Cato Findalen
Jostein Furnes

Bachelor's thesis

Secure deployment of applications in Kubernetes on Google Cloud

June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Design

Bachelor’s thesis

2020

Cato Findalen
Jostein Furnes

Secure deployment of applications in Kubernetes on Google Cloud

Bachelor's thesis
June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Design

NTNU
Norwegian University of
Science and Technology

Secure deployment of applications using Kubernetes in Google Cloud

Jostein Furnes
Cato Findalen Røsvik

June 10, 2020

Abstract

Security, efficiency and accessibility do not always go hand in hand, but can they? Kubernetes as a platform by default might not give the correct answer but by using a combination of existing functionality and coming ones, it can be easier than ever. Headit AS is a software development and integration company located in Hamar, Norway, and our theises employer. They asked if we could look into secure operations of Multi-tennant solutions within Kubernetes and if we had recommendations for improving their current Kubernetes system, as well as maping out ways to improve their current solution both in short and long term in regards to performance, setup and configuration.
In order to do these investigations, Headit provided us with a test environment within Google Cloud and a demo application running based on their current system operation.
In this test environment, we dismantled the security added through Network Policies by removing them one at a time and monitoring the cluster for Network traffic between Pods. In addition, we also conducted tests at different stages in order to validate our findings. Through in depth testing of provided materials and research into the platform, we can make Kubernetes go from unsecure to secure and efficient through modern added functionality and other services added to the Kubernetes environment.
To recommend improvements to Headit, we researched different solutions in order to make operating, setting up and updating a Kubernetes environment.

Sammendrag

Sikkerhet, effektivitet og tilgjengelighet går ikke alltid hånd i hånd, men kan det? Kubernetes er en usikker platform, og vil kanskje ikke sees som en riktig løsning for spørsmålet, men gjennom kombinasjoner av eksisterende og funksjoner under utvikling kan det bli veldig enkelt.
Headit AS er ett utviklingsfirma fra Hamar, Norge, og oppdraggiveren for denne oppgaven. De ville at vi skulle undersøke hvordan kjøre Multi-tennant løsnigner i Kubernetes på en sikker måte, og om vi hadde idéer til hvordan forbedre på ders nåværende Kubernetes system. Samtidig skulle vi kartlegge muligheter til å forbedre løsningen på kort og lang sikt med tanke på ytelse, oppsett av programmer og konfigurasjon.
For å gjøre disse undersøkelsene fikk vi tilgang til ett test miljø i Google Cloud, og demo applikasjoner basert på nåværende måter å kjøre applikasjoner i systemet.
Undersøkelsene våre baserte seg på å fjerne nettverks policyene en etter en for deretter å monitorere trafikken mellom Pod-er ved hjelp av Kali. Vi gjennomførte disse undersøkelsene flere ganger under gjennomføring av oppdaven for å validere dataene.
Gjennom grundige tester av de gitte ressursene og fordypning i platformen kan man trekke Kubernetes fra en usikker platform fra standardoppsettet til en sikker og effektiv plaform gjennom eksisterende og kommende funksjoner, med ekstra tjenester lagt til i Kubernetes miljøet.
I forbindelse med det å anbefale forbedringer til Headit så undersøkte vi forskjellige løsninger som hjelper med bruk, oppsett og oppdattering av ett Kubernetes miljø.

Preface

This Bachelor thesis is the end for the 3 year study "IT-drift og Informasjonssikkerhet (BITSEC)" and "Batchelor i ingeniørfag - Data (BIDAT)". It was done in the spring of 2020 at NTNU Gjøvik. The employer of this assignment is Headit with Rune Gaade as their representative.

The thesis has been challenging and an eye opening experience for the whole group. We have built on information and experiences we have gathered throughout our studies, as well as get new information we never expected to need. This is something we look forward to bring with us out into our futures.

We would like to thank Headit AS and especially Rune Gaade for all the help they have provided, answering all questions we have had, and provided relevant context where we have needed it.
We would also like to thank our supervisor Jia-Chun Lin for pushing us to work harder, and helping us do as best we can with this thesis, as well as helping out when we haven't known fully how to make progression.

Contents

Figures

Tables

Acronyms

API Application Programming Interface [1]. xi, 2, 6, 7, 15, 19–21, 23, 24

CVE Common Vulnerabilities and Exposures [2]. 2

GCP Google Cloud Platform [3]. 6

GKE Google Kubernetes Engine [4]. 1, 2, 10–14, 17, 18, 26, 27

HTTP Hypertext Transfer Protocol [5]. 12, 13, 19

JSON JavaScript Object Notation [6]. 32, 34

K8s Kubernetes. 5

NMAP Network Mapper [7]. 25

YAML YAML Ain't Markup Language [8]. 11, 17, 32, 34

Glossary

apt-get Apt-get is a package manager for debian based and related linux distributions. 19, 22

capabilities "*Linux capabilities provide a subset of the available root privileges to a process. This effectively breaks up root privileges into smaller and distinctive units. Each of these units can then be independently be granted to processes. This way the full set of privileges is reduced and decreasing the risks of exploitation*".[9]. 17

docker-image Docker images are containerized prebuilt applications that run on the docker and kubernetes platforms, usually they are tailored to perform only one task.. 13, 15

Java Java is a class-based and object oriented programming language.. 13

Kubectl Kubectl or kube control is a commandline tool used to control the components in a kubernetes cluster. 10, 11, 19, 24

Multi-Tenant Multi tenant describes two or more infrastructures running on the same hardware or cloud, each of the infrastructures belong to different "tenants" or customers. . 1, 40

NET_ADMIN NET_ADMIN or CAP_NET_ADMIN is a capability in linux for out of ordinary operations on network adapters and similar hardware, the capability can be given to binary's to gain more privileges on the network adapters . 17

NET_RAW NET_RAW or CAP_NET_RAW is a capability in linux given to binary's to use RAW and PACKET sockets, this gives the binary the ability to bind to any address for transparent proxying. 17

Nginx Nginx is a web server, reverse proxy and load-balancer.. 12

openjdk:12-alpine openjdk:12-alpine is a docker image with the Open Java Development Kit pre-installed, running on the lightweight alpine linux project.. 13

Spring Boot Spring boot is an extension for the spring framework enabling fast deployment of java applications in new enviroments such as docker images. The java applications can be build quickly upon container deployment.. 13

Chapter 1

Introduction

This thesis focuses on Kubernetes and the use of Multi-Tenant applications run within a Kubernetes system. In this chapter, we will introduce Headit (the provider of this project assignment), project goals, project requirements, boundaries, project groups, etc.

1.1 Background

Our thesis assignment was provided to us by Headit AS, which is a Norwegian business specializing in data science, system development and business development. Headit AS design and develop backend systems for their customers and maintain and run those systems in the cloud. Their customers' systems running inside containers on the (Google Kubernetes Engine [4] (GKE) is often set up as a Multi-Tenant configuration. They wanted to know if an attacker can make use of one customer's system to attack another customer's system or not given that these two systems are deployed in the same Kubernetes cluster.

1.2 Project Goal

This project aims to achieve the following two goals.
Part 1: Security investigation
This part focuses on exploring what part of the Kubernetes environment is exposed to attacks from external sources. In order for us to work on this project, Headit provided us with an isolated test environment. The environment is set up with two simple demo applications, configuration for a REST-controller, intentity provider (IDP) and a database. Our goal is to investigate if the configuration is secure so that different customer applications running in the same cluster will not be able to interfere (or even to attack) each other.

Part 2: Improvements to the system
The second part of the thesis focuses on investigating how the current Kubernetes implementation can be improved in terms of security, performance, ease of use, configuration etc. as well as suggestions for Docker images that could be used as a base OS and host for the different applications to be run within the Cluster. These different applications consist of, but might not be limited to: webservers, APIs, databases and identity providers.

1.3 Requirements

Headit haven't specified any requirements for us. They have asked the group to work with what we want to gather experience and investigate within the two parts of the assignment.

1.4 Boundaries

Given the amount of flexibility in this assignment we have found it better to set some boundaries for what we want to investigate, and we have decided to take away the things we are unable to control from the investigation. Since GKE takes care of patching and security mitigation without any user influence, we have chosen not to test for any Common Vulnerabilities and Exposures [2] (CVE)'s surrounding Kubernetes itself. Any exploits there should already be patched or will be soon by google anyways, and since Headit most likely wouldn't be able to take any action against them anyway there's no reason to investigate them. We will only look at how the current implementation is in the demo application provided to us.

1.5 Project Group

The members of the group are from two different studies at NTNU Gjøvik. Jostein Furnes studies "Batchelor i ingeniørfag - Data" and Cato F. Røsvik studies "IT-drift og Informasjonssikkerhet". Before this thesis the group had very little knowledge about Kubernetes and Google Cloud. We had to do a deep dive into the different components of the solutions before we could start the full investigation. As well as go in depth on external solutions for improving Kubernetes for looking at what to recommend in order to improve the current Solution used by Headit today.

1.6 Thesis Structure

- Chapter 1 **Introduction:** Introduction to the thesis

- Chapter 2 **Background:** This chapter provides necessary background information for the thesis.

- Chapter 3 **Security Investigation:** This chapter introduces how we investigate if Headit's current Kubernetes configuration is secure or not.

- Chapter 4 **Improvements to Headits current system:** This chapter suggests a set of services that we found can improve Headit's current Kubernetes service.

- Chapter 5 **Closing Remarks:** This chapter summarizes the thesis and outlines future work.

Chapter 2

Background

In this chapter we will describe background knowledge and terminology needed to understand Kubernetes and Google cloud.

2.1 Kubernetes

Kubernetes [10] is an open source container orchestration platform. It is also known as "Kube" and "Kubernetes (K8s)" in communities that works with this regularly. It automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

Kubernetes clusters are quite adaptable, and can span many different hosts. This can be used in conjuncture with hosts located on-premise, public, private or hybrid clouds. This functionality makes Kubernetes an ideal platform for hosting cloud-native applications that require rapid scaling. For our thesis this is relevant for Headit's hosting of their clients' applications within cloud environments where they can easily scale up, and down as use varies.

Within Kubernetes a lot of different terminologies are needed to be understood. The following are the common elements within Kubernetes that are quite useful to know about as a baseline.

- Master [11]: The main machine that controls nodes within the Kubernetes cluster. All task assignment originate from this machine. It is also called Kubernetes Master.
- Node [12]: Machines performing requested and assigned tasks. They are controlled by the Kubernetes Master.
- Pod [13]: A group of one or more containers that share the same IP address, identity provider, hostname and other resources. This constant in the configuration allow for easy transfer of containers around clusters.
- Replication controller [14]: A controller within the cluster responsible for controlling the amount of identical copies of pods running within the cluster.

- Service proxy [15]: The Service Proxy sends traffic from the client to a backend system. If a user requests something to be done within a Cluster, the Service proxy is responsible for the request to be sent to the correct receiver and take the requested information back to the user.
- Kubelet [16]: Service that runs within a node, reads container manifests and ensures the defined containers are started and running. It also corrects errors if discovered.
- Kubectl [17]: It is a command line tool for controlling Kubernetes clusters. It uses configuration files and user commands to control clusters and elements within it.
- Deployment [18]: Way to update the contents and assignments of clusters. Desired state is described by the user, and can create new, pause, rollback and scale up and down the elements of a Cluster.

2.2 Google Cloud

Google Cloud Platform [3] (GCP) is a Google product that offers the same infrastructure that Google uses internally in their own systems for its end-user products. In addition it also offers a set of tools in order to manage the services. Some of Google clouds uses are:

- Data Storage
- Computing
- Data Analytics
- Machine Learning

Within these services over 90 products are listed under the Google Cloud branding, and within this Kubernetes is one of these products. We will focus on the parts of Google Cloud that goes under Computing and , as these are the elements of the service most relevant to the thesis.

2.3 Network Protocols within Kubernetes

Kubernetes pods are by default unsecure, with open access to everything. For these reasons, configuring network policies is a necessary thing for standard operation. Network policies are implemented into Kubernetes as a solution to limit communications between pods within a cluster. The network policies are generally written by the people creating pods, or set as a standardized part of a businesses operation of the cluster. Network policies are also used to allow communications with other parts of a network. These parts could be external databases and APIs. The endpoints are devices connected to specific Local wired, or wireless networks.

Within a Kubernetes cluster, all the pods are by default allowed to communicate with each other and any other network endpoint. This means that traffic between pods are visible to other pods. Network policies are used to limit the open traffic, and apply restrictions where needed.

In order to use a network policy, it is required to use a networking solution that supports it. This is usually handled by installing a network plugin, and making a Kubernetes controller [17] implement them. Without the plugin or controller applying policies, any policies written will have no effect on the system.

A network policy contains the following elements as a minimum:

- apiVersion
- kind
- metadata

apiVersion: It tells the Kubernetes cluster what functions to have access to within the cluster. The application programming interface (API) within Kubernetes updates frequently, and features are constantly added. These could cause problems, so limiting the functions available by defining what the API can access is a must to avoid issues. For most usecases using v1 will be sufficient. v1 is the current updated API working for Kubernetes, without any extra modules outside the intended functionalities of Kubernetes.

However if there are some future features a user wants to test, like Cluster Roles or Certificate signing, he/she might want to use different APIs, such as "Certificates.k8s.io/v1beta1" or "rbac.authorization.k8s.io/v1". Whenever "Alpha" or "Beta" in their names, they are candidates for new features for future functionality within the Kubernetes service. Alpha and Beta indicates how far the features has progressed in testing and development. These versions might contain bugs, and the final added version added to Kubernetes might be changed before the final stable version is released.

kind: This tells the kubectl [17] (Kubernetes controller) what object the user wants to create. A few examples of objects: deployment[18] and pod [13].

Metadata: These are labels or things that the user wants to record as information within the deployment or other elements within the cluster, like pods or nodes. Metadata can be quite useful if the user is working with big clusters and he/she needs to list all pods and deployments with a certain feature logged in the Metadata.

Chapter 3

Kubernetes Security Investigation

In this chapter, we will investigate how secure Headit's configuration of Kubernetes at Headit is. Firstly we introduce all tools we used to conduct the investigation and then describe the demo application provided by headit. After that, we will show how we conducted the security investigation and discuss the corresponding result.

3.1 Used Tools

In order to properly study the security of the Kubernetes configuration we have choosen the following tools:

- Kali Linux Docker image
- Wireshark and Tshark
- Kubectl
- Nmap
- Ksniff

3.1.1 Kali Linux

Kali linux [19] is a flavor of linux designed for security testing and penetration testing, we chose to use a docker image of kali linux because of it's good selection of available penetration testing tools. Due to the popularity of Kali Linux, there should be less problems with troubleshooting if problems arise. This is the reason we chose it.

3.1.2 Wireshark and Tshark

Wireshark [20] is open-source network packet analyzer used for network analysis and troubleshooting. Wireshark uses a graphical user interface to represent the captured network packets. On the other hand, tshark is the headless version [21] of wireshark without a graphical user interface, but with all the same functionality in regards to collecting network packets. We used tshark to collect packets visible to the kali pod and analysed the results with wireshark on a different computer.

3.1.3 Kubectl

[17] is a command line tool used to controlling kubernetes clusters. We used it for deploying the infrastructure to GKE and execute commands inside the pods. The output of those commands were also piped [22] to our local computers.

3.1.4 Nmap

Nmap [7] is a network port mapping tool, used to scan a subnet or network address and look for active hosts to see if any ports are open on those hosts. We used nmap for trying to map the subnet of the cluster and all pods within it.

3.1.5 Ksniff

Ksniff [23] is a plugin for Kubectl that attaches to running pods to sniff their network traffic. Ksniff is launched via kubectl with the pod name you want to attach it.

3.2 Demo Application Provided by Headit

Headit provided a demo application with a simple REST API for us to investigate if there is any security issue with the corresponding configuration. When the API is called, the application returns a json formatted list of books. The configuration is defined in a set of .YAML Ain't Markup Language [8] (YAML) files ready to deploy within the Google Kubernetes Engine (GKE) with the Kubectl tool.

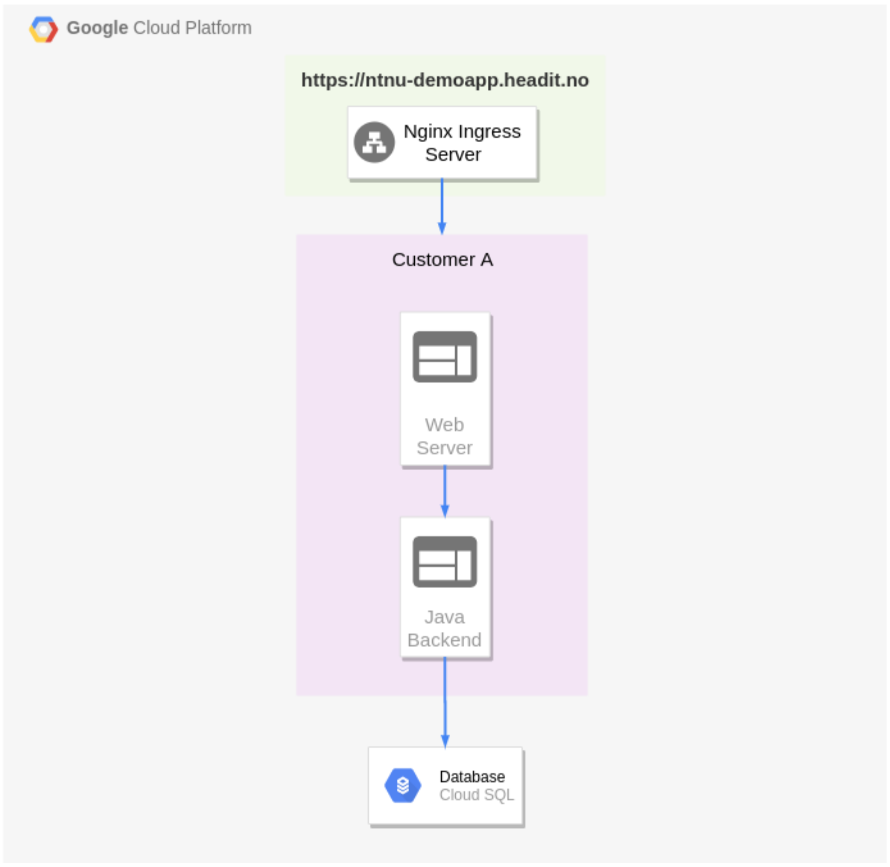


Figure 3.1: Illustration of the original demo application provided by Headit.

Figure 3.1 illustrates the demo application, which consists of the following elements:

- Nginx ingress server
- Web server named: headit-demoapp-web
- Java backend server named: headit-demoapp-svc
- Database
- Network policies

3.2.1 Nginx Ingress Server

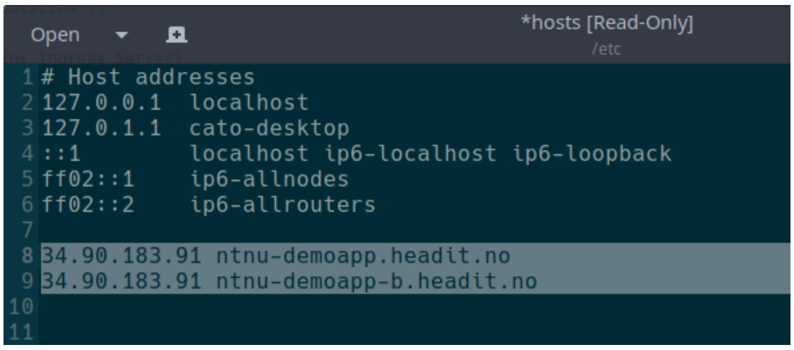
Nginx ingress server is a builtin function in GKE providing a remotely accessible endpoint with a public facing IP address. Nginx Ingress server has the ability to balance traffic load between pods and act as a reverse proxy. Note that a reverse proxy is a web server that retrieves resources from one or more other servers, these servers are usually on the same local network as the reverse proxy[24]. Since this demo application does not host the same content on more than one server, load-balancing is not needed.

In this project, the ingress server is set up as a reverse proxy and routes incoming network traffic to the local web server on port 80. It provides a public IP and domain name called:

```
ntnu-demoapp.headit.no
```

This domain is not registered in any public DNS registers, meaning that it is not automatically accessible like most other websites. Reverse proxies can host multiple domains on a single public IP with a technique called Virtual hosting [25]. To accomplish this the reverse proxy looks for the destination domain in the Hypertext Transfer Protocol [5] (HTTP) header sent by the user to distinguish what local server the traffic was intended for.

Since the domain is not publicly connected to the public IP nginx uses, the client machine needs to edit its local DNS register to make that connection. Most operating systems like windows, linux and mac use a hosts file where the user can link IP addresses to domains like in Figure 3.2 below.



```
 1 # Host addresses
 2 127.0.0.1  localhost
 3 127.0.1.1  cato-desktop
 4 ::1        localhost ip6-localhost ip6-loopback
 5 ff02::1    ip6-allnodes
 6 ff02::2    ip6-allrouters
 7
 8 34.90.183.91 ntnu-demoapp.headit.no
 9 34.90.183.91 ntnu-demoapp-b.headit.no
10
11
```

Figure 3.2: Modified client hosts file

As shown on lines 8 and 9 in Figure 3.2, two different domains are linked to the same public IP address. When the user accesses either one, the domain name is included in the HTTP headers, and the Nginx server sends the request to the local web server hosting that domain. Trying to access the IP address directly would result in Nginx not knowing what local web server the traffic was intended for, and instead return a "*404 Page Not Found*"[26] error.

3.2.2 Web Server

The web server is based on the docker-image "*openjdk:12-alpine*", with Java already installed on the docker-image inside the lightweight alpine variant of docker images. A set of Spring Boot scripts made by Headit configures the web server when the pod deploys. The docker image exposes its internal port 8080 to port 80 outside the pod facing the ingress server and the Java backend server.

The web server uses a Java application to provide a REST API with two defined API calls:

```
/getAllBooks
/getBookFromAPI
```

A HTTP or https request with these API calls to the configured domain name will return two different lists of books. The demo application is set up to respond to API requests on these two complete links:

```
https://ntnu-demoapp.headit.no/getAllBooks
https://ntnu-demoapp.headit.no/getBookFromAPI
```

"*/getAllBooks*" will query the Java backend server for a list of books, while "*/getBookFromAPI*" will get a json list of books from:

"*http://openlibrary.org/books/OL16370673M.json*".

3.2.3 Java Backend Server

Similar to the webserver, the backend server is based on the docker-image "*openjdk:12-alpine*" with Spring Boot scripts that is executed upon container deployment. This container listens on port 80 from the web server. When a /getAllBooks request is sent, it retrieves the book list from the database and returns it back to the web server.

3.2.4 Database

The database is also a builtin function of GKE, so we do not have much information on it except for a initializing script that create a new database. The script creates a new database with a predefined account and password and then grants all necessary privileges to that user. The user credentials and database name are reflected in the Spring Boot scripts on the Java backend server.

3.2.5 Network Polices

The most important security implementation in the demo application are network policies. Headit has followed best practice here in accordance with Google's recommendations [27]. GKE is set up with a labeling system, so network policies can be defined to apply to any container with a given label, the demo application uses labels in the recommended way. All the network policies are as follows:

deny_all.yaml[Please see **A.1** for details]:
This network rule denies all traffic by default, and this rule applies to all pods in the cluster. This policy exists to deny all traffic that are not defined in any of the other rules. This rule is best practice and allows the user control of what traffic can pass inside the cluster.

allow-mysql-access.yaml[Please see **A.2** for details]:
This rule is applies to all pods with the label "*allow-mysql-access*", the pods with this label are allowed to speak with the mysql server on the default port on a different subnet.

allow-internet-access-http.yaml [Please see **A.3** for details]:
This rule grants access to 0.0.0.0/0 or all subnets, with the exception of 10.0.0.0/8, which are the local subnet the pods in the cluster uses. This policy applies to pods with the label "*allow-internet-access-http*" and grants them access to the internet, but not pods in the cluster.

allow-ingress-to-web-tier.yaml [Please see **A.4** for details]:
This rule allows the ingress server to receive traffic from any pod with the label: "*web*". Without this rule, the deny-all.yaml policy would block this kind of traffic.

allow-egress-to-dns.yaml [Please see **A.5** for details]:
This rule grants DNS[28] lookup to all pods in the cluster by opening port 53. This rule enables all pods to resolve domain names to IP addresses.

allow-egress-from-headit-demoapp-web-to-headit-demoapp-svc.yaml [Please see **A.6** for details]:
This rule allows all traffic from demoapp-web(Web Server) to demoapp-svc(Java Backend Server) listed in section 3.2. Without this rule the deny-all.yaml would block the traffic.

allow-ingress-to-headit-demoapp-svc-from-headit-demoapp-web.yaml[Please see **A.7** for details]:
This rule is just a reversed version of the previous rule. Without this rule demoapp-web can talk to demoapp-svc but can not receive an answer from demoapp-svc.

3.3 Method

In this section we explain how we created a testing infrastructure inside the cluster, how we configured it, and what tools we used to conduct the investigation. For each tool we explain how the testing was conducted, what commands were used, then show the results of the tests.

3.3.1 Setup

As the demo application shown in Figure 3.1, the current infrastructure contains only one customer. We call it customeer A. The main concern for Headit was if a potential attacker can gain access to a different customer in the same cluster or not. In order to simulate a environment, we duplicated the configuration files of the demo application, made another identical deployment with the same functionality, and called it customer B. Figure 3.3 shows the resulting cluster. Pod names were changed from headit-demoapp-xxx to headit-demoapp-xxx-b. This change would be enough to create a new set of pods but not make them function as intended. Hence the last two network policies mentioned in 3.2.5 needed to be duplicated as **A.10** and **A.11** for new and modified policies).
These two new rules enabled both the web and java backend server of the newly created customer B to talk to each other. The labeling with tier web/allow-mysql-access/etc remains the same since customer B's pods runs the same services as customer A. Customer B also needs a domain name so the REST API can be reached over the internet. Therefore a new domain called "ntnu-demoapp-b.headit.no" was implemented on the Ingress server linked to the new headit-demoapp-web-b pod.

3.3.2 Kali Linux

With two working customer deployments ready in the cluster, we could start to do some testing. Installing the test programs needed directly on the demo application was decided against. This was due to the minimal design of the alpine based docker images, which are mainly designed for running pre-installed applications and not security testing.

Instead we decided to implement a worst case scenario, by deploying a standalone Kali linux container inside the customer A side of the cluster. With this new container we wanted to sniff network packets to see if any of customer A or customer B's network traffic leaked through anywhere.

We deployed a vanilla "*Kali-linux:latest*" docker-image with the following network policy labels:

- allow-internet-access-http
- web

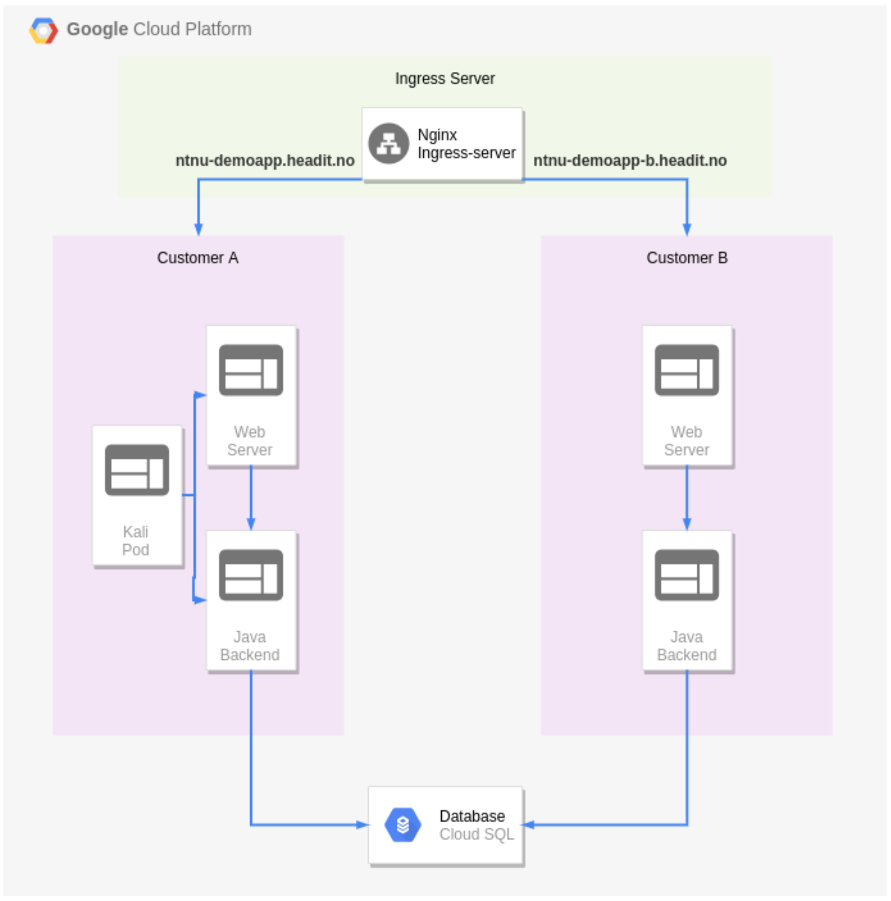
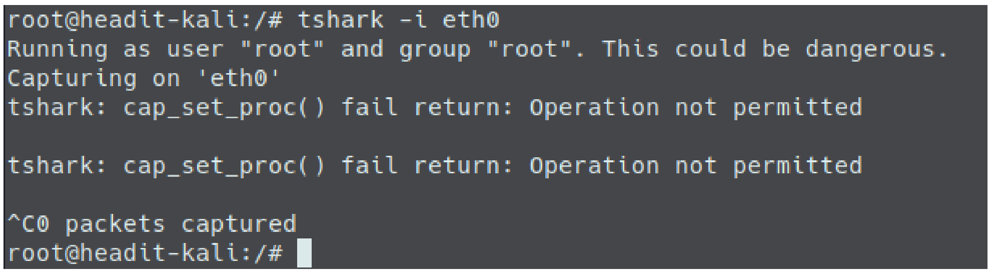


Figure 3.3: Illustration of the newly created multi-tenant cluster

Since the cluster pods are headless [21] and does not provide a graphical user interface, we opted to use Tshark as the packet sniffer and later analyse the sniffed data with Wireshark. We ran into a lot of trouble in this stage, because kubernetes has several ways of starting and running containers. The simplest way is to use the run command that takes very limited options, but works great for short term testing like this.

```
kubectl run --generator=run-pod/v1 headit-demoapp-kali -i --tty \
--image kalilinux/kali \
--labels="access-to-internet-http=true,app=demoapp-kali,tier=web" \
-- bash
```

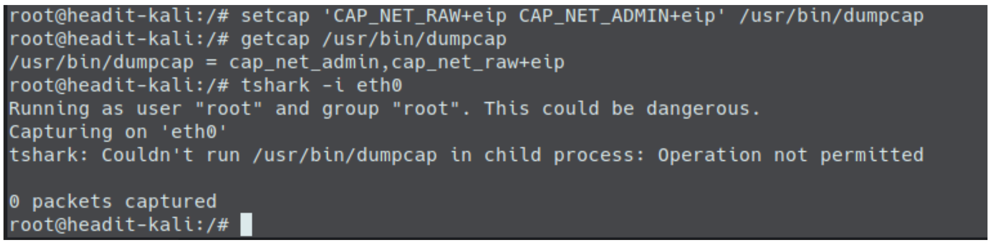
This command deploys the pod straight into a root shell with only basic tools/-binary's installed including a package manager. In the root shell we install Tshark and when we try to run Tshark, we get an error shown as below:



```
root@headit-kali:/# tshark -i eth0
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
tshark: cap_set_proc() fail return: Operation not permitted

tshark: cap_set_proc() fail return: Operation not permitted

^C0 packets captured
root@headit-kali:/# 
```

Figure 3.4: Error shown when trying to run tshark for the first time

After some research into this error, we discover that a binary called dumpcap lacks some capabilities [29]. Dumpcap[30] is a network traffic dump tool used by tshark to sniff packets from an ethernet adapter. After identifying what capabilities were missing, we provide the dumpcap binary the missing capabilities (NET_RAW and NET_ADMIN) which gives the binary full access to the ethernet adapter the pod uses. This produces a new error shown in Figure 3.5.

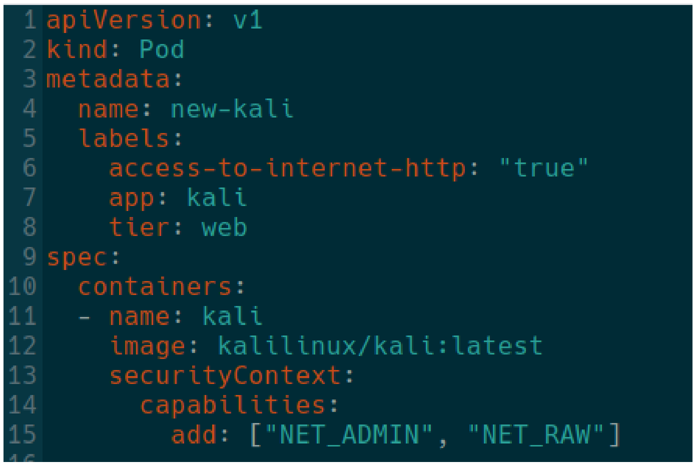


```
root@headit-kali:/# setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/dumpcap
root@headit-kali:/# getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
root@headit-kali:/# tshark -i eth0
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
tshark: Couldn't run /usr/bin/dumpcap in child process: Operation not permitted

0 packets captured
root@headit-kali:/# 
```

Figure 3.5: Another error occurs after we applied the two missing capabilities.

This error took us a very long time to resolve, and it postpones our progress on the project by weeks. Eventually we found out that these capabilities were not available on the GKE host node where the pods are running when the pods are started with the "run" command. No options for adding the capabilities within the run command exists either. Therefore when Kali thought it had the capabilities, it was denied to use them by the host node it was running on and thereby produced the above error shown in Figure 3.5. There were no logs within GKE or the system logs that reflected any of this, which made it so much harder to debug and resolve.

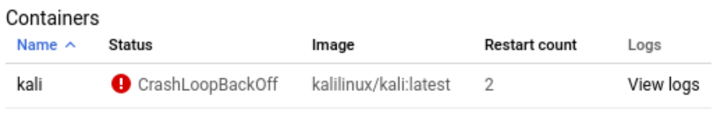
Running tshark within GKE is apparently not too popular either since all the online research we did returned no comparable situations. Looking into how capabilities interacts with docker and kubernetes, we eventually found out that the run command was the problem. It has a predefined set of capabilities, and it is not possible to add more. This meant that we had to launch Kali from a .YAML file instead, only then could we add the missing capabilities to the pod as Figure 3.6 shows.



```
apiVersion: v1
kind: Pod
metadata:
  name: new-kali
  labels:
    access-to-internet-http: "true"
    app: kali
    tier: web
spec:
  containers:
  - name: kali
    image: kalilinux/kali:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "NET_RAW"]
```

Figure 3.6: Yaml file for deploying kali with correct capabilities

However when applying this valid configuration, the pod crashes in an endless loop as shown in Figure 3.7.



Containers

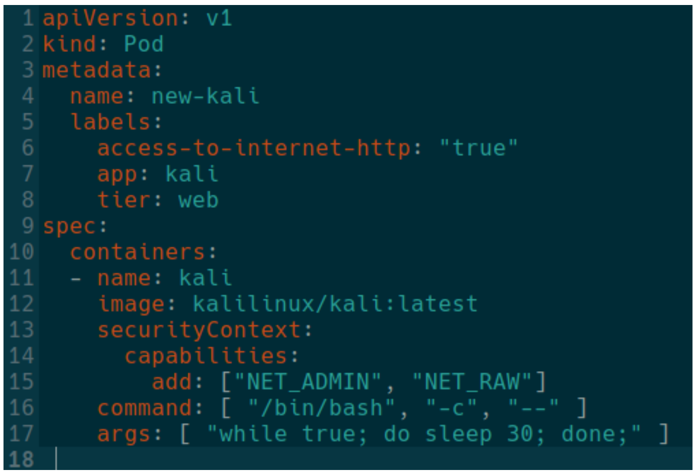
Name ^	Status	Image	Restart count	Logs
kali	CrashLoopBackOff	kalilinux/kali:latest	2	View logs

Figure 3.7: CrashLoopBackOff error from GKE web interface

Just like our problem shown in Figure 3.5, no logs are available to help in resolving the issue. The kali container produces zero logs since it never gets to start, and GKE's event logs show an ever expanding list of CrashLoopBackOff. The solutions we found on the internet while searching for "CrashLoopBackOff" combined with multiple other keywords only gave answers to a lot of things that were not related to our actual problem. We also tried a docker-compose locally with the same setup, but there was no problem launching the container. Therefore the issue was not with the docker image or configuration, but probably was a kubernetes quirk of some kind.

After more time spent trying things that were not relevant at all, we figured out that since the kali image had no scripts or tasks configured, kubernetes would see the pod was "idle" and then shut it off before starting it up again and repeating this cycle. Since containers often are deployed on demand for doing one task and then shut down again, it was treated as "finished" and shut down. Kubernetes does not mention this in the logs or anywhere on the wiki, we could find with the only keyword we had "*CrashLoopBackOff*".

The problem was then resolved by adding a task in the configuration file that would never end, shown in Figure 3.8.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: new-kali
  labels:
    access-to-internet-http: "true"
    app: kali
    tier: web
spec:
  containers:
  - name: kali
    image: kalilinux/kali:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "NET_RAW"]
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
```

Figure 3.8: Modified yaml file that does not produce the CrashBackLoopOff error

With a working configuration file, we could now adapt it further by opening the same ports used by the web server and java backend server. Our final kali configuration can be viewed in appendix **B.1**.

With a fully configured and running container we could now install all the needed tools and start testing. Since the kali docker image contains almost no programs to begin with, we used apt-get to download what we needed. The corresponding commands are shown below:

```
apt update
apt install -y tshark nmap net-tools iproute2 iputils-ping /
curl traceroute

Should non-superusers be able to capture packets?: no
setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/dumpcap
```

3.3.3 Tshark Testing

With Tshark properly installed and working, we used Kubectl to execute the tshark program on the kali pod and piped[22] the output to a locally running instance of wireshark with the following command:

```
kubectl exec headit-demoapp-kali -- bash -c \
"tshark -i eth0 -w -  2>/dev/null" | sudo wireshark -k -i -
```

This command enables us to see all the traffic that our kali container can view on its ethernet adapter, represented on a locally running instance of wireshark.
The illustration in Figure 3.9 shows the traffic flow when the API is triggered from outside the cluster. The API request reaches the ingress server which then determines the destination domain name based on the HTTP headers mentioned

in 3.2.1. The ingress server then forwards the API request to the destination, which is either customer A or B's web server. The webserver then forwards the request to the java backend server which then makes a database query to the database. The traffic then goes all the way in reverse back to the user who made the API request. The traffic flows down the red and green arrows, when the data is retrieved from the database it returns in the direction of the hollow colored arrows in Figure 3.9 below.

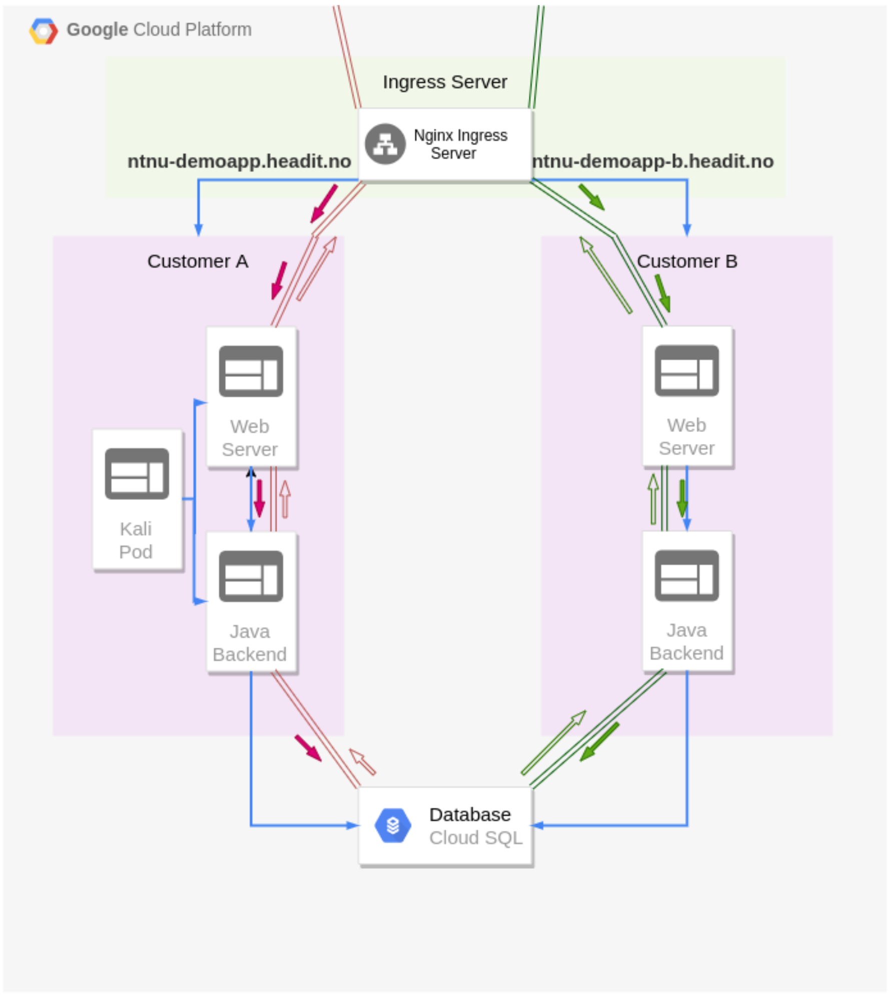
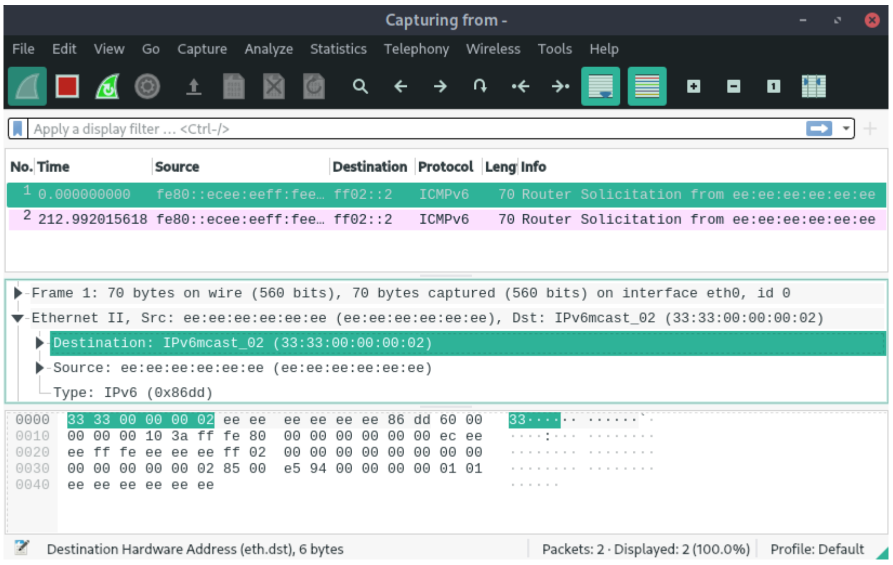


Figure 3.9: Illustration of the network packets path when the API is triggered, shown going up and down the green and red lines.

The screenshot in Figure 3.10 shows wireshark after running for about 5 minutes with only two ICMPv6 packets collected/sniffed. During the 5 minutes of sniffing, two API requests were sent to customer both A and B with the following two commands:

```
curl https://ntnu-demoapp.headit.no/demoapp/getAllBooks
curl https://ntnu-demoapp.headit-b.no/demoapp/getAllBooks
```

This was done to make sure packets were flowing as depicted in Figure 3.9, but the kali pod was not able to sniff them as shown in Figure 3.10 below.

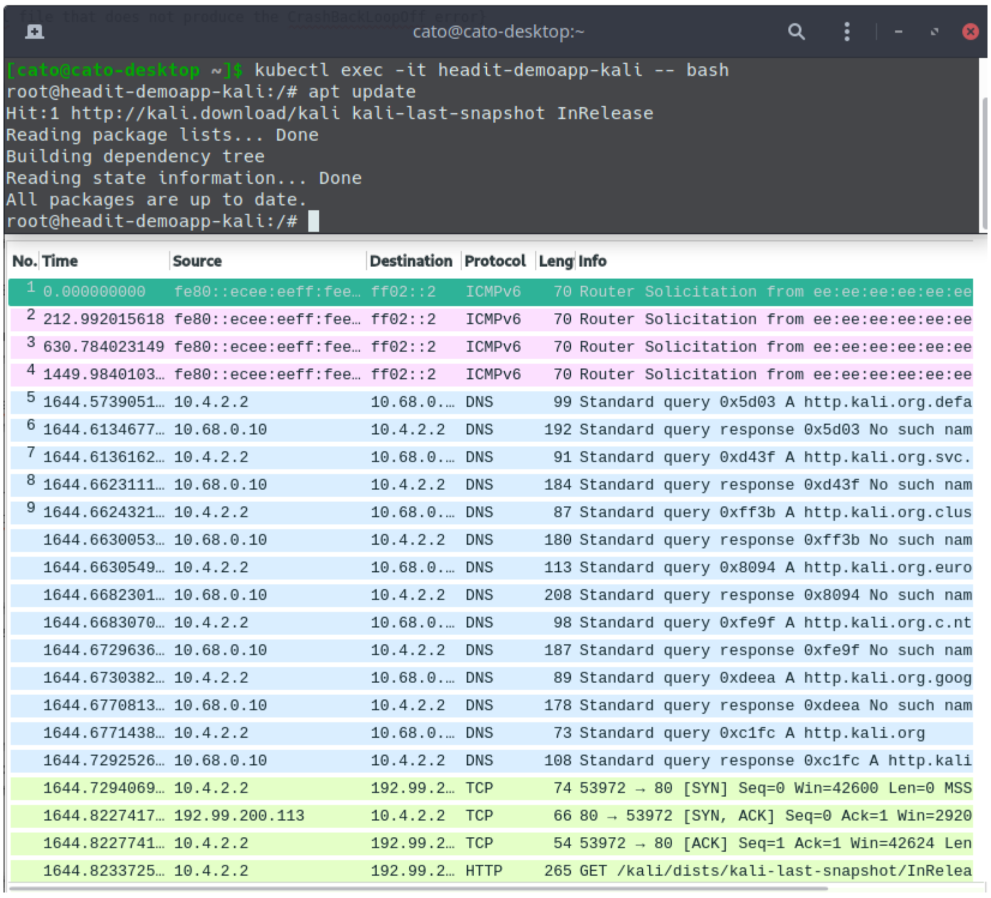

Capturing from -
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No. Time          Source               Destination Protocol Leng Info
  1 0.000000000   fe80::ecee:eeff:fee… ff02::2     ICMPv6     70 Router Solicitation from ee:ee:ee:ee:ee:ee
  2 212.992015618 fe80::ecee:eeff:fee… ff02::2     ICMPv6     70 Router Solicitation from ee:ee:ee:ee:ee:ee

▶ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth0, id 0
▼ Ethernet II, Src: ee:ee:ee:ee:ee:ee (ee:ee:ee:ee:ee:ee), Dst: IPv6mcast_02 (33:33:00:00:00:02)
  ▶ Destination: IPv6mcast_02 (33:33:00:00:00:02)
  ▶ Source: ee:ee:ee:ee:ee:ee (ee:ee:ee:ee:ee:ee)
    Type: IPv6 (0x86dd)

0000  33 33 00 00 00 02 ee ee  ee ee ee ee 86 dd 60 00   33······ ······`·
0010  00 00 00 10 3a ff fe 80  00 00 00 00 00 00 ec ee   ····:··· ········
0020  ee ff fe ee ee ee ff 02  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 02 85 00  e5 94 00 00 00 00 01 01   ········ ········
0040  ee ee ee ee ee ee                                  ······

Destination Hardware Address (eth.dst), 6 bytes    Packets: 2 · Displayed: 2 (100.0%)    Profile: Default
```

Figure 3.10: Wireshark instance recording network traffic from the kali container for about 5 minutes

The kali pod could not see any of the traffic destined for either customer A or B. At this point we had assumed that at least the API request sent to customer A should have been collected/sniffed. The Kali pod has two network policies opening up port 80 to and from customer A's web server shown in kali's configuration file located in **B.1**.

To investigate if the network sniffing was working locally inside the kali pod itself, we did a apt-get update request as depicted in Figure 3.11.



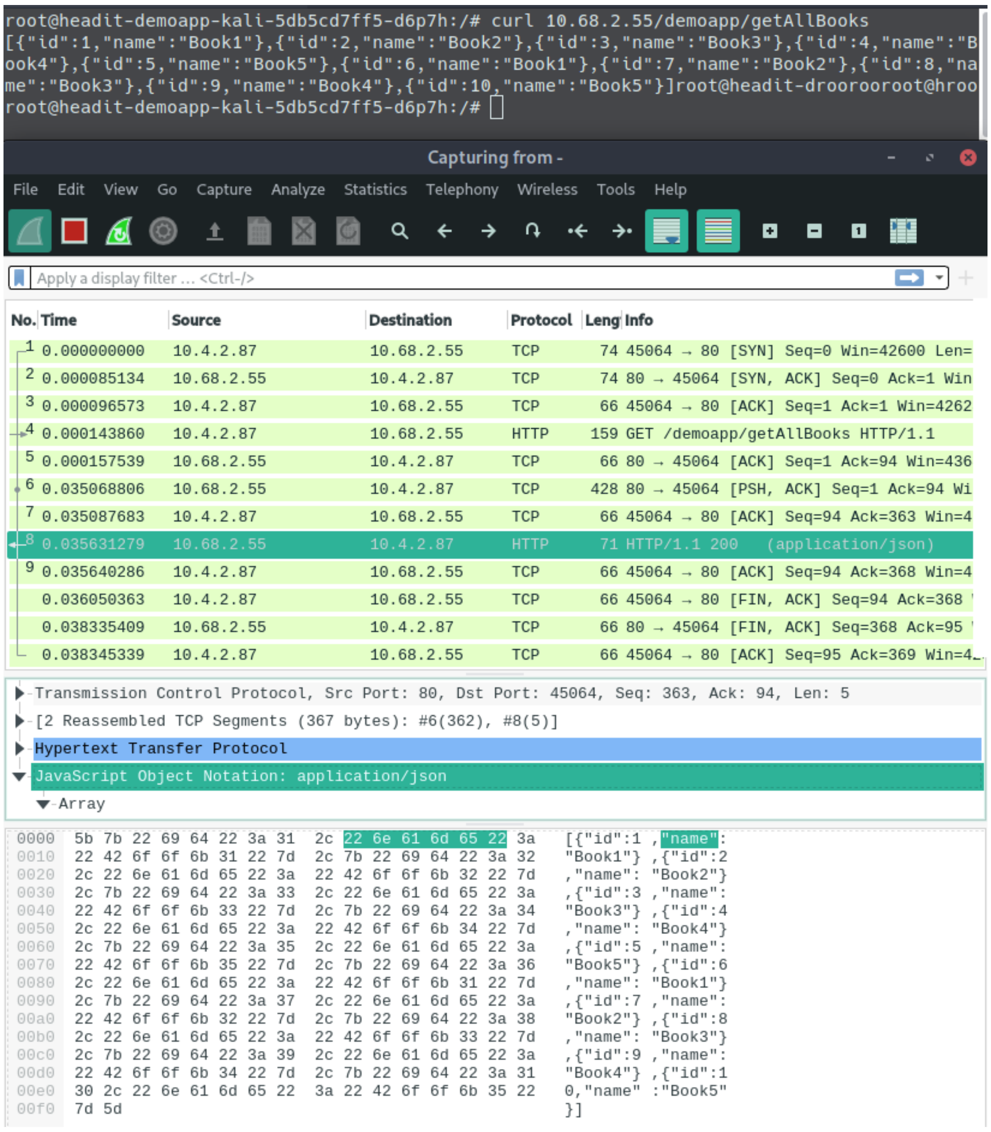
```
[cato@cato-desktop ~]$ kubectl exec -it headit-demoapp-kali -- bash
root@headit-demoapp-kali:/# apt update
Hit:1 http://kali.download/kali kali-last-snapshot InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
root@headit-demoapp-kali:/#
```

```
No. Time             Source               Destination  Protocol  Leng Info
 1  0.000000000      fe80::ecee:eeff:fee… ff02::2      ICMPv6      70 Router Solicitation from ee:ee:ee:ee:ee:ee
 2  212.992015618    fe80::ecee:eeff:fee… ff02::2      ICMPv6      70 Router Solicitation from ee:ee:ee:ee:ee:ee
 3  630.784023149    fe80::ecee:eeff:fee… ff02::2      ICMPv6      70 Router Solicitation from ee:ee:ee:ee:ee:ee
 4  1449.9840103…    fe80::ecee:eeff:fee… ff02::2      ICMPv6      70 Router Solicitation from ee:ee:ee:ee:ee:ee
 5  1644.5739051…    10.4.2.2             10.68.0.…    DNS         99 Standard query 0x5d03 A http.kali.org.defa
 6  1644.6134677…    10.68.0.10           10.4.2.2     DNS        192 Standard query response 0x5d03 No such nam
 7  1644.6136162…    10.4.2.2             10.68.0.…    DNS         91 Standard query 0xd43f A http.kali.org.svc.
 8  1644.6623111…    10.68.0.10           10.4.2.2     DNS        184 Standard query response 0xd43f No such nam
 9  1644.6624321…    10.4.2.2             10.68.0.…    DNS         87 Standard query 0xff3b A http.kali.org.clus
    1644.6630053…    10.68.0.10           10.4.2.2     DNS        180 Standard query response 0xff3b No such nam
    1644.6630549…    10.4.2.2             10.68.0.…    DNS        113 Standard query 0x8094 A http.kali.org.euro
    1644.6682301…    10.68.0.10           10.4.2.2     DNS        208 Standard query response 0x8094 No such nam
    1644.6683070…    10.4.2.2             10.68.0.…    DNS         98 Standard query 0xfe9f A http.kali.org.c.nt
    1644.6729636…    10.68.0.10           10.4.2.2     DNS        187 Standard query response 0xfe9f No such nam
    1644.6730382…    10.4.2.2             10.68.0.…    DNS         89 Standard query 0xdeea A http.kali.org.goog
    1644.6770813…    10.68.0.10           10.4.2.2     DNS        178 Standard query response 0xdeea No such nam
    1644.6771438…    10.4.2.2             10.68.0.…    DNS         73 Standard query 0xc1fc A http.kali.org
    1644.7292526…    10.68.0.10           10.4.2.2     DNS        108 Standard query response 0xc1fc A http.kali
    1644.7294069…    10.4.2.2             192.99.2…    TCP         74 53972 → 80 [SYN] Seq=0 Win=42600 Len=0 MSS
    1644.8227417…    192.99.200.113       10.4.2.2     TCP         66 80 → 53972 [SYN, ACK] Seq=0 Ack=1 Win=2920
    1644.8227741…    10.4.2.2             192.99.2…    TCP         54 53972 → 80 [ACK] Seq=1 Ack=1 Win=42624 Len
    1644.8233725…    10.4.2.2             192.99.2…    HTTP       265 GET /kali/dists/kali-last-snapshot/InRelea
```

Figure 3.11: Wireshark output of traffic from kali after running apt-get update

This confirms that our networking sniffing does collect local traffic inside kali as it should, but we could not see any other internal traffic within the cluster.

Since it was unexpected to not see the API request sent to customer A due to the network policies in kali's configuration file , we decided to check if kali could speak directly to customer A's web server inside the cluster. A new API request was sent, only this time from the kali pod to the internal cluster ip of "*headit-demoapp-web*" as shown in Figure 3.12 below.



```
root@headit-demoapp-kali-5db5cd7ff5-d6p7h:/# curl 10.68.2.55/demoapp/getAllBooks
[{"id":1,"name":"Book1"},{"id":2,"name":"Book2"},{"id":3,"name":"Book3"},{"id":4,"name":"B
ook4"},{"id":5,"name":"Book5"},{"id":6,"name":"Book1"},{"id":7,"name":"Book2"},{"id":8,"na
me":"Book3"},{"id":9,"name":"Book4"},{"id":10,"name":"Book5"}]root@headit-drooroooroot@hroo
root@headit-demoapp-kali-5db5cd7ff5-d6p7h:/#
```

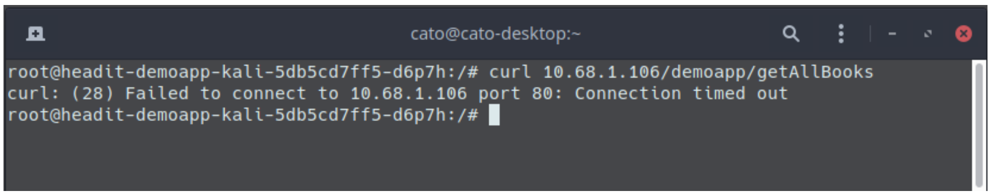
Capturing from -
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No. Time         Source       Destination  Protocol Leng Info
1   0.000000000  10.4.2.87    10.68.2.55   TCP       74 45064 → 80 [SYN] Seq=0 Win=42600 Len=
2   0.000085134  10.68.2.55   10.4.2.87    TCP       74 80 → 45064 [SYN, ACK] Seq=0 Ack=1 Win
3   0.000096573  10.4.2.87    10.68.2.55   TCP       66 45064 → 80 [ACK] Seq=1 Ack=1 Win=4262
4   0.000143860  10.4.2.87    10.68.2.55   HTTP     159 GET /demoapp/getAllBooks HTTP/1.1
5   0.000157539  10.68.2.55   10.4.2.87    TCP       66 80 → 45064 [ACK] Seq=1 Ack=94 Win=436
6   0.035068806  10.68.2.55   10.4.2.87    TCP      428 80 → 45064 [PSH, ACK] Seq=1 Ack=94 Wi
7   0.035087683  10.4.2.87    10.68.2.55   TCP       66 45064 → 80 [ACK] Seq=94 Ack=363 Win=4
8   0.035631279  10.68.2.55   10.4.2.87    HTTP      71 HTTP/1.1 200    (application/json)
9   0.035640286  10.4.2.87    10.68.2.55   TCP       66 45064 → 80 [ACK] Seq=94 Ack=368 Win=4
    0.036050363  10.4.2.87    10.68.2.55   TCP       66 45064 → 80 [FIN, ACK] Seq=94 Ack=368
    0.038335409  10.68.2.55   10.4.2.87    TCP       66 80 → 45064 [FIN, ACK] Seq=368 Ack=95
    0.038345339  10.4.2.87    10.68.2.55   TCP       66 45064 → 80 [ACK] Seq=95 Ack=369 Win=4…

▶ Transmission Control Protocol, Src Port: 80, Dst Port: 45064, Seq: 363, Ack: 94, Len: 5
▶ [2 Reassembled TCP Segments (367 bytes): #6(362), #8(5)]
▶ Hypertext Transfer Protocol
▼ JavaScript Object Notation: application/json
  ▼ Array

0000  5b 7b 22 69 64 22 3a 31  2c 22 6e 61 6d 65 22 3a   [{"id":1 ,"name":
0010  22 42 6f 6f 6b 31 22 7d  2c 7b 22 69 64 22 3a 32   "Book1"} ,{"id":2
0020  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 32 22 7d   ,"name": "Book2"}
0030  2c 7b 22 69 64 22 3a 33  2c 22 6e 61 6d 65 22 3a   ,{"id":3 ,"name":
0040  22 42 6f 6f 6b 33 22 7d  2c 7b 22 69 64 22 3a 34   "Book3"} ,{"id":4
0050  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 34 22 7d   ,"name": "Book4"}
0060  2c 7b 22 69 64 22 3a 35  2c 22 6e 61 6d 65 22 3a   ,{"id":5 ,"name":
0070  22 42 6f 6f 6b 35 22 7d  2c 7b 22 69 64 22 3a 36   "Book5"} ,{"id":6
0080  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 31 22 7d   ,"name": "Book1"}
0090  2c 7b 22 69 64 22 3a 37  2c 22 6e 61 6d 65 22 3a   ,{"id":7 ,"name":
00a0  22 42 6f 6f 6b 32 22 7d  2c 7b 22 69 64 22 3a 38   "Book2"} ,{"id":8
00b0  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 33 22 7d   ,"name": "Book3"}
00c0  2c 7b 22 69 64 22 3a 39  2c 22 6e 61 6d 65 22 3a   ,{"id":9 ,"name":
00d0  22 42 6f 6f 6b 34 22 7d  2c 7b 22 69 64 22 3a 31   "Book4"} ,{"id":1
00e0  30 2c 22 6e 61 6d 65 22  3a 22 42 6f 6f 6b 35 22   0,"name" :"Book5"
00f0  7d 5d                                              }]
```

Figure 3.12: API request sent internally from headit-demoapp-kali to the internal cluster ip for headit-demoapp-web: 10.68.2.55, below is the wireshark result of captured network traffic.

The internal API request worked as expected shown in Figure 3.12, meaning that the network policies defined in kali's configuration file works as expected.

Now we needed to know if our kali container could talk to the webserver inside customer B's environment. To find out we made an identical API request to the cluster ip belonging to headit-demoapp-web-b. The API request can be seen in Figure 3.13. The API request to customer B's web server did not come through confirming that the network policies are indeed blocking them.



```
root@headit-demoapp-kali-5db5cd7ff5-d6p7h:/# curl 10.68.1.106/demoapp/getAllBooks
curl: (28) Failed to connect to 10.68.1.106 port 80: Connection timed out
root@headit-demoapp-kali-5db5cd7ff5-d6p7h:/#
```

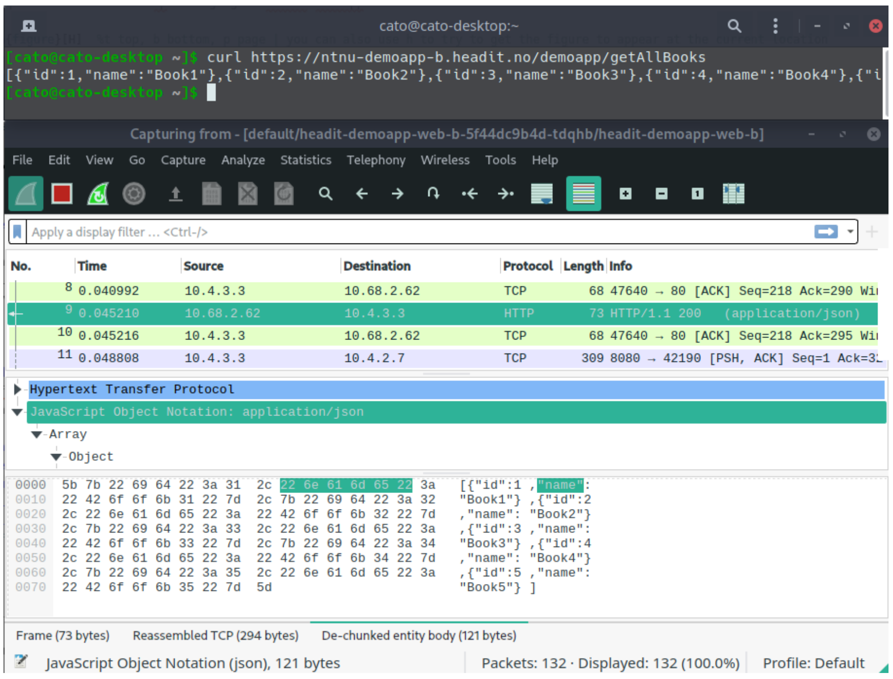
Figure 3.13: API request internally from headit-demoapp-kali to the internal cluster ip for headit-demoapp-web-b: 10.68.1.106

3.3.4 Ksniff Testing

To confirm the testing done with Tshark on the Kali container, we found a Kubectl plugin later in the project called ksniff. This is networking sniffing tool as well that dumps network traffic from any running pod. It attaches itself to the target pod and uses a binary called tcpdump and then pipes the result to a local wireshark instance. This functionality is basically the same as our solution but since it works a little different we felt it was a good opportunity to confirm our findings from subsection 3.3.3.
Installing Ksniff is done via a plugin manager for Kubectl called krew [31]. First we install Krew and from there install Ksniff. We started two instances of Ksniff with one attached to headit-demoapp-web, and the other to headit-demoapp-web-b. To confirm our previous results, we made an API request to customer B's web-server to see if the traffic was visible to customer A this can be seen from Figure 3.14.
Figure 3.14 confirms that customer B's web-server can see it's own traffic just like in our previous testing with Wireshark in Figure 3.11.
Figure 3.14 and Figure 3.15 were taken simultaneously, during that time a API request was sent to customer B in Figure 3.14 but no traffic leaked over to customer A's webserver. This result is identical to the previous testing with tshark in 3.3.3.

3.3.5 Nmap

is a very popular network mapping tool and port scanner, it is a good tool to make a map of a network infrastructure and the computers hosts residing in that infrastructure. In this case we did not expect much information from it, we know that only port 80 is used in the cluster and the infrastructure is very restricted in it's configuration compared to a physical workplace lan for example.



```
cato@cato-desktop:~
[cato@cato-desktop ~]$ curl https://ntnu-demoapp-b.headit.no/demoapp/getAllBooks
[{"id":1,"name":"Book1"},{"id":2,"name":"Book2"},{"id":3,"name":"Book3"},{"id":4,"name":"Book4"},{"i
[cato@cato-desktop ~]$
```

```
Capturing from - [default/headit-demoapp-web-b-5f44dc9b4d-tdqhb/headit-demoapp-web-b]

No.  Time      Source      Destination  Protocol  Length Info
   8 0.040992  10.4.3.3    10.68.2.62   TCP       68 47640 → 80 [ACK] Seq=218 Ack=290 Wi
   9 0.045210  10.68.2.62  10.4.3.3     HTTP      73 HTTP/1.1 200   (application/json)
  10 0.045216  10.4.3.3    10.68.2.62   TCP       68 47640 → 80 [ACK] Seq=218 Ack=295 Wi
  11 0.048808  10.4.3.3    10.4.2.7     TCP       309 8080 → 42190 [PSH, ACK] Seq=1 Ack=3

▶ Hypertext Transfer Protocol
▼ JavaScript Object Notation: application/json
  ▼ Array
    ▼ Object

0000  5b 7b 22 69 64 22 3a 31  2c 22 6e 61 6d 65 22 3a   [{"id":1 ,"name":
0010  22 42 6f 6f 6b 31 22 7d  2c 7b 22 69 64 22 3a 32   "Book1"} ,{"id":2
0020  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 32 22 7d   ,"name": "Book2"}
0030  2c 7b 22 69 64 22 3a 33  2c 22 6e 61 6d 65 22 3a   ,{"id":3 ,"name":
0040  22 42 6f 6f 6b 33 22 7d  2c 7b 22 69 64 22 3a 34   "Book3"} ,{"id":4
0050  2c 22 6e 61 6d 65 22 3a  22 42 6f 6f 6b 34 22 7d   ,"name": "Book4"}
0060  2c 7b 22 69 64 22 3a 35  2c 22 6e 61 6d 65 22 3a   ,{"id":5 ,"name":
0070  22 42 6f 6f 6b 35 22 7d  5d                        "Book5"} ]

Frame (73 bytes)   Reassembled TCP (294 bytes)   De-chunked entity body (121 bytes)
JavaScript Object Notation (json), 121 bytes   Packets: 132 · Displayed: 132 (100.0%)   Profile: Default
```

Figure 3.14: API request to customer B's headit-demoapp-web-b above wireshark using ksniff on headit-demoapp-web-b's network traffic

A text file was set up as a list with the cluster IP's for every pod in the cluster, this was to limit the scan to both save time and have a much smaller report to attach to this thesis. We did a full scan of the entire 10.68.0.0/16 subnet but the result of that scan returned identical results. The list of known IP's was named "*list.txt*" and contained the following:

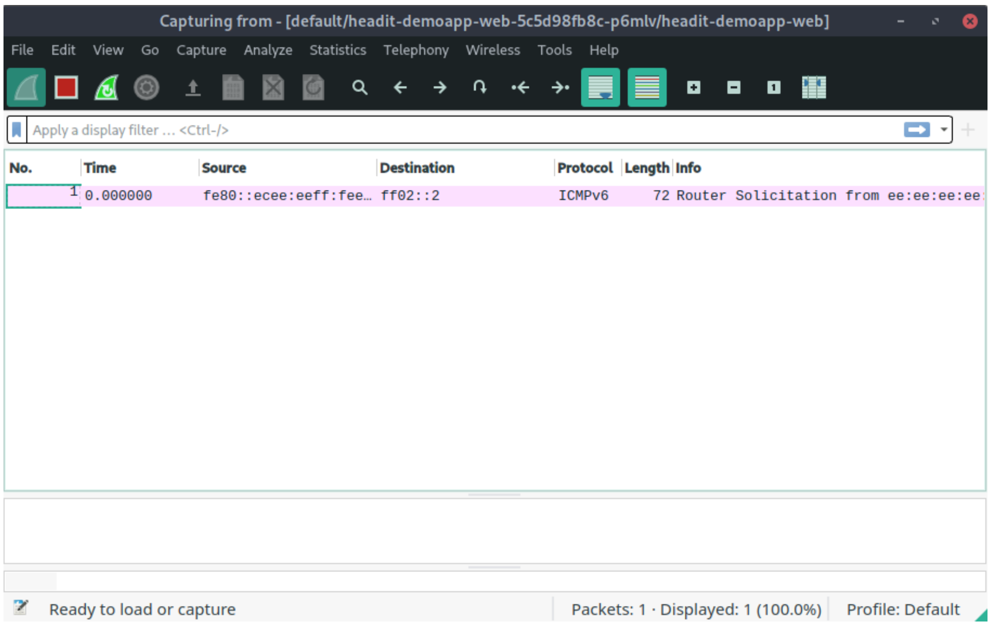
```
## list.txt
10.68.12.122  ## headit-demoapp-svc
10.68.2.55    ## headit-demoapp-web
10.68.2.62    ## headit-demoapp-svc-b
10.68.7.69    ## headit-demoapp-web-b
10.68.0.1     ## kubernetes.default.svc.cluster.local
```

The scan was started from the kali pod with this command:

```
nmap -Pn -A -sT -iL list.txt -oX nmap.xml
```

The resulting scan can be found in Appendix C.1. The only noteworthy thing about this scan is the fact that Network Mapper [7] (NMAP) was able to resolve the hostnames for all the cluster IP's.

It can look like this should not be possible due to the "*deny_all.yaml*" **A.1** network policy, but the reason most likely are due to the "*allow-egress-to-dns.yaml*" **A.5** network policy that applies to all pods in the cluster.


Capturing from - [default/headit-demoapp-web-5c5d98fb8c-p6mlv/headit-demoapp-web]
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source             Destination  Protocol Length Info
   1 0.000000  fe80::ecee:eeff:fee… ff02::2    ICMPv6       72 Router Solicitation from ee:ee:ee:ee
```

Ready to load or capture | Packets: 1 · Displayed: 1 (100.0%) | Profile: Default

Figure 3.15: Wireshark using ksniff on headit-demoapp-web network traffic running simultaneously as ksniff in Figure 3.14.

3.4 Results

After all the testing mentioned in 3.3.3, 3.3.4 and 3.3.5 we can conclude that network traffic behaves differently in GKE in comparison to a traditional network. These differences were noticed throughout the entire project and are important to understand in conjunction with our results. An overview of how GKE networks work can be read about here [32]. *It is important to note that Headit uses network policy's and most likely other configurations that are not default behaviour in GKE.*

In a traditional computer network, computers are separated with subnets, vlans, routes and firewalls via routers or physical switches [33]. Since GKE runs in the cloud and many pods can share the same hardware, there is a different approach with virtual switches that do the exact same job but only in software.

In a traditional network it can be pretty easy to spoof/pretend to be a different computer by just telling the router that you are someone else with a technique known as ARP spoofing [34]. We made one such attempt but ARP messages are just not accepted within the configuration of GKE Headit uses. Therefore without knowledge of a major exploit in GKE or kubernetes itself, using ARP spoofing is not an option unlike in most traditional network configurations.

Sending a ping [35] to check if a computer is online is also something trivial in a traditional computer network, but in the GKE configuration Headit uses, pinging is not allowed even when the pods are allowed to communicate over different protocols. Therefore using tools like OpenVAS [36] and other vulnerability scanners are next to impossible in this case since they all rely on ping and ARP for host detection. We briefly tried OpenVAS but without the ability to ping OpenVAS just assumed the pods were offline and stopped any further tests.

The way GKE handles network traffic looks to be watertight, Headit uses network policies as recommended by google and we were not able to see that they were not configured right or not working as intended. All the testing mentioned in 3.3.3, 3.3.4 and 3.3.5 were done multiple times with variation in what network policies were active and how many of them to see if traffic would leak. Only removing essentially all policies would open up traffic from customer A to customer B, therefore we are confident that Headit can feel safe about how they use and create network polices in GKE as long as their production workloads reflect how the provided demo application is configured.

Chapter 4

Improvements to Headits current system

As mentioned earlier, the second part of this project is to investigate how to improve Headit's current solution for offering better quality products to their customers.

In research working with Kubernetes, we have found some tools/services that we feel can make Kubernetes easier to work with and easier to manage. Table 4.1 lists all the recommended tools/services. This chapter will go in depth on each of them we find to be useful, especially in line with the wishes Headit have for continued operation of their Kubernetes systems.

Table 4.1: Tools/Services that might be able to improve Headit's current system.

Tool name	Description
Goldpinger	Connection visualizing tool, with options to create automated alerts for ocurring errors
Kube-applier	Auto-updater for clusters, connected to a Git-repository that applies updates at set intervals given there are updates available.
Kubespy	Real time resource and change monitoring tool for Kubernetes operating in the terminal.
Kubeval	YAML and JSON validator for any Kubernetes version.
Google Anthos	Hybrid cloud software allowing for combining cloud software with on-premises datacenters.

4.1 Goldpinger [37]

During times where everything is moving to clouds, and off site services debugging issues with services interacting could be harder than using on-site solutions. This is where Goldpinger comes into play. It is a Debugging tool for Kubernetes[37], and originally made and used by Bloomberg, before they put the project out under Open Source in December of 2018.

The tool allows for testing and displaying connectivity between nodes in a Kubernetes cluster. When it is set up within Kubernetes, it will ask for all the pods within the system, and graph out every single one and its connections into a graphical user interface for easy visibility of the system at work. This applies to a small cluster shown in Figure 4.1 and big cluster shown in Figure 4.2.

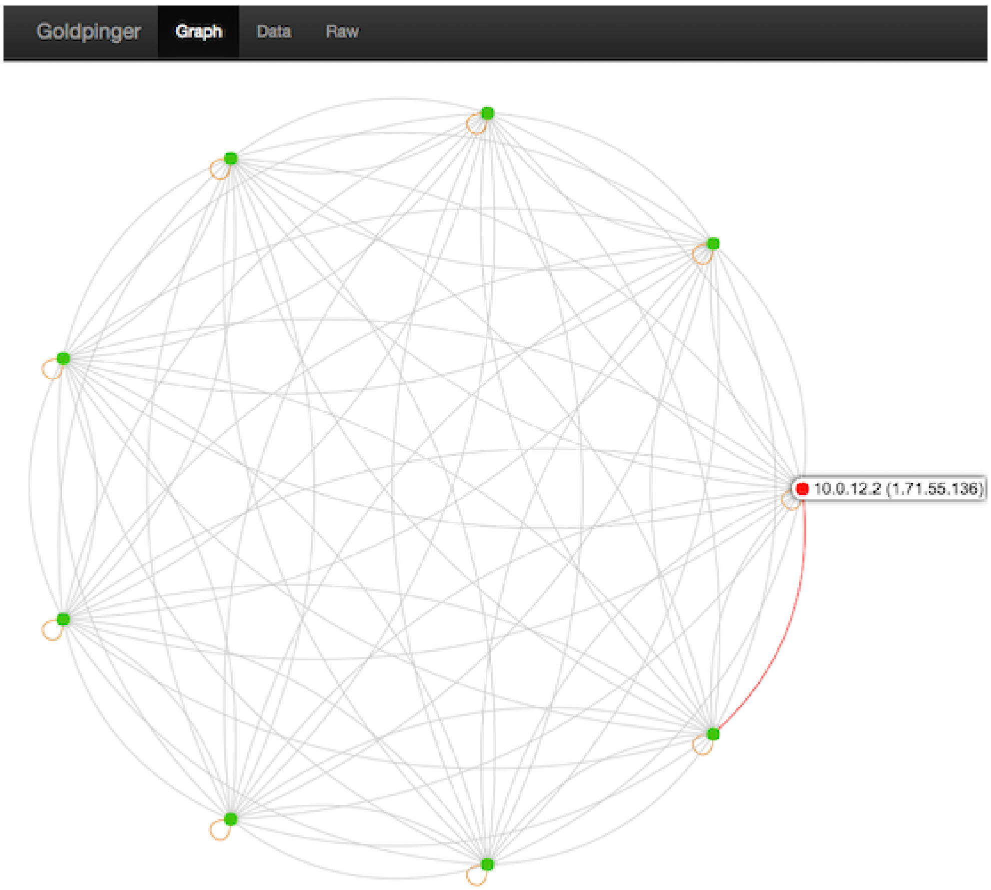


Figure 4.1: Example of Goldpinger added into a small Kubernetes cluster, and a display of the network connections of Pod in Red with connections to other pods as red lines. [37]

Goldpinger scales with any number of pods. If new pods are added or removed from the Kubernetes cluster, all it needs is a refresh in the browser and the up-

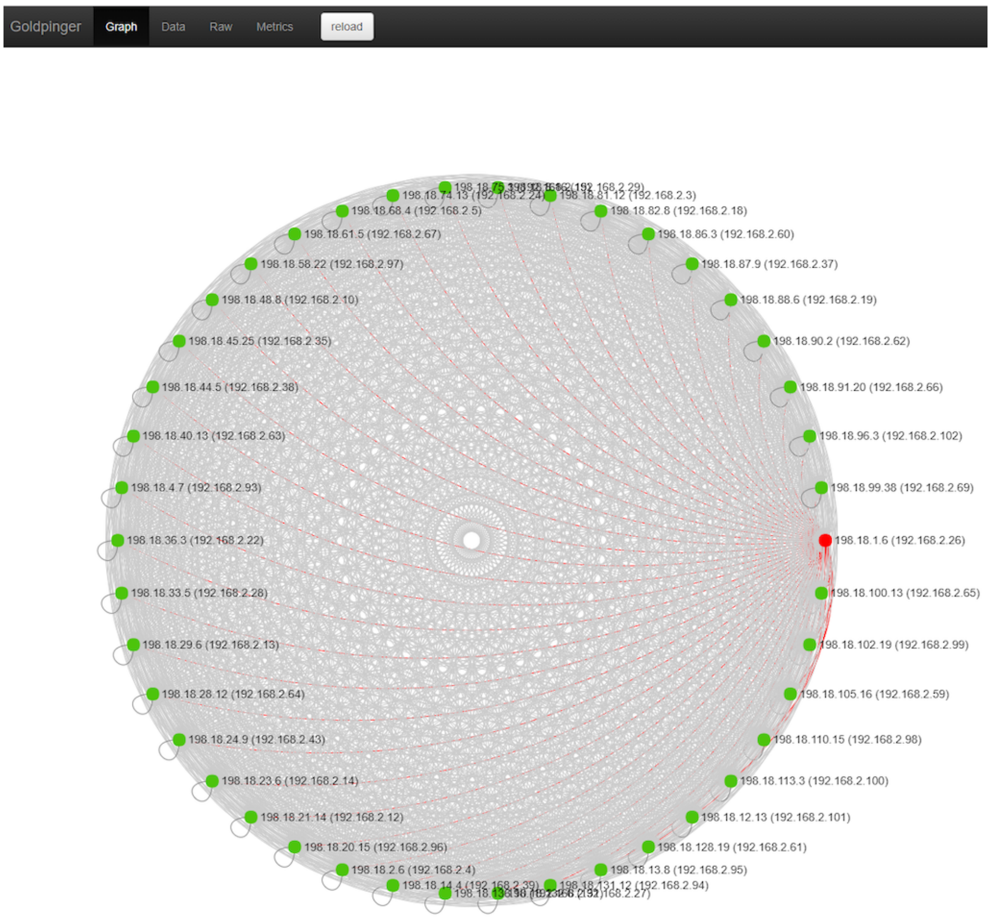


Figure 4.2: Example of Goldpinger added into a Kubernetes cluster, and a display of the network connections of Pod in Red with every other pod as a red line [37]

dated information will show. This can be a useful tool to figure out where issues are if any were to occur during every day operation. As a debugging tool being able to see every connetion, and interruption in connection between applications relying on communications between each other. While this tool is unable to automate correcting issues, it is a valueable tool for ensuring the lowest amount of downtime possible.

For Headit we would recommend the use of this especially for larger clusters administered for clients. Goldpinger also has some other integrated solutions that can be used from the same pod, such as Grafana and Prometheus.

Grafana[38] can be used to show different amounts of data related to the Kubernetes clusters and individual pods. You can also go into detail of the datasets if you need to see anomalies or just extra detailed look into current use of system resources. A lot of this can also be seen through the Google Cloud Kubernetes

Engine, but to a less organized degree.

Prometheus[39] is a powerful monitoring and alerting system. It gathers data on what it is set to monitor and can either record the operation, or generate alerts. It is designed to be a system that you use for quickly diagnosing problems during an outage. This system is already used by a few of Headits clients, and we would recommend that it is implemented and used where possible.

4.2 Kube-applier [40]

Managing every deployment or many groups of pods can be time consuming and might cause confusion over time, especially when the system needs a large system wide update.

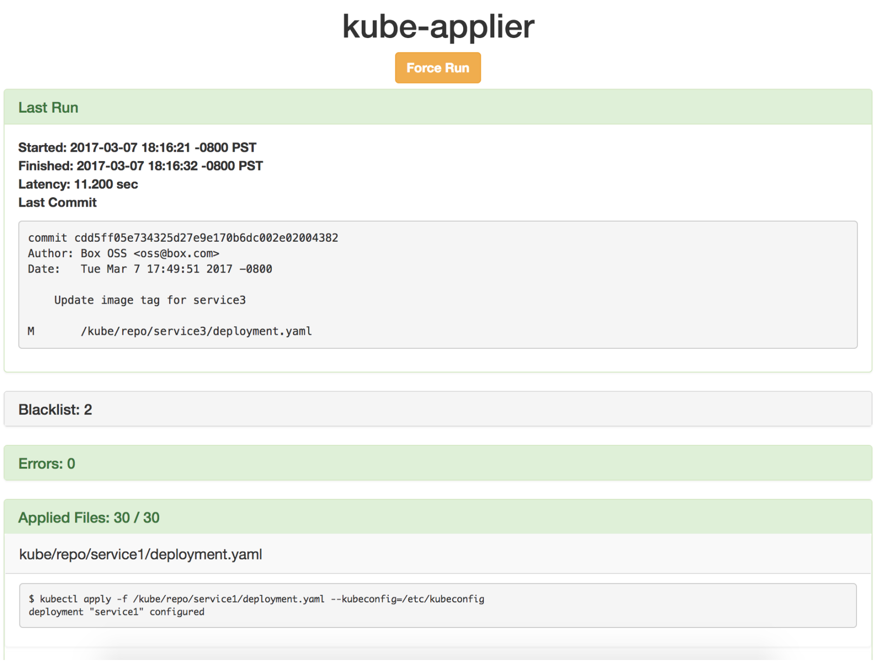
Kube-applier is a service for automatically applying updates to Kubernetes clusters from Git repositories specified by the administrator[40]. The service is designed to be run within a cluster with the sole intention to perform the application command for all JavaScript Object Notation [6] (JSON) and YAML files within the admin specified repository.

These JSON and YAML files are what specifies how a pod is supposed to be set up. For managing multiple instances of the same base solution Kube-applier can be worked on and deployed from outside a cluster. From here the code can be tested within git, before it is later what will be replacing the old approved configurations with new and improved versions. In addition to this Kube-applier allows for logging things within a git repo, as well as making logs about each attempt at running the application command.

Kube-applier also has a user interface telling of the status of the most recent run. The information that can be seen in this User interface is generally

- Run type
- Start and end times
- Latency
- Most recent commit to the dedicated repository
- Whitelisted files
- Blacklisted files
- Errors
- List of successfully applied files

Kube-applier allows for a deeper focus on writing code and improving what is currently in production over needing to focus on deployment. This aligns with Headit's wishes to be able to improve quality and spend less time focusing on debugging and updating.


kube-applier

Last Run

Started: 2017-03-07 18:16:21 -0800 PST
Finished: 2017-03-07 18:16:32 -0800 PST
Latency: 11.200 sec
Last Commit

```
commit cdd5ff05e734325d27e9e170b6dc002e02004382
Author: Box OSS <oss@box.com>
Date:   Tue Mar 7 17:49:51 2017 -0800

    Update image tag for service3

M       /kube/repo/service3/deployment.yaml
```

Blacklist: 2

Errors: 0

Applied Files: 30 / 30

kube/repo/service1/deployment.yaml

```
$ kubectl apply -f /kube/repo/service1/deployment.yaml --kubeconfig=/etc/kubeconfig
deployment "service1" configured
```

Figure 4.3: Example of Kube-applier UI, with successful last run given 30 applied files[40]

4.3 Kubespy [41]

Kubespy is a real time resource monitoring tool, When running containers, it is possible to lose some insight into what is going on inside when making changes or updating the system in different ways. This is where Kubespy comes in. It is a light tool running in the command line from a company called Pulumi. The program is Open Source, with new functionality being worked on over time.

When using Kubespy there are three main commands:

- **kubespy status** It monitors changes made to the .status field and displays changes as a syntax-highlited JSON difference.
- **kubespy changes** It monitors changes made to a user defined field and displays the changes as a syntax-highlited JSON difference.
- **kubespy trace** It provides a real-time summary of changes. These changes are related to more complex cluster level commands such as deployments.

For now these are the commands provided by Kubespy, and there are more commands planned for future releases.

4.4 Kubeval [42]

The simplest tool we have looked into and want to recommend for use is a YAML and JSON validator. Early in our assignment we got recommended a website for viewing network policies [43], and the details of them. While this is great for working and validating the functionality of the policies written for different pods, this does not help out if we have a misconfigured pod from the beginning.

In order to test YAML or JSON files in Kubernetes, it is simplest to test a deployment and correct errors if detected. This is not an efficient way, it also requires that Headit has a testing environment that they use for test deployments before updating the services in live production. Testing the files beforehand can be more efficient than to test into production.

Kubeval operates with schemas generated from a specification for Kubernetes OpenAPI. Because of this it allows for testing and validation of multiple versions of Kubernetes. When running the command to test you can select the Kubernetes version that the configuration file will be deployed in, and it will test using the schemas for that version.

4.5 Anthos

Anthos is a service within the Google Cloud ecosystem that allows for applications to be built, deployed and managed from anywhere. This software integrates with Kubernetes and with the Anthos GKE on-prem solution. This can be run within Google Cloud like the normal Kubernetes engine, as well as having the option to use an on-site server.

Anthos is integrated in Kubernetes as a manager for the service. Kubernetes is set up in two main parts: a control plane, and the node components. Within Google Cloud, the control plane is hosted within the Cloud service, and the Kubernetes API service is the only component accessible to customers. GKE manages the node komponents in the customers project using instances in Compute Engine. With GKA on-prem, all components are hosted in the customers on-prem virtualization environment.

With K8s installed and running, the system administrator have access to a common orchestration layer that manages application deployment, configuration, upgrades and scaling.

4.5.1 Anthos GKE on-prem

Anthos GKE on-prem (GKE on-prem) is a hybrid cloud software that beings the GKE to local server solutions owned by the operator. The engine adds the ability to manage, create, and upgrade clusters running within Kubernetes in the local server environment. It is also able to manage a varied amount of clusters independently of its host source. It allows for easy conversion of locally hosted Windows apps to be translated over to a modernized infrastructure system like Kubernetes, while also allowing to keep a local host within the business offices.

4.5.2 Binary Authorization

When running local servers or data centers, security controls are quite important, and can generally be used to limit communications between machines in the same system. System administrators also need ways to secure quality of hosted machines, and prevent downtime because of a bad push onto the environment. Anthos has similar functionality available in it, and this is called Binary Authorization.

The main uses for Binary Authorization are to do the following:

- Enforce standardized container release practices
- Put proactive security measures in place
- Native GCP integration

The methods that the system uses for handling these things are quite similar to the services mentioned before. To enforce the standardized container release practices they use methods like that of Kubeval and Kube-applier, with the same ability to use repositories as Kube-applier.

When working towards proactive sevurity measures within Anthos, the configuration for these are quite similar to what you find when working with network policies in Kubernetes. The proactive measures can be managed and edited from repositories, and Anthos allows for a deeper level of configuration than what can be found within Kubernetes itself.

Through the Google Cloud Kubernetes Engine a system administrator can manage his/her clusters and pods, Anthos has a similar system expanding in features and options for hosts a system administrator wants to manage. Through this combining a Google Cloud environment with a server on site would work and operate together, with it all managed within the same administrative software solution. Through the Anthos GKE you get the ability to sync your multiple systems to make sure it all uses the same configurations and stays up to date with the rapidly updated Kubernetes system. It also allows for seamless transfer of applications and data between cloud and on-site computing.

4.6 Our view on ways to improve workflow within using recommended services

As mentioned earlier the second goal of the thesis is to investigate better ways to maintain and improve the security for Headits offerings to their clients. At the same time finding better ways to make working with the system more efficient.

Through the fact that Headit is selling their services to their clients they will want to edit and change functionality over time in order to maintain their access to their system while also allowing for monitoring and debugging of possible errors on Headits side. For this adding Kube-applier to the workflow can be benefitial.

With Kube-applier a system administrator can limit what files are supposed to deploy with white- and blacklists. This can be combined with hosting multiple instances of Kube-applier to handle dedicatd files. Alternatively running multiple clusters with one instance of Kube-applier each would be easier to work with, and allow for more ability to share management access to clients who want more insight into the hosting of their systems.

In order to make these fixes easier to do adding Goldpinger is one efficient way of seeing where the issues are arising. Combine Goldpinger with some of the integrated connections it has to Prometheus for alerting about errors, and the sum of these could allow for easier localization of where an error arises and why.

In order to reduce the risk of errors ocurring, we would recommend running configurations through Kubeval before pushing changes to the Kubernetes cluster, or adding it as an automated test within a Git repository. Doing this could make it less likely for an error to occur with the pod, and instead if any problems were to arise the issue would be with the application running inside the pod.

In addition, when adding new deployments checking that everything is going properly is something that will be done no matter what, so running a Kubespy instance through the terminal in order to see what is going on and if any errors are happening in real time correct them before other errors occur because of this. Anthos might be a good way to enhance the offerings to the customers. Using Anthos, operations could still work like they are today, but with increased levels of logging and quality assurance on pods and application before entering production. Properly setup environments could also automate a lot of the debugging that currently take time and cause annoyances because of faulty configured policies and pods.

Depending on client workloads Anthos could also save money. On the 6th of June 2020 pricing for worker nodes and cluster management in GKE will get an extra fee added of $0.10 per hour of time a cluster is powered on. Reducing the amount of background hours and running services needing 24 hour operation out of the cloud could reduce the amount this new fee will have an effect over time. Given that only one cluster needs to run 24/7 for a client, per year the hourly fee sums up to around $873.60. This cost applies to a per-cluster basis. Operating with multiple clusters for clients could make this more expensive depending on how long some pods need to be active.

Chapter 5

Closing Remarks

To close off our thesis we will now discuss our results and suggestions for future work on this topic.

5.1 Discussion

Early in the project we thought the security investigation would be more comprehensive with more tools and testing. But upon starting the investigation we realized that the demo application is limited in its potential attack vectors and possible scope for testing. Nonetheless we were able to perform enough tests to conclude that the provided configuration does not leak network packets, and does not allow for communication between pods that are not specifically allowed in the network policies. We were hoping to find issues and provide solutions for fixing them, but instead we confirmed that the network policies works as intended.

5.2 Conclusion

In conclusion to the security investigation we are confident that the network policies in the provided configuration block unwanted traffic, and also blocks traffic between customers which was the main concern for Headit. We have only seen and tested the demo application, but if Headit uses the same logic when configuring network policies for their customers infrastructure, the same results should apply.

From our experience with Kubernetes it can get hard to setup a few things, and expecially when setting up multiple different types of Pods in the same deployment, and looking for errors that could occur. Thankfully there are a lot of solutions for this that makes it simpler to operate and debug a cluster. The mentioned solutions will not be the complete way to work faster and easier in Kubernetes, but it is a good step in the right direction.

5.3 Learning Outcome

Through our thesis we have learned in depth about cloud environments and working with Multi-Tenant solutions. We have learned more basics about working with cloud infrastructure and cloud technologies, as well as gone in depth about working within Google Cloud and how to operate and deploy services within it. As well as learned a lot about Kubernetes and the future of the service. Kubernetes is a widely used solution getting more and more adoption within cloud hosted systems each day that passes. We also got more experience withing with Docker, a technology we have been learning about in a limited amount within our studies, and that we now have gone more in depth in to understand Kubernetes in a better way.

We also got a better insight into software security and important metrics to be aware of when working with software. Combining this with important Cloud ideas we have gained quite important experience that will allow for better operation of infrastructure in possible future work.

5.4 Future work

For future work we would recommend looking more into what can be done if someone acquires a shell with administrator rights on a pod within the system. Our investigation was primarily focused on what can be seen between pods inside the same cluster. We didn't look at how deep you can get in the system architecture. Questions like weather or not a malicious user can get information from a database with a direct connection though a pod. Or if a user could get administrative access of the cluster in order to cause issues.

Bibliography

[1] W. Contributors, *Application programming interface*, https://en.wikipedia.org/wiki/Application_programming_interface, 2020.

[2] W. Contributors, *Common vulnerabilities and exposures*, https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures, 2020.

[3] W. Contributors, *Google cloud platform*, https://en.wikipedia.org/wiki/Google_Cloud_Platform, 2020.

[4] G. Cloud, *Google kubernetes engine*, https://cloud.google.com/kubernetes-engine#section-2, 2020.

[5] W. Contributors, *Hypertext transfer protocol*, https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, 2020.

[6] W. Contributors, *Json - javascript object notation*, https://en.wikipedia.org/wiki/JSON, 2020.

[7] W. Contributors, *Nmap*, https://en.wikipedia.org/wiki/Nmap, 2020.

[8] W. Contributors, *Yaml*, https://en.wikipedia.org/wiki/YAML, 2020.

[9] M. Boelen, *What are linux capabilities*, https://linux-audit.com/linux-capabilities-101/#what-are-linux-capabilities, 2020.

[10] T. K. Authors, *What is kubernetes?*, https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/, 2020.

[11] T. K. Authors, *Kubernetes master*, https://kubernetes.io/docs/concepts/#kubernetes-master/, 2020.

[12] T. K. Authors, *Nodes*, https://kubernetes.io/docs/concepts/architecture/nodes/, 2020.

[13] T. K. Authors, *Pods*, https://kubernetes.io/docs/concepts/workloads/pods/pod/, 2020.

[14] T. K. Authors, *Replicationcontroller*, https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/, 2020.

[15] T. K. Authors, *Virtual ips and service proxies*, https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-and-service-proxies, 2020.

[16] T. K. Authors, *Kubelet*, https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/, 2020.

[17] K. Team, *Overview of kubectl*, https://kubernetes.io/docs/reference/kubectl/overview/, 2020.

[18] T. K. Authors, *Deployments*, https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, 2020.

[19] W. Contributors, *Kali linux*, https://en.wikipedia.org/wiki/Kali_Linux, 2020.

[20] W. Contributors, *Wireshark*, https://en.wikipedia.org/wiki/Wireshark, 2020.

[21] W. Contributors, *Headless software*, https://en.wikipedia.org/wiki/Headless_software, 2020.

[22] W. Contributors, *Pipeline (unix)*, https://en.wikipedia.org/wiki/Pipeline_(Unix), 2020.

[23] G. eldadru, *Ksniff - kubectl plugin*, https://github.com/eldadru/ksniff, 2020.

[24] W. Contributors, *Reverse proxy*, https://en.wikipedia.org/wiki/Reverse_proxy, 2020.

[25] W. Contributors, *Virtual hosting*, https://en.wikipedia.org/wiki/Virtual_hosting, 2020.

[26] W. Contributors, *Http 404*, https://en.wikipedia.org/wiki/HTTP_404, 2020.

[27] Google, *Configuring network policies for applications*, https://cloud.google.com/kubernetes-engine/docs/tutorials/network-policy#step_2_restrict_incoming_traffic_to_pods, 2020.

[28] W. Contributors, *Domain name system*, https://en.wikipedia.org/wiki/Domain_Name_System, 2020.

[29] A. L. Team, *Capabilities - arch wiki*, https://wiki.archlinux.org/index.php/Capabilities, 2020.

[30] W. Org, *Dumpcap - the wireshark network analyzer*, https://wireshark.org/docs/man-pages/dumpcap.html, 2020.

[31] K. Team, *Extend kubectl with plugins*, https://kubernetes.io/docs/tasks/extend-kubectl/kubectl-plugins/, 2020.

[32] G. Cloud, *Network overview - gke*, https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview#pods, 2020.

[33] W. Contributors, *Network segmentation*, https://en.wikipedia.org/wiki/Network_segmentation, 2020.

[34] W. Contributors, *Arp spoofing*, https://en.wikipedia.org/wiki/ARP_spoofing, 2020.

[35] W. Contributors, *Ping (networking utility)*, https://en.wikipedia.org/wiki/Ping_(networking_utility), 2020.

[36] W. Contributors, *Openvas*, https://en.wikipedia.org/wiki/OpenVAS, 2020.

[37] B. Github, *Goldpinger - debugging tool for kubernetes which tests and displays connectivity between nodes in the cluster*, https://github.com/bloomberg/goldpinger.

[38] G. Labs, *Grafana*, https://grafana.com/oss/grafana/, 2020.

[39] P. Authors, *What is prometheus?*, https://prometheus.io/docs/introduction/overview/, 2020.

[40] B. Github, *Kube-applier enables automated deployment and declarative configuration for your kubernetes cluster*, https://github.com/box/kube-applier.

[41] pulumi Github, *Tools for observing kubernetes resources in real time, powered by pulumi*, https://github.com/pulumi/kubespy.

[42] instrumenta Github, *Validate your kubernetes configuration files, supports multiple kubernetes versions*, https://github.com/instrumenta/kubeval.

[43] tufin.io, *Kubernetes network policies viewer*, https://orca.tufin.io/netpol/, 2020.

Appendix A

Network Polices

A.1 deny_all.yaml

```
#
# Deny everything
#
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  # podSelector is empty, this means it will match all the pods
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
---
```

A.2 allow-mysql-access.yaml

```
##
## Allow mysql egress access for pods in namespace default with
## label 'access-to-mysql=true'
##
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-mysql-access
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      access-to-mysql: 'true'
  egress:
  # allow access to vpc backend subnets
  - to:
    - ipBlock:
        # ntnu-sql
        cidr: 172.24.48.3/32
    ports:
    - protocol: TCP
      # mysql
      port: 3306
```

```
---
```

A.3 allow-internet-access-http.yaml

```
##
## Allow internet egress access for pods in namespace default with label:
## 'access-to-internet-http'
##
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow—internet—access—http
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      access—to—internet—http: 'true'
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        # all addresses in 10 : k8s
        - 10.0.0.0/8
    ports:
    - protocol: TCP
      port: 443
    - protocol: TCP
      port: 80
---
```

A.4 allow-ingress-to-web-tier.yaml

```
##
## Allow nginx ingress controller to all pods in web tier
##
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: nginx—ingress—web
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      tier: web
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          ingress—nginx: 'true'
---
```

A.5 allow-egress-to-dns.yaml

```
##
## Allow dns lookup from all pods in default namespace
##
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-dns
spec:
  policyTypes:
  - Egress
  # podSelector is empty, this means it will match all the pods
  podSelector: {}
  egress:
  - ports:
    # dns lookup
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
---
```

A.6 allow-egress-fromheadit-demoapp-web-to-headit-demoapp-svc.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-egress-from-headit-demoapp-web-to-headit-demoapp-svc
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: headit-demoapp-web
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: headit-demoapp-svc
```

A.7 allow-ingress-to-headit-demoapp-svc-from-headit-demoapp-web.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-ingress-to-headit-demoapp-svc-from-headit-demoapp-web
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: headit-demoapp-svc
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: headit-demoapp-web
```

A.8 allow-ingress-to-headit-demoapp-kali-from-headit-demoapp-web.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-ingress-to-headit-demoapp-kali-from-headit-demoapp-web
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: headit-demoapp-kali
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: headit-demoapp-web
```

A.9 allow-egress-from-headit-demoapp-web-to-headit-demoapp-kali.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-egress-from-headit-demoapp-kali-to-headit-demoapp-svc
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: headit-demoapp-kali
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: headit-demoapp-svc
```

A.10 allow-ingress-to-headit-demoapp-svc-b-from-headit-demoapp-web-b.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-ingress-to-headit-demoapp-svc-b-from-headit-demoapp-web-b
spec:
  policyTypes:
```

```
  - Ingress
  podSelector:
    matchLabels:
      app: headit-demoapp-svc-b
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: headit-demoapp-web-b
```

A.11 allow-egress-from-headit-demoapp-web-b-to-headit-demoapp-svc-b.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-egress-from-headit-demoapp-web-b-to-headit-demoapp-svc-b
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: headit-demoapp-web-b
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: headit-demoapp-svc-b
```

Appendix B

Configuration Files

B.1 headit-demoapp-kali.yaml

```
---
---   PROJECT: headit—demoapp—kali
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: headit—demoapp—kali—config
  namespace: default

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: headit—demoapp—kali
  labels:
    access—to—internet—http: "true"
    app: kali
    tier: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: headit—demoapp—kali
      access—to—internet—http: 'true'
      tier: web
  template:
    metadata:
      labels:
        app: headit—demoapp—kali
        access—to—internet—http: 'true'
        tier: web
    spec:
      containers:
      - name: kali
        image: kalilinux/kali:latest
        securityContext:
          capabilities:
            add: ["NET_ADMIN", "NET_RAW"]
        command: [ "/bin/bash", "-c", "--" ]
        args: [ "while␣true;␣do␣sleep␣30;␣done;" ]
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "50m"
        envFrom:
        - configMapRef:
            name: headit-demoapp-kali-config

---
apiVersion: v1
kind: Service
metadata:
  name: headit-demoapp-kali
spec:
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: headit-demoapp-kali
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress-from-headit-demoapp-kali-to-headit-demoapp-web
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: headit-demoapp-kali
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: headit-demoapp-web
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-to-headit-demoapp-web-from-headit-demoapp-kali
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: headit-demoapp-web
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: headit-demoapp-kali
---
```

Appendix C

Nmap

C.1 Nmap Scan Report - Scanned at Thu May 28 10:28:36 2020

- Scan Summary
- | kubernetes.default.svc.cluster.local (10.68.0.1)
- | headit-demoapp-web.default.svc.cluster.local (10.68.2.55)
- | headit-demoapp-svc-b.default.svc.cluster.local (10.68.2.62)
- | headit-demoapp-web-b.default.svc.cluster.local (10.68.7.69)
- | headit-demoapp-svc.default.svc.cluster.local (10.68.12.122)

C.1.1 Scan Summary

Nmap 7.60 was initiated at Thu May 28 10:28:36 2020 with these arguments:
nmap -Pn -A -sT -iL list.txt -oX nmap.xml

Verbosity: 0; Debug level 0
Nmap done at Thu May 28 10:32:15 2020; 6 IP addresses (6 hosts up) scanned in 224.53 seconds

---

C.1.2 10.68.0.1 / kubernetes.default.svc.cluster.local(online)

Address

- 10.68.0.1 (ipv4)

Hostnames

- kubernetes.default.svc.cluster.local (PTR)

Ports

The 999 ports scanned but not shown below are in state: **filtered**

- 999 ports replied with: **no-responses**

Port
State
Service
Reason
Product
Version
Extra info
443
tcp
open
https
syn-ack

fingerprint-strings

```
  FourOhFourRequest:
    HTTP/1.0 403 Forbidden
    Audit-Id: b8480a03-a218-41fe-99bf-e32155a153c2
    Content-Type: application/json
    X-Content-Type-Options: nosniff
    Date: Thu, 28 May 2020 10:29:38 GMT
    Content-Length: 212
    {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous"
    cannot get path "/nice ports,/Trinity.txt.bak"","reason":"Forbidden","details":{},"code":403}
  GenericLines, Help, RTSPRequest:
    HTTP/1.1 400 Bad Request
    Content-Type: text/plain; charset=utf-8
    Connection: close
    Request
  GetRequest:
    HTTP/1.0 403 Forbidden
    Audit-Id: 6e6a0367-e6e5-40d1-943d-7ae9d6c82d85
    Content-Type: application/json
    X-Content-Type-Options: nosniff
    Date: Thu, 28 May 2020 10:29:37 GMT
    Content-Length: 185
    {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous"
    cannot get path "/"","reason":"Forbidden","details":{},"code":403}
  HTTPOptions:
    HTTP/1.0 403 Forbidden
    Audit-Id: 2c0e3b16-ce2a-4009-aba4-31679555fb3a
    Content-Type: application/json
    X-Content-Type-Options: nosniff
    Date: Thu, 28 May 2020 10:29:38 GMT
    Content-Length: 189
    {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous"
    cannot options path "/"","reason":"Forbidden","details":{},"code":403}
```

http-title

```
Site doesn't have a title.
```

ssl-cert

```
Subject: commonName=35.204.228.51
Subject Alternative Name: DNS:kubernetes, DNS:kubernetes.default, DNS:kubernetes.default.svc,
DNS:kubernetes.default.svc.cluster.local, IP Address:35.204.228.51, IP Address:10.68.0.1
Not valid before: 2020-01-24T09:54:51
Not valid after:  2025-01-22T09:54:51
```

ssl-date

```
TLS randomness does not represent time
```

tls-nextprotoneg

```
  h2
  http/1.1
```

Remote Operating System Detection

- Used port: **443/tcp** (**open**)
- OS match: **Crestron XPanel control system** (**87%**)
- OS match: **HP P2000 G3 NAS device** (**85%**)

Metric	Value
Ping Results	user-set
System Uptime	3464783 seconds (last reboot: Sat Apr 18 08:05:52 2020)
TCP Sequence Prediction	Difficulty=252 (Good luck!)
IP ID Sequence Generation	All zeros

C.1.3 10.68.2.55 /headit-demoapp-web(online)

Address

- 10.68.2.55 (ipv4)

Hostnames

- headit-demoapp-web.default.svc.cluster.local (PTR)

Ports

The 999 ports scanned but not shown below are in state: **filtered**

- 999 ports replied with: **no-responses**

Port
State
Service
Reason
Product
Version
Extra info
80
tcp
open
http
syn-ack

fingerprint-strings

```
  GetRequest, HTTPOptions:
    HTTP/1.1 404
    Content-Type: text/html;charset=utf-8
    Content-Language: en
    Content-Length: 431
    Date: Thu, 28 May 2020 10:29:31 GMT
    Connection: close
    <!doctype html><html lang="en"><head><title>HTTP Status 404
    Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;}
    h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line
    {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404
    Found</h1></body></html>
  RTSPRequest:
    HTTP/1.1 505
    Content-Type: text/html;charset=utf-8
    Content-Language: en
    Content-Length: 465
    Date: Thu, 28 May 2020 10:29:31 GMT
    <!doctype html><html lang="en"><head><title>HTTP Status 505
    HTTP Version Not Supported</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;}
    h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;}
    p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}
    </style></head><body><h1>HTTP Status 505
    HTTP Version Not Supported</h1></body></html>
```

http-title

```
HTTP Status 404 \xE2\x80\x93 Not Found
```

Remote Operating System Detection

- Used port: **80/tcp** (**open**)
- OS match: **Crestron XPanel control system** (**87%**)
- OS match: **HP P2000 G3 NAS device** (**85%**)

Metric	Value
Ping Results	user-set
System Uptime	3394820 seconds (last reboot: Sun Apr 19 03:31:55 2020)
TCP Sequence Prediction	Difficulty=246 (Good luck!)
IP ID Sequence Generation	All zeros

C.1.4 10.68.2.62 / headit-demoapp-svc-b(online)

Address

- 10.68.2.62 (ipv4)

Hostnames

- headit-demoapp-svc-b.default.svc.cluster.local (PTR)

Ports

The 1000 ports scanned but not shown below are in state: **filtered**

- 1000 ports replied with: **no-responses**

Remote Operating System Detection

Unable to identify operating system.

Metric	Value
Ping Results	user-set

C.1.5 10.68.7.69 / headit-demoapp-web-b(online)

Address

- 10.68.7.69 (ipv4)

Hostnames

- headit-demoapp-web-b.default.svc.cluster.local (PTR)

Ports

The 1000 ports scanned but not shown below are in state: **filtered**

- 1000 ports replied with: **no-responses**

Remote Operating System Detection

Unable to identify operating system.
Misc Metrics (click to expand)

Metric	Value
Ping Results	user-set

C.1.6 10.68.12.122 / headit-demoapp-svc(online)

Address

- 10.68.12.122 (ipv4)

Hostnames

- headit-demoapp-svc.default.svc.cluster.local (PTR)

Ports

The 1000 ports scanned but not shown below are in state: **filtered**

- 1000 ports replied with: **no-responses**

Remote Operating System Detection

Unable to identify operating system.

Metric	Value
Ping Results	user-set

Appendix D

Prosjektavtale

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Headit AS (oppdragsgiver), og

Jostein Furnes
Cato Findalen
Eirik Amrud

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 21.01.2020 til 20.05.2020.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk

NTNUs veileder (navn): EIGIL OBRESTAD

Oppdragsgivers kontaktperson (navn): Rune Gaarde

Student(er) (signatur): EIRIK AMRUD [illegible] dato 21/01/2020

[illegible] dato 21/01/2020

Jostein Furnes dato 21/01/2020

____________ dato ______

Oppdragsgiver (signatur): [illegible] dato ______

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): [illegible] dato 22/1/20