:
        - /nginx-ingress-controller
        - --publish-service=$(POD_NAMESPACE)/ingress-nginx-controller
        - --election-id=ingress-nginx-leader
        - --controller-class=k8s.io/ingress-nginx
        - --ingress-class=nginx
        - --configmap=$(POD_NAMESPACE)/ingress-nginx-controller
        - --validating-webhook=:8443
        - --validating-webhook-certificate=/usr/local/certificates/cert
        - --validating-webhook-key=/usr/local/certificates/key
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: LD_PRELOAD
          value: /usr/local/lib/libmimalloc.so
        image: registry.k8s.io/ingress-nginx/controller:v1.6.4@sha256:
15be4666c53052484dd2992efacf2f50ea77a78ae8aa21ccd91af6baaa7ea22f
        imagePullPolicy: IfNotPresent
        lifecycle:
          preStop:
            exec:
              command:
              - /wait-shutdown
        livenessProbe:
          failureThreshold: 5
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        name: controller
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
        - containerPort: 443
          name: https
          protocol: TCP
        - containerPort: 8443
          name: webhook
          protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        resources:
          requests:
            cpu: 100m
            memory: 90Mi
        securityContext:
          allowPrivilegeEscalation: true
          capabilities:
            add:
            - NET_BIND_SERVICE
            drop:
            - ALL
          runAsUser: 101
        volumeMounts:
        - mountPath: /usr/local/certificates/
          name: webhook-cert
          readOnly: true
      dnsPolicy: ClusterFirst
      nodeSelector:
        kubernetes.io/os: linux
      serviceAccountName: ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
      - name: webhook-cert
        secret:
          secretName: ingress-nginx-admission
```

Listing 32: Deployment manifest for Nginx ingress controller

Supporting manifests for Nginx ingress controller

```
---
apiVersion: v1
data:
  allow-snippet-annotations: "true"
```

```yaml
kind: ConfigMap
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-controller
  namespace: ingress-nginx
---
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: nginx
spec:
  controller: k8s.io/ingress-nginx
---
apiVersion: batch/v1
kind: Job
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission-create
  namespace: ingress-nginx
spec:
  template:
    metadata:
      labels:
        app.kubernetes.io/component: admission-webhook
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/part-of: ingress-nginx
        app.kubernetes.io/version: 1.6.4
      name: ingress-nginx-admission-create
    spec:
      containers:
      - args:
        - create
        - --host=ingress-nginx-controller-admission,ingress-nginx-
controller-admission.$(POD_NAMESPACE).svc
        - --namespace=$(POD_NAMESPACE)
        - --secret-name=ingress-nginx-admission
        env:
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        image: registry.k8s.io/ingress-nginx/kube-webhook-certgen:
v20220916-gd32f8c343@sha256:
39c5b2e3310dc4264d638ad28d9d1d96c4cbb2b2dcfb52368fe4e3c63f61e10f
        imagePullPolicy: IfNotPresent
        name: create
        securityContext:
          allowPrivilegeEscalation: false
      nodeSelector:
        kubernetes.io/os: linux
      restartPolicy: OnFailure
      securityContext:
        fsGroup: 2000
        runAsNonRoot: true
        runAsUser: 2000
      serviceAccountName: ingress-nginx-admission
---
apiVersion: batch/v1
kind: Job
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission-patch
  namespace: ingress-nginx
spec:
  template:
    metadata:
      labels:
        app.kubernetes.io/component: admission-webhook
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/part-of: ingress-nginx
        app.kubernetes.io/version: 1.6.4
      name: ingress-nginx-admission-patch
    spec:
      containers:
      - args:
        - patch
        - --webhook-name=ingress-nginx-admission
        - --namespace=$(POD_NAMESPACE)
        - --patch-mutating=false
        - --secret-name=ingress-nginx-admission
        - --patch-failure-policy=Fail
        env:
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        image: registry.k8s.io/ingress-nginx/kube-webhook-certgen:
v20220916-gd32f8c343@sha256:
39c5b2e3310dc4264d638ad28d9d1d96c4cbb2b2dcfb52368fe4e3c63f61e10f
        imagePullPolicy: IfNotPresent
        name: patch
        securityContext:
          allowPrivilegeEscalation: false
      nodeSelector:
        kubernetes.io/os: linux
      restartPolicy: OnFailure
      securityContext:
        fsGroup: 2000
        runAsNonRoot: true
        runAsUser: 2000
      serviceAccountName: ingress-nginx-admission
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx
  namespace: ingress-nginx
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - configmaps
  - pods
  - secrets
  - endpoints
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses/status
  verbs:
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - ingressclasses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - coordination.k8s.io
  resourceNames:
  - ingress-nginx-leader
  resources:
  - leases
  verbs:
  - get
  - update
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
  - get
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
  namespace: ingress-nginx
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - nodes
  - pods
  - secrets
  - namespaces
  verbs:
  - list
  - watch
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses/status
  verbs:
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - ingressclasses
  verbs:
```

```yaml
  - get
  - list
  - watch
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
  - get
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
rules:
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - validatingwebhookconfigurations
  verbs:
  - get
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx
subjects:
- kind: ServiceAccount
  name: ingress-nginx
  namespace: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
```

```yaml
kind: RoleBinding
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx-admission
subjects:
- kind: ServiceAccount
  name: ingress-nginx-admission
  namespace: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx
subjects:
- kind: ServiceAccount
  name: ingress-nginx
  namespace: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx-admission
subjects:
```

```yaml
- kind: ServiceAccount
  name: ingress-nginx-admission
  namespace: ingress-nginx
---
apiVersion: v1
automountServiceAccountToken: true
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx
  namespace: ingress-nginx
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
  namespace: ingress-nginx
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    # service.beta.kubernetes.io/aws-load-balancer-type: "external"
    # service.beta.kubernetes.io/aws-load-balancer-nlb-target-type:
↪   "instance"
    # service.beta.kubernetes.io/aws-load-balancer-scheme:
↪   "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp

↪   service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled:
↪   "true"
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
    service.beta.kubernetes.io/aws-load-balancer-name:
↪   "sidecartap-nginx-lb"
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  externalTrafficPolicy: Local
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - appProtocol: http
    name: http
    port: 80
    protocol: TCP
    targetPort: http
  - appProtocol: https
    name: https
    port: 443
    protocol: TCP
    targetPort: https
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-controller-admission
  namespace: ingress-nginx
spec:
  ports:
  - appProtocol: https
    name: https-webhook
    port: 443
    targetPort: webhook
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  type: ClusterIP
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  labels:
    app.kubernetes.io/component: admission-webhook
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.6.4
  name: ingress-nginx-admission
webhooks:
- admissionReviewVersions:
  - v1
  clientConfig:
    service:
      name: ingress-nginx-controller-admission
      namespace: ingress-nginx
      path: /networking/v1/ingresses
  failurePolicy: Fail
  matchPolicy: Equivalent
  name: validate.nginx.ingress.kubernetes.io
  rules:
  - apiGroups:
    - networking.k8s.io
    apiVersions:
    - v1
    operations:
    - CREATE
    - UPDATE
    resources:
    - ingresses
  sideEffects: None
```

Listing 33: Supporting manifests for Nginx ingress controller

Appendix C

The sidecar

C.1 Sidecar container

C.1.1 vxlan.py

```python
#!/usr/bin/env python3

import fcntl
import os
import re
import requests
import socket
import struct
import sys
import threading
import time

class Sensor():
    def __init__(self):
        self.sensor = os.environ.get('SENSOR')
        if not self.sensor:
            raise ValueError('$SENSOR is not set')

        self.ipMatch = '.'.join(['(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)']*4)
        self.isAddr = re.match('\A%s\Z' % self.ipMatch, self.sensor)

        self.lastUpdated = None
        self.addr = None

        self.mutex = threading.Lock()

    def ip(self):
        if self.isAddr:
            return self.sensor

        if self.lastUpdated and time.time() - self.lastUpdated < 60 and self.addr:
            return self.addr

        with self.mutex:
            self.addr = None

            serviceAccountDirectory = '/var/run/secrets/kubernetes.io/serviceaccount'
            with open(os.path.join(serviceAccountDirectory, 'token'), 'r') as fd:
                k8stoken = fd.read().strip()

            # Get metadata about all containers
            metadata = requests.get('https://kubernetes.default.svc.
            cluster.local/api/v1/pods', verify=os.path.join(serviceAccountDirectory, 'ca.crt'), headers={'Authorization': 'Bearer %s' % k8stoken}).json()

            for pod in metadata['items']:
                if pod['metadata']['labels'].get('run') == self.sensor:
                    print(pod['status'])
                    self.addr = pod['status'].get('podIP')

            if not re.match('\A%s\Z' % self.ipMatch, self.addr):
                # Possible temporary failure
                print('pod %s IP address not found' % self.sensor)
            else:
                self.lastUpdated = time.time()

            print(self.addr)
            return self.addr

def main():
    interface = os.environ.get('INTERFACE')
    if not interface:
        print('$INTERFACE is not set')
        return 1

    try:
        sensor = Sensor()
    except ValueError as e:
        print(str(e))
        return 1

    vni = os.environ.get('VNI')
    if not vni:
        print('$VNI is not set')
        return 1

    try:
        vni = int(vni, 16)
    except ValueError:
        print('$VNI is not parsable as hexadecimal')
        return 1

    if vni & 0xff000000:
        print('$VNI should be no greater than 0xffffff')
        return 1

    ipAddr = fcntl.ioctl(socket.socket(socket.AF_INET, socket.SOCK_DGRAM),
    ↪  0x8915, struct.pack('256s', interface.encode('ascii')))[20:24]  #
    ↪  SIOCGIFADDR

    sniff = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
    ↪  socket.ntohs(3)) # ETH_P_ALL
    sniff.bind((interface, 0))

    vxlan = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # VXLAN header
    vxlanHeader  = struct.pack('!L', 0x08000000)
    vxlanHeader += struct.pack('!L', vni << 8)

    # Packet number setup
    def getconn():
        connected = False
        while not connected:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                s.connect((os.getenv("PACKET_HOST", "127.0.0.1"), 4444))
                connected = True
                return s
            except socket.error:
                print("Lost connection")
                time.sleep(2)
    s = getconn()
    hostname = socket.gethostname()
    ## getting the IP address using socket.gethostbyname() method
    ip_address = socket.gethostbyname(hostname)
    s.sendall(f"IP:{ip_address}".encode())
    s.recv(1024)
    s.sendall(b"Sending packet numbers")
    s.recv(1024)

    while True:
        (data, _) = sniff.recvfrom(65535)

        sensorAddr = sensor.ip()
        if not sensorAddr:
            continue

        ipPacket = data[14:]
        ipHeaderLength = (struct.unpack('B', bytes([data[0]]))[0] & 0x0f) *
        ↪  4
```

```python

        srcIP = ipPacket[12:16]
        dstIP = ipPacket[16:20]
        protocol = ipPacket[9]

        if protocol in [6, 17] and len(ipPacket) >= ipHeaderLength+3:
            srcPort = struct.unpack('!H', ipPacket[ipHeaderLength+0:ipHeaderLength+2])[0]
            dstPort = struct.unpack('!H', ipPacket[ipHeaderLength+2:ipHeaderLength+4])[0]

            # Not our own traffic on UDP/4789 to sensor
            if protocol == 17 and dstPort == 4789 and dstIP == sensorAddr:
                continue

            # Not traffic to redis
            if socket.inet_ntoa(dstIP) == "172.20.205.187":
                print(f"Dropping dstIP => {socket.inet_ntoa(dstIP)}")
                continue

            if socket.inet_ntoa(srcIP) == "172.20.205.187":
                print(f"Dropping SRC => {socket.inet_ntoa(srcIP)}")
                continue

            if protocol == 17:
                data = ipPacket[ipHeaderLength+8:]
                if data[:len(vxlanHeader)] == vxlanHeader:
                    continue

            if protocol == 17:
                continue

            print('Got %s byte%s from %s:%s to %s:%s proto %s (VXLAN %s->%s)' % (len(data), len(data) != 1 and 's' or '', socket.inet_ntoa(srcIP), srcPort, socket.inet_ntoa(dstIP), dstPort, protocol, socket.inet_ntoa(ipAddr), sensorAddr))

        vxlan.sendto(vxlanHeader+data, (sensorAddr, 4789))

        s.sendall(b"1")
        s.recv(1024)

if __name__ == '__main__':
    sys.exit(main())
```

Listing 34: Python script running network capture

C.1.2 Dockerfile

```
FROM python:3.9.12-slim

RUN pip install requests

COPY vxlan.py /vxlan.py
RUN chmod 755 vxlan.py

CMD ["/vxlan.py"]
```

Listing 35: Dockerfile for vxlan.py

Appendix D

k6s

D.1 k6s test file

D.1.1 test.js

```
import http from 'k6/http';
import { check, group, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '5m', target: 20 }, // simulate ramp-up of traffic from 1
    ↪  to 100 users over 5 minutes.
    { duration: '10m', target: 20 }, // stay at 100 users for 10 minutes
    { duration: '5m', target: 0 }, // ramp-down to 0 users
  ],
  thresholds: {
    'http_req_duration': ['p(99)<3000'], // 99% of requests must complete
    ↪  below 1.5s
  },
};

const BASE_URL = 'https://confluence.amonguslab.net';


export function setup() {
  const login_url = `${BASE_URL}/dologin.action`;

  const payload = JSON.stringify({
    os_username: 'admin',
    os_password: 'oTG82D#ZKp7P42EvtLsusYhz%',
    login: "Log+in",
    os_destination: "/index.action"
  });

  const params = {
    headers: {
      'Content-Type': 'application/json'
    }
  };


  const res = http.post(login_url, payload, params);


  check(res, {
    'has cookie jsessionid': (r) => r.cookies.JSESSIONID.length > 0,
  });

  const jar = http.cookieJar();

  jar.set(BASE_URL, 'JSESSIONID', res.cookies.JSESSIONID.value, {
    domain: BASE_URL,
    path: '/',
    secure: true,
  });



  return jar;

}

export default (jar) => {
  const url = `${BASE_URL}/display/MAS/Master`;

  const res = http.get(url, { jar });

  check(res, {
    'has status 200': (r) => r.status === 200
  })



  // let checkRes = check(res, {
  //   "Homepage body size is 612 bytes": (r) => r.body.length === 612,
  //   "Homepage welcome header present": (r) => r.body.indexOf("Welcome
  ↪  to nginx!") !== -1
  // });

  sleep(1);
};


```

Listing 36: Javascript code for k6s test.js

Appendix E

iperf

E.1 Data rate results

E.1.1 50 Mb/s

```
Connecting to host 128.39.145.94, port 5201
[  6] local 10.0.123.138 port 46822 connected to 128.39.145.94 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  6]   0.00-1.00   sec  6.06 MBytes  50.8 Mbits/sec    0   1.73 MBytes
[  6]   1.00-2.00   sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]   2.00-3.00   sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]   3.00-4.00   sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]   4.00-5.00   sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]   5.00-6.00   sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]   6.00-7.00   sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]   7.00-8.00   sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]   8.00-9.00   sec  6.00 MBytes  50.4 Mbits/sec    0   1.73 MBytes
[  6]   9.00-10.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  10.00-11.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  11.00-12.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  12.00-13.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  13.00-14.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  14.00-15.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  15.00-16.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  16.00-17.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  17.00-18.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  18.00-19.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  19.00-20.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  20.00-21.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  21.00-22.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  22.00-23.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  23.00-24.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  24.00-25.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  25.00-26.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  26.00-27.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  27.00-28.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  28.00-29.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  29.00-30.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  30.00-31.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  31.00-32.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  32.00-33.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  33.00-34.00  sec  5.88 MBytes  49.2 Mbits/sec    0   1.73 MBytes
[  6]  34.00-35.00  sec  6.00 MBytes  50.4 Mbits/sec    0   1.73 MBytes
[  6]  35.00-36.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  36.00-37.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  37.00-38.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  38.00-39.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  39.00-40.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  40.00-41.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  41.00-42.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  42.00-43.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  43.00-44.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  44.00-45.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  45.00-46.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  46.00-47.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  47.00-48.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  48.00-49.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  49.00-50.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  50.00-51.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  51.00-52.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  52.00-53.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  53.00-54.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  54.00-55.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  55.00-56.00  sec  5.88 MBytes  49.3 Mbits/sec   28   1.73 MBytes
[  6]  56.00-57.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  57.00-58.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  58.00-59.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  59.00-60.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  60.00-61.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  61.00-62.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  62.00-63.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  63.00-64.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  64.00-65.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  65.00-66.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  66.00-67.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  67.00-68.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  68.00-69.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  69.00-70.00  sec  6.00 MBytes  50.2 Mbits/sec    0   1.73 MBytes
[  6]  70.00-71.00  sec  6.00 MBytes  50.5 Mbits/sec    0   1.73 MBytes
[  6]  71.00-72.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  72.00-73.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  73.00-74.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  74.00-75.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  75.00-76.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  76.00-77.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  77.00-78.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  78.00-79.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  79.00-80.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  80.00-81.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  81.00-82.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  82.00-83.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  83.00-84.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  84.00-85.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  85.00-86.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  86.00-87.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  87.00-88.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  88.00-89.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  89.00-90.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  90.00-91.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  91.00-92.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  92.00-93.00  sec  6.00 MBytes  50.2 Mbits/sec    0   1.73 MBytes
[  6]  93.00-94.00  sec  5.88 MBytes  49.4 Mbits/sec    0   1.73 MBytes
[  6]  94.00-95.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  95.00-96.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  96.00-97.00  sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6]  97.00-98.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  98.00-99.00  sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6]  99.00-100.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 100.00-101.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 101.00-102.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 102.00-103.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 103.00-104.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 104.00-105.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 105.00-106.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 106.00-107.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 107.00-108.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 108.00-109.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 109.00-110.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 110.00-111.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 111.00-112.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 112.00-113.00 sec  5.88 MBytes  49.1 Mbits/sec    0   1.73 MBytes
[  6] 113.00-114.00 sec  6.00 MBytes  50.5 Mbits/sec    0   1.73 MBytes
[  6] 114.00-115.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 115.00-116.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 116.00-117.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 117.00-118.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
[  6] 118.00-119.00 sec  5.88 MBytes  49.3 Mbits/sec    0   1.73 MBytes
[  6] 119.00-120.00 sec  6.00 MBytes  50.3 Mbits/sec    0   1.73 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec   715 MBytes  50.0 Mbits/sec   28
↪  sender
[  6]   0.00-120.04 sec   715 MBytes  50.0 Mbits/sec
↪  receiver

iperf Done.
```

Listing 37: iperf test result 50 Mb/s

E.1.2 100 Mb/s

```
Connecting to host 128.39.145.94, port 5201
[  6] local 10.0.123.138 port 41706 connected to 128.39.145.94 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  6]   0.00-1.00   sec  11.9 MBytes   100 Mbits/sec    0   1.86 MBytes
[  6]   1.00-2.00   sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]   2.00-3.00   sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]   3.00-4.00   sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]   4.00-5.00   sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]   5.00-6.00   sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]   6.00-7.00   sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]   7.00-8.00   sec  11.9 MBytes  99.7 Mbits/sec    0   1.86 MBytes
[  6]   8.00-9.00   sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]   9.00-10.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  10.00-11.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  11.00-12.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  12.00-13.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  13.00-14.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  14.00-15.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  15.00-16.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  16.00-17.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  17.00-18.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  18.00-19.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  19.00-20.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  20.00-21.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  21.00-22.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  22.00-23.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  23.00-24.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  24.00-25.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  25.00-26.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  26.00-27.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  27.00-28.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  28.00-29.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  29.00-30.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  30.00-31.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  31.00-32.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  32.00-33.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  33.00-34.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  34.00-35.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  35.00-36.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  36.00-37.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  37.00-38.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  38.00-39.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  39.00-40.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  40.00-41.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  41.00-42.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  42.00-43.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  43.00-44.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  44.00-45.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  45.00-46.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  46.00-47.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  47.00-48.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  48.00-49.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  49.00-50.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  50.00-51.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  51.00-52.00  sec  11.9 MBytes  99.7 Mbits/sec    0   1.86 MBytes
[  6]  52.00-53.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  53.00-54.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  54.00-55.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  55.00-56.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  56.00-57.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  57.00-58.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  58.00-59.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  59.00-60.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  60.00-61.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  61.00-62.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  62.00-63.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  63.00-64.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  64.00-65.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  65.00-66.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  66.00-67.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  67.00-68.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  68.00-69.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  69.00-70.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  70.00-71.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  71.00-72.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  72.00-73.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  73.00-74.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  74.00-75.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  75.00-76.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  76.00-77.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  77.00-78.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  78.00-79.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  79.00-80.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  80.00-81.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  81.00-82.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  82.00-83.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  83.00-84.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  84.00-85.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  85.00-86.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  86.00-87.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  87.00-88.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  88.00-89.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  89.00-90.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  90.00-91.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  91.00-92.00  sec  12.0 MBytes   100 Mbits/sec    0   1.86 MBytes
[  6]  92.00-93.00  sec  11.9 MBytes  99.8 Mbits/sec    0   1.86 MBytes
[  6]  93.00-94.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  94.00-95.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  95.00-96.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  96.00-97.00  sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6]  97.00-98.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  98.00-99.00  sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6]  99.00-100.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 100.00-101.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 101.00-102.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 102.00-103.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 103.00-104.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 104.00-105.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 105.00-106.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 106.00-107.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 107.00-108.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 108.00-109.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 109.00-110.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 110.00-111.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 111.00-112.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 112.00-113.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 113.00-114.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 114.00-115.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 115.00-116.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 116.00-117.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 117.00-118.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
[  6] 118.00-119.00 sec  12.0 MBytes   101 Mbits/sec    0   1.86 MBytes
[  6] 119.00-120.00 sec  11.9 MBytes  99.6 Mbits/sec    0   1.86 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  1.40 GBytes   100 Mbits/sec    0
↪  sender
[  6]   0.00-120.04 sec  1.40 GBytes   100 Mbits/sec
↪  receiver

iperf Done.
```

Listing 38: iperf test result 100 Mb/s

E.1.3 250 Mb/s

```
Connecting to host 128.39.145.94, port 5201
[  6] local 10.0.123.138 port 49576 connected to 128.39.145.94 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  6]   0.00-1.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   1.00-2.00   sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]   2.00-3.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   3.00-4.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   4.00-5.00   sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]   5.00-6.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   6.00-7.00   sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]   7.00-8.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   8.00-9.00   sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]   9.00-10.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  10.00-11.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  11.00-12.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  12.00-13.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  13.00-14.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  14.00-15.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  15.00-16.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  16.00-17.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  17.00-18.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  18.00-19.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  19.00-20.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  20.00-21.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  21.00-22.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  22.00-23.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  23.00-24.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  24.00-25.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  25.00-26.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  26.00-27.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  27.00-28.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  28.00-29.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  29.00-30.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  30.00-31.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  31.00-32.00  sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6]  32.00-33.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  33.00-34.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  34.00-35.00  sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6]  35.00-36.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  36.00-37.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  37.00-38.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  38.00-39.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  39.00-40.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  40.00-41.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  41.00-42.00  sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6]  42.00-43.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  43.00-44.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  44.00-45.00  sec  29.6 MBytes   248 Mbits/sec    0   2.68 MBytes
[  6]  45.00-46.00  sec  30.0 MBytes   252 Mbits/sec    0   2.68 MBytes
[  6]  46.00-47.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  47.00-48.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  48.00-49.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  49.00-50.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  50.00-51.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  51.00-52.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  52.00-53.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  53.00-54.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  54.00-55.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  55.00-56.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  56.00-57.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  57.00-58.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  58.00-59.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  59.00-60.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  60.00-61.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  61.00-62.00  sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6]  62.00-63.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  63.00-64.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  64.00-65.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  65.00-66.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  66.00-67.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  67.00-68.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  68.00-69.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  69.00-70.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  70.00-71.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  71.00-72.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  72.00-73.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  73.00-74.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  74.00-75.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  75.00-76.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  76.00-77.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  77.00-78.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  78.00-79.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  79.00-80.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  80.00-81.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  81.00-82.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  82.00-83.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  83.00-84.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  84.00-85.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  85.00-86.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  86.00-87.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  87.00-88.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  88.00-89.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  89.00-90.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  90.00-91.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  91.00-92.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  92.00-93.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  93.00-94.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  94.00-95.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  95.00-96.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  96.00-97.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  97.00-98.00  sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6]  98.00-99.00  sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6]  99.00-100.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 100.00-101.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 101.00-102.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 102.00-103.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 103.00-104.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 104.00-105.00 sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6] 105.00-106.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 106.00-107.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 107.00-108.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 108.00-109.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 109.00-110.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 110.00-111.00 sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6] 111.00-112.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 112.00-113.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 113.00-114.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 114.00-115.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 115.00-116.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 116.00-117.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
[  6] 117.00-118.00 sec  29.8 MBytes   249 Mbits/sec    0   2.68 MBytes
[  6] 118.00-119.00 sec  29.8 MBytes   250 Mbits/sec    0   2.68 MBytes
[  6] 119.00-120.00 sec  29.9 MBytes   251 Mbits/sec    0   2.68 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  3.49 GBytes   250 Mbits/sec    0
↪  sender
[  6]   0.00-120.04 sec  3.49 GBytes   250 Mbits/sec
↪  receiver

iperf Done.
```

Listing 39: iperf test result 250 Mb/s

E.1.4 500 Mb/s

```
Connecting to host 128.39.145.94, port 5201
[  6] local 10.0.123.138 port 60062 connected to 128.39.145.94 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  6]   0.00-1.00   sec  31.3 MBytes   263 Mbits/sec   14   1.72 MBytes
[  6]   1.00-2.00   sec  41.4 MBytes   347 Mbits/sec    0   1.74 MBytes
[  6]   2.00-3.00   sec  41.9 MBytes   351 Mbits/sec    0   1.76 MBytes
[  6]   3.00-4.00   sec  41.8 MBytes   350 Mbits/sec    0   1.78 MBytes
[  6]   4.00-5.00   sec  42.0 MBytes   352 Mbits/sec    0   1.79 MBytes
[  6]   5.00-6.00   sec  42.9 MBytes   360 Mbits/sec    0   1.81 MBytes
[  6]   6.00-7.00   sec  43.5 MBytes   365 Mbits/sec    0   1.88 MBytes
[  6]   7.00-8.00   sec  45.5 MBytes   382 Mbits/sec    0   1.96 MBytes
[  6]   8.00-9.00   sec  47.6 MBytes   400 Mbits/sec    0   2.07 MBytes
[  6]   9.00-10.00  sec  50.6 MBytes   425 Mbits/sec    0   2.21 MBytes
[  6]  10.00-11.00  sec  54.2 MBytes   455 Mbits/sec    0   2.38 MBytes
[  6]  11.00-12.00  sec  58.9 MBytes   494 Mbits/sec    0   2.59 MBytes
[  6]  12.00-13.00  sec  62.9 MBytes   527 Mbits/sec    0   2.73 MBytes
[  6]  13.00-14.00  sec  62.6 MBytes   525 Mbits/sec    0   2.73 MBytes
[  6]  14.00-15.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  15.00-16.00  sec  62.6 MBytes   525 Mbits/sec    0   2.73 MBytes
[  6]  16.00-17.00  sec  62.4 MBytes   523 Mbits/sec    0   2.73 MBytes
[  6]  17.00-18.00  sec  62.9 MBytes   527 Mbits/sec    0   2.73 MBytes
[  6]  18.00-19.00  sec  62.9 MBytes   527 Mbits/sec    0   2.73 MBytes
[  6]  19.00-20.00  sec  62.4 MBytes   524 Mbits/sec    0   2.73 MBytes
[  6]  20.00-21.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  21.00-22.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  22.00-23.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  23.00-24.00  sec  62.2 MBytes   522 Mbits/sec    0   2.73 MBytes
[  6]  24.00-25.00  sec  59.5 MBytes   499 Mbits/sec    1    257 KBytes
[  6]  25.00-26.00  sec  48.8 MBytes   409 Mbits/sec    0   2.10 MBytes
[  6]  26.00-27.00  sec  52.2 MBytes   438 Mbits/sec    0   2.27 MBytes
[  6]  27.00-28.00  sec  55.0 MBytes   461 Mbits/sec    0   2.40 MBytes
[  6]  28.00-29.00  sec  57.6 MBytes   483 Mbits/sec    0   2.50 MBytes
[  6]  29.00-30.00  sec  60.6 MBytes   509 Mbits/sec    0   2.58 MBytes
[  6]  30.00-31.00  sec  62.2 MBytes   522 Mbits/sec    0   2.64 MBytes
[  6]  31.00-32.00  sec  62.6 MBytes   525 Mbits/sec    0   2.68 MBytes
[  6]  32.00-33.00  sec  62.9 MBytes   528 Mbits/sec    0   2.71 MBytes
[  6]  33.00-34.00  sec  62.6 MBytes   525 Mbits/sec    0   2.73 MBytes
[  6]  34.00-35.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  35.00-36.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  36.00-37.00  sec  62.9 MBytes   527 Mbits/sec    0   2.73 MBytes
[  6]  37.00-38.00  sec  62.4 MBytes   523 Mbits/sec    0   2.73 MBytes
[  6]  38.00-39.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  39.00-40.00  sec  63.0 MBytes   528 Mbits/sec    0   2.73 MBytes
[  6]  40.00-41.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  41.00-42.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  42.00-43.00  sec  62.4 MBytes   523 Mbits/sec    0   2.75 MBytes
[  6]  43.00-44.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  44.00-45.00  sec  62.1 MBytes   521 Mbits/sec    0   2.75 MBytes
[  6]  45.00-46.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  46.00-47.00  sec  62.5 MBytes   524 Mbits/sec    0   2.75 MBytes
[  6]  47.00-48.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  48.00-49.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  49.00-50.00  sec  63.0 MBytes   529 Mbits/sec    0   2.75 MBytes
[  6]  50.00-51.00  sec  62.9 MBytes   527 Mbits/sec    0   2.75 MBytes
[  6]  51.00-52.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  52.00-53.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  53.00-54.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  54.00-55.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  55.00-56.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  56.00-57.00  sec  62.1 MBytes   521 Mbits/sec    0   2.75 MBytes
[  6]  57.00-58.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  58.00-59.00  sec  62.4 MBytes   523 Mbits/sec    0   2.75 MBytes
[  6]  59.00-60.00  sec  63.0 MBytes   529 Mbits/sec    0   2.75 MBytes
[  6]  60.00-61.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  61.00-62.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  62.00-63.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  63.00-64.00  sec  62.5 MBytes   524 Mbits/sec    0   2.75 MBytes
[  6]  64.00-65.00  sec  62.9 MBytes   527 Mbits/sec    0   2.75 MBytes
[  6]  65.00-66.00  sec  62.9 MBytes   527 Mbits/sec    0   2.75 MBytes
[  6]  66.00-67.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  67.00-68.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  68.00-69.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  69.00-70.00  sec  62.6 MBytes   525 Mbits/sec    0   2.75 MBytes
[  6]  70.00-71.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  71.00-72.00  sec  62.4 MBytes   523 Mbits/sec    0   2.75 MBytes
[  6]  72.00-73.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  73.00-74.00  sec  62.9 MBytes   527 Mbits/sec    0   2.75 MBytes
[  6]  74.00-75.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  75.00-76.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  76.00-77.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  77.00-78.00  sec  63.0 MBytes   528 Mbits/sec    0   2.75 MBytes
[  6]  78.00-79.00  sec  62.4 MBytes   523 Mbits/sec    0   2.75 MBytes
[  6]  79.00-80.00  sec  62.8 MBytes   526 Mbits/sec    0   2.75 MBytes
[  6]  80.00-81.00  sec  60.0 MBytes   503 Mbits/sec    0   2.75 MBytes
[  6]  81.00-82.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  82.00-83.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  83.00-84.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  84.00-85.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  85.00-86.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  86.00-87.00  sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6]  87.00-88.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  88.00-89.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  89.00-90.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  90.00-91.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  91.00-92.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  92.00-93.00  sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6]  93.00-94.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  94.00-95.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  95.00-96.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  96.00-97.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  97.00-98.00  sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6]  98.00-99.00  sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6]  99.00-100.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 100.00-101.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 101.00-102.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 102.00-103.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 103.00-104.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 104.00-105.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 105.00-106.00 sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6] 106.00-107.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 107.00-108.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 108.00-109.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 109.00-110.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 110.00-111.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 111.00-112.00 sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6] 112.00-113.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 113.00-114.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 114.00-115.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 115.00-116.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 116.00-117.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 117.00-118.00 sec  59.5 MBytes   499 Mbits/sec    0   2.75 MBytes
[  6] 118.00-119.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
[  6] 119.00-120.00 sec  59.6 MBytes   500 Mbits/sec    0   2.75 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  6]   0.00-120.00 sec  6.98 GBytes   500 Mbits/sec   15
↪  sender
[  6]   0.00-120.04 sec  6.98 GBytes   500 Mbits/sec
↪  receiver

iperf Done.
```

Listing 40: iperf test result 500 Mb/s

Appendix F

Ansible

F.1 Ansible Playbook

F.1.1 Role: iperf

```yaml
---

- name: Create a master-test namespace
  kubernetes.core.k8s:
    state: present
    definition:
      api_version: v1
      kind: Namespace
      name: "master-test" # defining the namespace
      metadata:
        name: "master-test"
  register: iperf_out

# - name: Print debug
#   ansible.builtin.debug:
#     var: iperf_out

- name: Deploy iperf pod with standard vxlan to Kubernetes
  kubernetes.core.k8s:
    state: present
    namespace: "master-test"
    src: unbuffer.yml

# Create iperf deployment then wait for it to run and get ip then start
↪  tcpdump on
- name: Wait until iperf-test-1 is up
  kubernetes.core.k8s_info:
    api_version: v1
    kind: Pod
    namespace: "master-test"
    label_selectors:
      - app = iperf-test-1
  register: iperf_pod_list
  until: iperf_pod_list|json_query('resources[*].status.phase')|unique ==
  ↪ ["Running"]

- name: Search for all Pods labelled app=iperf2
  kubernetes.core.k8s_info:
    kind: Pod
    label_selectors:
      - app = iperf-test-1
  register: iperf_pod_list

- name: Print podip
  ansible.builtin.debug:
    var: iperf_pod_list.resources[0].status.podIP

- name: Install iperf3 part 1
  kubernetes.core.k8s_exec:
    namespace: master-test
    pod: "{{ iperf_pod_list.resources[0].metadata.name }}"
    container: iperf-test-1
    command: apt update
  register: iperf_out
  changed_when: iperf_out.rc != 0

- name: Install iperf3 part 2
  kubernetes.core.k8s_exec:
    namespace: master-test
    pod: "{{ iperf_pod_list.resources[0].metadata.name }}"
    container: iperf-test-1
    command: apt install iperf3 -y
  register: iperf_out
  changed_when: iperf_out.rc != 0
```

Listing 41: Tasks for the iperf role

F.1.2 Role: tcpdump

```yaml
---

- name: Debug info
  ansible.builtin.debug:
    var:
  ↪ hostvars['localhost']['iperf_pod_list']['resources'][0]['status']['podIP']

- name: Debug info2
  ansible.builtin.debug:
    var:
  ↪ hostvars['localhost']['iperf_pod_list']['resources'][0]['metadata']['name']

```

```yaml
- name: Pause until you can verify updates to an application were
↪  successful
  ansible.builtin.pause:

- name: Start tcpdump
  ansible.builtin.command:
    cmd: >
      sudo tcpdump -G {{ dur_in_sec }} -W 1 -i enX0 -s 0
      -w {{ dest_folder }}/{{ pod_ip }}_{{ cap_file }} not port 22 and host
      {{ pod_ip }}
  register: tcpdump_real_out
  async: 180
  poll: 0

# - name: Press enter to start iperf3
#   ansible.builtin.pause:

# 128.39.145.94

- name: Start iperf3 against uia server
  kubernetes.core.k8s_exec:
    namespace: master-test
    pod: "{{ pod_name }}"
    container: iperf-test-1
    command: iperf3 -c 128.39.145.94 -b 1000000000 -t 10
  register: tcpdump_command_status
  delegate_to: localhost

- name: Check iperf
  ansible.builtin.debug:
    var: tcpdump_command_status
  when: tcpdump_command_status.rc != 0

- name: Check tcpdump finished
  ansible.builtin.async_status:
    jid: "{{ tcpdump_real_out.ansible_job_id }}"
  register: tcpdump_job_result
  until: tcpdump_job_result.finished
  retries: 100
  delay: 5

- name: Compress capture file
  ansible.builtin.command:
    cmd: "sudo gzip {{ pod_ip }}_{{ cap_file }}"
    chdir: "{{ dest_folder }}"
  register: tcpdump_out
  changed_when: tcpdump_out.rc != 0

- name: Change file permission
  ansible.builtin.command:
    cmd: sudo chmod 755 {{ dest_folder }}/{{ pod_ip }}_{{ cap_file }}.gz
  register: tcpdump_out
  changed_when: tcpdump_out.rc != 0

- name: Copy logs to /tmp/ansible/
  ansible.builtin.fetch:
    src: "{{ dest_folder }}/{{ pod_ip }}_{{ cap_file }}.gz"
    dest: /tmp/ansible/
    flat: true

- name: Decompress locally
  ansible.builtin.command:
    cmd: "gzip -d {{ pod_ip }}_{{ cap_file }}.gz"
    chdir: /tmp/ansible
  register: tcpdump_out
  changed_when: tcpdump_out.rc != 0
  delegate_to: localhost

# - name: Remove files from remote server
#   ansible.builtin.command:
#     cmd: sudo rm -rf {{ dest_folder }}/{{ cap_file }}.gz
#   register: tcpdump_out
#   changed_when: tcpdump_out.rc != 0

```

Listing 42: Tasks for the tcpdump role