

Final Report

|  |
| --- |
| School of Computing  Faculty of Engineering AND PHYSICAL SCIENCES |

Securing microservice-based healthcare applications in Kubernetes

Danielle Kushnir

Submitted in accordance with the requirements for the degree of  
MSc Computer Science

**2022/23**

**COMP3931 Individual Project**

The candidate confirms that the following have been submitted*:*

|  |  |  |
| --- | --- | --- |
| **Items** | **Format** | **Recipient(s) and Date** |
| *Final Report* | *PDF file* | *Uploaded to Minerva (DD/MM/YY)* |
| *Link to online code repository* | *URL* | *Sent to supervisor and assessor (DD/MM/YY)* |
| *User manuals* | *PDF* | *Sent to client and supervisor (DD/MM/YY)* |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) Danielle Kushnir

© 2023 The University of Leeds and Danielle Kushnir

# Summary

* **Aims, objectives and deliverables:**The concise project aim (1-3 sentences), and lists of Objectives and Deliverables. Objectives (usually 3-5) are what you want to achieve *by the end of the project* and should be measurable, *i.e.*possible to provide evidence to confirm they have been achieved. Deliverables (usually 2-3) are what you actually hand in at the end of the project, such as the Final Report, links to code repositories *etc*.

*<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>*

# Table of Contents

Summary iii

Table of Contents iv

Chapter 1 Level 1 Heading with ‘heading 1’ Style 1

1.1 Level 2 Heading with ‘heading 2’ Style 1

1.1.1 Level 3 Heading with ‘heading 3’ Style 1

1.1.1.1 Level 4 Heading with ‘heading 4’ Style 1

Chapter 2 Tables and Figures 2

2.1 Tables using the ‘table caption’ and ‘table description’ Styles 2

2.2 Figures using the ‘figure caption’ and ‘figure description’ Styles 2

List of References 3

Appendix A Self-appraisal 4

A.1 Level 2 Heading with ‘heading 2’ Style 4

A.1.1 Level 3 Heading with ‘heading 3’ Style 4

A.1.1.1 Level 4 Heading with ‘heading 4’ Style 4

Appendix B External materials 4

B.1 Level 2 Heading with ‘heading 2’ Style 4

# Chapter 1 Introduction and Background Research

## 1.1 Introduction

Deploying a system as a group of individual services working together rather than the traditional single application leads to interesting ways in which we can view security. In a use-case such as a healthcare security is crucial, so micro-service-based applications potentially offering better security has led to this investigation.

This project aims to deploy a basic web application as a microservice-based application with security features tailored specifically to the vulnerabilities a healthcare system may face. The effectiveness of the security measures will be discussed resulting in recommending the best security practices for a microservice-based application and whether the required security for a healthcare system has been achieved.

## 1.2 Background research

The initial problem with the project is understanding whether the security functionalities of Kubernetes provide the requirements demanded by the strict regulations of a healthcare system. To investigate that, it’s also important to find what these regulations are.

The background research has been split into two different sections based on the main areas that will need to be researched. The first section looks at current attempts of running medicine technologies on Kubernetes and the security requirements of a healthcare system. The second section is for investigating the current security features that are provided by Kubernetes and seeing if any fit the requirements identified in the first section. Text

Description automatically generatedTable

Description automatically generated

### 1.2.1 Healthcare systems

What is considered as a “secure system” is vague, especially when using in the context of a healthcare system. Healthcare systems tend to have strict regulations as they must collect very personal information about many people. These regulations, for the UK, require a system to comply with the Data Protection Act (ref: <https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted> ). A very relevant part in this legislation is the following quotation: “Personal data shall be processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures.” (ref: <https://www.england.nhs.uk/wp-content/uploads/2019/10/data-protection-policy-v5.1.pdf> ). Although it can be a bit vague on what is quantified as “secure” for a healthcare system, the legislation does provide some key criteria’s for what the system must cater for. These criteria are summarized in a table, shown in Table 1.1.

|  |  |
| --- | --- |
| **Security feature** | **Reference** |
| Authorisation | “Protection against unauthorised or unlawful processing” |
| Backup and restoration of data | “Protection against accidental loss, destruction or damage” |

**Table 1.1** Table summarising the two security features mentioned in the Data Protection Act (2018)

According to (ref: <https://www.healthtechzone.com/topics/healthcare/articles/2023/01/24/454713-kubernetes-the-health-industry-what-should-know.htm> ), a big reason why healthcare systems are not using microservice-based applications is due to compliance concerns. Compliance tends to provide strict regulations due to handling sensitive data. Kubernetes can help maintain the required regulations by deploying secure containerised applications and also providing many security tools that allow features such as encryption. This article however is written about American healthcare system which differs greatly from the NHS. They have stricter regulations (HIPAA) (ref: <https://www.hhs.gov/hipaa/index.html> ) compared to the UK, which only requires healthcare services to follow the Data Protection Act.

(ref: https://healthtechmagazine.net/article/2020/10/how-kubernetes-paves-way-secure-innovation-healthcare)

With the reference to encryption, the research moved to looking at how (and if) encryption is used in healthcare. In the U.S. unencrypting patient record is a violation of HIPAA regulations, however the Data Protection Act does not require encryption as it states that “appropriate” measures should be taken, and one could argue that encryption might not be appropriate since “appropriate” is a vague term. However, healthcare has data that can be used to clearly identify a person which can lead to identity thefts and other such problems that occur when an attacker gains access to a person’s sensitive information. In this scenario, encrypting patient data is an appropriate measure to take. (ref: <https://apricorn.com/data-encryption-in-healthcare> <https://www.england.nhs.uk/wp-content/uploads/2019/10/data-protection-policy-v5.1.pdf> )

According to (ref: <https://healthtechmagazine.net/article/2020/10/how-kubernetes-paves-way-secure-innovation-healthcare>), the most common security problems in healthcare systems are malware attacks and operating system patching. A malware attack is when a harmful piece of code gets on the system and causes damage, steals data, opens a backdoor for future attacks and more. Operating system patching is when a new patch is released for the operating system that fixes a vulnerability, but the patch is not secured (thus deploying this on active systems will lead to further issues). This is something that Kubernetes can help with, because the application is kept in a container as a cluster. Destroying and recreating clusters helps alleviate the pressure of having to constantly patch securities, update operating systems and search for hidden malware. (ref: <https://www.fairwinds.com/blog/how-kubernetes-is-changing-the-face-of-medical-technology> <https://healthtechmagazine.net/article/2020/10/how-kubernetes-paves-way-secure-innovation-healthcare>)

DinoCloud (ref: <https://dinocloud.co/kubernetes-preventive-medicine/>) published an article about using Kubernetes in preventative medicine and provided a list of best practices. These best practices have been summarised in Table 1.2.

|  |  |
| --- | --- |
| **Best practice** | **Description** |
| Up-to-date cluster | Use the latest version for the technologies as updates can release patches for security issues. |
| Firewalls | A firewall will prevent requests coming from the Internet. Whitelist IP accepted IP addresses and use port firewalling rules to restrict open ports. Essentially guarding possible entry points to the cluster. |
| Role-based access control | Access controls so that only users with a certain role have permission to do relevant actions, important to prevent anyone from having administrative control when they don’t need it. |
| Logs | Allows a certain user to be “blamed” for a certain action which helps in two ways. Firstly, if something in the system messes up, you can find out who it was and what they did so you can react accordingly. Secondly, these can be monitored to regulate what happens on the system and flag suspicious behaviour. |
| Network policies | Network policies control traffic in a cluster so they can permit only necessary actions and thus avoid as much bad traffic as possible. |

**Table 1.2** Table describing some best practices for using a healthcare system with Kubernetes

One of the best practices in Table 1.2 is using firewalls to safeguard communication between services. However, there is a misconception that communication between services that share the same firewall are safe. Further good practices to have are encryption when sending and storing data and certificates for authentication. When using certificates there should be a method for revoking and issuing new certificates as needed, for example when a service is killed or has been compromised.

(<https://www.traceable.ai/blog-post/6-new-requirements-for-securing-microservices-versus-monolithic-applications#:~:text=A%20monolithic%20application%20is%20by,with%20their%20own%20attack%20surface>)

Although the NHS does not provide as strict regulations for securing patient data, there is an online toolkit called Data Security and Protection Toolkit which assesses a system to “provide assurance that they are practising good data security”. However, this assessment tool requires an access code which is provided to healthcare organisations, thus it is not publicly available. (ref: <https://www.dsptoolkit.nhs.uk/>)

**Examples of Kubernetes being used in medicine:**

Utah-based CHG Healthcare which uses Kubernetes. System works by providing temporary medical staff to healthcare institutions. It has not yet had a security breach or a public scandal which happened due to leakage of data which implies that the security is adequate. Furthermore, since it is a U.S. healthcare company, it must comply to HIPAA’s strict regulations. If it has managed to implement a secure Kubernetes system in healthcare that manages to follow the necessary compliances it means that this is an example of a successful microservice-based application in healthcare.

(ref: <https://chghealthcare.com/>

<https://thenewstack.io/containers-kubernetes-fuel-innovation-in-medical-staffing/>)

### 1.2.2 Kubernetes security

There are two main types of applications, monolith and microservices. Monolith are the more common types of application and it is where a service is deployed as one single application, whereas a microservice-based application is comprised of many applications working together to provide a service.

Naturally, microservices have more attack surfaces than a monolith application (which has just one attack surface). However, this is not as bad as it seems. The attack surface in a monolith application is much larger since the entire architecture is comprised of that single application. On the other hand, a microservice architecture is made up of potentially smaller independent services thus the attack surfaces are smaller. It is also easier to isolate which service has a security issue since the services are separated. With more attack surfaces, securing each surface can add latency and thus degrade performance which would really be unideal in hospital systems if urgency is required for treating a patient. A healthcare system can get very complex, for example a general hospital system will have to work with a large amount of data to provide emergency services, in-patient services, surgery scheduling and more. (Ref: <https://techbeacon.com/app-dev-testing/4-ways-exploit-microservices-architecture-better-app-sec> <https://nordicapis.com/which-is-more-secure-monolith-or-microservices/> <https://www.traceable.ai/blog-post/6-new-requirements-for-securing-microservices-versus-monolithic-applications#:~:text=A%20monolithic%20application%20is%20by,with%20their%20own%20attack%20surface> <https://techbeacon.com/app-dev-testing/4-ways-exploit-microservices-architecture-better-app-sec>)

Another advantage of microservices is that they have less chance of a cascading attack. A cascading attack is when a vulnerability of one service also leads to the vulnerability of another. There is a lesser risk of this type of attack occurring because each service in a microservice-based application is independent, therefore a vulnerability in one cannot be passed since it will only affect this individual service. Whereas a monolith is inherently coupled, so one vulnerability can affect the entire system. (ref: <https://nordicapis.com/which-is-more-secure-monolith-or-microservices/> <https://dzone.com/articles/increase-security-by-transitioning-from-monolith-t>)

Deploying an application in a container provides an extra layer of security. To quote Gain, (ref: <https://thenewstack.io/microservices-security-probably-not-what-you-think-it-is/>

) "Their isolated and stand-alone structure within applications makes them easier to defend". Although securing a microservice network is much more difficult as it's more complex than a monolith, securing the application as a microservice prevents an attack from spreading sooner.

In an article published by (ref: <https://dzone.com/articles/increase-security-by-transitioning-from-monolith-t> ), they define a secure application to be “one with fail-safes and monitoring that handles disruptions happening across the application infrastructure”. In a monolith architecture, if there is a security problem it can be very difficult to isolate potential security threats as the whole system may be compromised. Locking down only parts of the system is difficult due to the nature of a monolithic architecture and turning the entire system off to fix one problem is not ideal. A microservice-based architecture does not have this issue because you can turn off a service while allowing the rest of the system to function. Furthermore, since each individual service can run on different servers, it minimises the risk of the entire system breaking down due to one attack or having issues such as memory leaks. Consequently, microservice-based applications follow this definition of a secure application.

There are many exploits involving vulnerabilities in systems that aren't up to date. It's much slower to update a monolith architecture but patching microservices is easier and quicker. Monolithic applications are a big mix of code so it can be difficult to know how it works, which means that a security error in one may results in the same error in the other and they won't know this error exists because they aren't aware of other people's code. It can be very challenging to find the root of this problem when the code is varied due to many different contributions. Microservices makes it easier to split the work and allow collaboration since they work on individual services instead of contributing to the same complex one. They also spread out the functionality so that not all of them can be publicly accessed, which shields this specific service from the public and thus makes it difficult for a third party to gain access. (ref: <https://techbeacon.com/app-dev-testing/4-ways-exploit-microservices-architecture-better-app-sec> <https://www.traceable.ai/blog-post/6-new-requirements-for-securing-microservices-versus-monolithic-applications#:~:text=A%20monolithic%20application%20is%20by,with%20their%20own%20attack%20surface>)

Kubernetes is self-monitoring and self-healing which help minimize operational vulnerabilities by adding more automation (and thus also minimising human error). Kubernetes "provide a secure framework for data protection” and may be able to give healthcare systems “the security and compliance they need".

(ref: <https://www.fairwinds.com/blog/how-kubernetes-is-changing-the-face-of-medical-technology> <https://healthtechmagazine.net/article/2020/10/how-kubernetes-paves-way-secure-innovation-healthcare>) However one of the biggest issues with Kubernetes is if a third party manages to gain access to the underlying operating system. (ref: <https://www.youtube.com/watch?v=oBf5lrmquYI> ).

Kubernetes comes with a Kubernetes Dashboard which is a web-based user interface for the management of a Kubernetes cluster. Since web-based applications tend to be more intuitive than terminals this could help prevent human errors and provide ease-of-use. This is specifically useful in a security sense because it's much easier to monitor the state of a cluster visually so security issues can be identified quickly. You can also view logs and metrics which may provide information with vulnerabilities, internal attacks that leak information or weird behaviour that could lead to the discovery of malware. The malware detection is especially good for this use-case because it was established in the previous section that malware attacks are one of the most common attacks in healthcare systems. Finally, the dashboard provides access control functionalities, which is one of the best practices discussed in the previous section.

(ref: <https://www.healthtechzone.com/topics/healthcare/articles/2023/01/24/454713-kubernetes-the-health-industry-what-should-know.htm> )

ADD THIS TO RESEARCH: https://www.cisecurity.org/benchmark/kubernetes

Using the Kubernetes documentation (ref: <https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/>) I have constructed a list of ways to secure a cluster, along with contributions from other sources (ref: <https://gitlab.com/nanuchi/k8s-in-1-hour> <https://www.youtube.com/watch?v=oBf5lrmquYI> <https://kubernetes.io/docs/concepts/security/overview/> ). These can be seen in Table 1.3. In addition to the recommendations given in the documentation, they also provide a security checklist. (ref: <https://kubernetes.io/docs/concepts/security/security-checklist/> )

|  |  |
| --- | --- |
| **Security practice** | **Description** |
| TLS for communication traffic | By default communication between pods is unencrypted. An attacker can easily read the communication between the pods because they are in plaintext. Although TLS can have performance issues, it uses encryption and certificates which protect communication and can help identify insecure traffic. |
| Authorisation using Role-Based Access Control | Only allow certain users/groups to do certain actions. Keep as restrictive as possible so they only do what they absolutely need to. |
| Controlling access to the Kubelet | The Kubelet (the primary node) allows unauthenticated access by default, so this should be restricted. |
| Limit resource usage on a cluster | Relates to the idea of using policies to limit the number/capacity of resources and the size of resources to prevent users from requesting unreasonable values. |
| Container privileges | Avoid using root and running containers as root. If a container is compromised this provides the attacker with easier privilege escalation. Prevent containers from loading unwanted kernels (clear security issue because it is as if the attacker had access to a computer’s terminal/command prompt). |
| Network policies | Restricting network access using policies. By default all pods can communicate with every other pod in their namespace. |
| Pods access control | By default all pods can talk to each other, so if an attacker gains access to one they can access any other pod. Remove any unnecessary communication between pods. |
| Restrict access to etcd | Should be limited to control plane only because etcd stores the data of the cluster. An attacker can bypass the API server and destroy/update resource if they gain access to etcd. They can also gain access to the data in resources. |
| Audit logging | For analysing compromise. |
| Secrets | Use secrets for confidential information since data is encoded instead of stored as plaintext. This will need to be encrypted with a third party tool. |
| Network access to control plane | Restrict which addresses can access the cluster instead of allowing it to be public. |
| Network access to nodes | Nodes should only accept connections from control plane on specific ports, NodePort and LoadBalancer. |
| Encrypt etcd | Encrypt all storage at rest. |
| Scan containers for known vulnerabilities |  |
| Sign container images | To create trusted content in containers. Images from untrusted sources may have malware or backdoors or they may be using vulnerable tools/libraries. It is important for images to be secure because an attacker can break out of the container the image is in and gain access to the host and from there access every other container that run on this host. From here they can access any data stored and see Kubelet configuration files which they can use for further attacks. |
| Limit port ranges | Less possible entry points in the application. |
| Static code analysis | Discover common vulnerabilities in code. |
| Automate popular attacks | To see if popular attacks penetrate the system. |
| Automated restoration of data using backups | Backup data in the case of losing and leaking of data or even ransomware attacks. Should be a system that automates regular backups and restoration of data. Stores the backup safely (so you cannot lose the backup too). |
| Configuration security policies | Define security policies that enforce specific configurations so you can enforce a desired behaviour all the time rather than relying on the developers to understand how to configure an application securely. |

**Table 1.3** Table showing different ways of securing a cluster

# Chapter 2 Methods

Is it clear that…

* The solution exists?
* I have produced the deliverables specified?
* Appropriate steps or standards were taken to ensure a quality output (*e.g.* design documentation, comments in code)?
* The challenges were clearly articulated?

Text

Description automatically generated

Text

Description automatically generated with medium confidence

## 2.1 Sprint 1 – Constructing the problem

This was a four week sprint that started on September 14th and ended October 14th. Time here was used to investigate the problem – how Kubernetes is currently used in healthcare systems (if applicable). Research was also done on general Kubernetes knowledge such as what is Kubernetes and how a Kubernetes cluster functions. All the information was briefly summarised with its reference so it could be drafted in a later sprint.

One of the biggest challenges discovered in this sprint was working with the image. It will take further effort to create a custom image for testing security features while also following image security measures, hence it was decided that a pre-existing image will be used instead, this has been linked in the external materials section.

## 2.2 Sprint 2 – Healthcare security

Sprint 2 lasted around four weeks, from October 17th to November 18th. The main goal of this sprint was to research security in healthcare systems and what sort of standards/compliances a system must have.

## 2.3 Sprint 3 – Kubernetes in healthcare

Similarly to Sprint 1 and 2, Sprint 3 lasted around four weeks, from November 21st to December 23rd. The main goal here was to research Kubernetes and healthcare as well as finding any existing solutions that use Kubernetes.

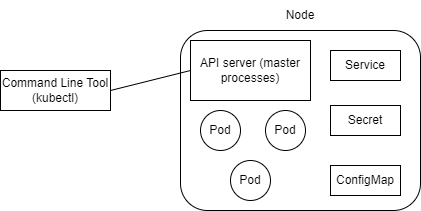
## 2.4 Sprint 4 – Kubernetes security

This sprint started on December 1st and ended on December 30th. This was the last research sprint, where most of the research focused on security features offered by Kubernetes. Many of these features overlapped with the requirements collected in Sprint 2.

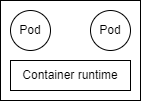
## 2.5 Sprint 5 – Deploy application

Sprint 5 lasted from January 1st to February 10th. The designs of the application used for this sprint can be seen in figures 2.1-2.4. The second half of the sprint was for configurating the application following the application’s architectural design. The configuration files from this sprint can be seen on the GitHub repository, as seen in Figure 2.5.

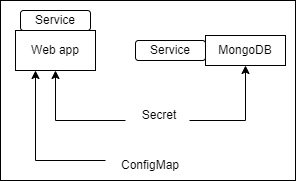
As part of the design of the application, a security features checklist was also created, which can be seen in Table 2.1. Although the plan is to implement all of the security features, test and discuss them, realistically it may pose a challenge due to time. Therefore the main purpose of the checklist is to track what features have been investigated.



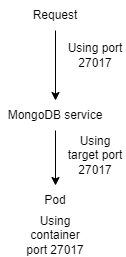
**Figure 2.1** The API server enables the interaction with cluster. The Command Line Tool provides a way of communicating with the API server. A user interface or API can be used instead of Kubectl, but Kubectl is the most powerful of the three because it lets you do anything in the cluster. Pods are the worker processes. (ref: <https://gitlab.com/nanuchi/k8s-in-1-hour> )



**Figure 2.2** Diagram of the Minikube cluster which runs as a Docker container **.**

****

**Figure 2.3** Project overview. The MongoDB service forwards requests to MongoDB pods. The ConfigMap stores MongoDB’s endpoint. The Secret stores MongoDB credentials (username and password which should not be stored as plaintext). There are two deployed services in the architecture, the MongoDB app with internal service and the example website with external service.



**Figure 2.4** How request forwarding works on this application.

Graphical user interface, text, website

Description automatically generated

**Figure 2.5** Sprint 5 commit with a typo in the commit message, it should say “Sprint 5” instead of “Sprint 1”. However, since it was the first programming sprint it was mistakenly labelled.

|  |  |
| --- | --- |
| **Security feature** | **Investigated? (Yes/No)** |
| TLS for communication traffic | https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/ |
| Limit resource usage on a cluster | https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ |
| Container privileges | https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ |
| Network policies | https://k21academy.com/docker-kubernetes/network-policies-in-kubernetes/ |
| Pods access control | https://medium.com/@LachlanEvenson/hands-on-with-kubernetes-pod-security-admission-b6cac495cd11 |
| Restrict access to etcd | https://sysdig.com/blog/kubernetes-security-kubelet-etcd/ |
| Audit logging | https://www.containiq.com/post/kubernetes-audit-logs |
| Secrets instead of ConfigMap |  |
| Encrypt etcd (encrypt secrets at rest) | https://blog.knoldus.com/how-to-encrypt-kubernetes-secrets-with-sealed-secrets/ |
| Scan containers for known vulnerabilities |  |
| Limit port ranges |  |
| Automate popular attacks | https://www.reliaquest.com/blog/best-practices-for-detecting-5-common-attacks-against-kubernetes/ |
| Automated restoration of data using backups | https://www.youtube.com/watch?v=01qcYSck1c4 |
| Role access-based control | https://www.strongdm.com/blog/kubernetes-rbac-role-based-access-control |

**Table 2.1** Project security features checklist

## 2.6 Sprint 6 – Initial container scan

Scanning containers for vulnerabilities is one of the items on the project’s security feature checklist. To track the project’s progress, this sprint will focus on initially setting up the external applications that will be used to scan the container vulnerabilities and extract a set of results. In the final sprint of the project, these same applications will be used to scan the container again and the results will be compared to help evaluate how secure the microservice-based application is. This sprint lasted five days, from Monday 27th of February to Friday 3rd of March. Two applications were set up during this time.

The first security application is called kube-bench, developed by Aqua Security (ref: <https://github.com/aquasecurity/kube-bench/>). This is a tool for checking if pods in a Kubernetes container comply with CIS Kubernetes Benchmark. The way the container uses this application is by adding another pod which runs kube-bench. The pod will run as a job and then complete. Once completed it provides a detailed log. Both the kube-bench pod configuration file and the result of this log can be found in the “External” folder. Kube-bench tests for a lot of vulnerabilities, in summary for policies there were 0 passes, 0 failures and 30 warnings and in total there were 58 passes, 13 fails and 53 warnings.

The second security application is developed by the same company, Aqua Security, and is called kube-hunter (ref: <https://github.com/aquasecurity/kube-hunter/> ). This application checks for container vulnerabilities. Unlike the first application, the configuration file for the pod running kube-hunter was taken directly from its GitHub page instead of being stored in the “External” folder (ref: <https://medium.com/@heshani.samarasekara/kube-hunter-installing-and-testing-3ea6cfc09738> ). The results of this application can be found in a detailed log in the “External” folder.

Along with comparing the results of vulnerability scanners initially and at the end of the project, the vulnerability scanners also provide additional security tasks to complete as part of the “Scan containers for known vulnerabilities”. For example, one of the vulnerabilities that kube-hunter found in the container is that the container’s Kubernetes version is exposed, which could help an attacker because there might be a vulnerability associated with this particular version.

## 2.x Sprint x – Securing secrets

In the current state of the application before this sprint, not a lot of security measures have been taken to protect secrets in the cluster. The background research led to the discovery that if configuration information needs to be kept secure, instead of using the traditional ConfigMap, a Secret should be used and encoded so that it is not stored as plaintext. In this applications there is only one Secret and it is used to store the MongoDB username and password. A healthcare system will most likely be using a database to store patient data, which is very sensitive thus should remain confidential. Since it is vital to prevent an attacker from accessing all the patient data by simply using these credentials, it has been encoded using Base64 encoding. However, this is still not secure enough. An attacker can simply convert Base64 back to plaintext and get the password. The next step to take is to encrypt the secret, which is unfortunately something that Kubernetes does not provide (in a secure way), but there are many third-party tools which do this.

There are three main security limitations of secrets in Kubernetes:

1. There is no way to manage which containers can access specific secrets.
2. Storing secrets inside container images exposes it to many more potential vulnerabilities.
3. Containers store secrets in situations when they do not need to be stored.

(ref: <https://blog.aquasec.com/managing-kubernetes-secrets> )

Due to these reasons, this sprint focuses on securing secrets in the cluster.

# Chapter 3 Results

# Chapter 4 Discussion

(including results) is it clear that…

* Appropriate tests and evaluation were conducted and analysed to validate the quality of deliverables within the remit of the project?
* There are objective criteria for evaluating the achievement of the project against the initial problem?
* These criteria are justified?
* The criteria have been used professionally to judge whether the problem has been solved?
* **Project plan**: Description and graphical representation (*e.g.* a Gantt chart) of the full project timeline.
* **Risk Mitigation:** Identification of likely risks and strategies for mitigation.

Text

Description automatically generated

Text

Description automatically generated

## 4.1 Conclusions

<Text in 11-point size and 1.5 line spacing.>

## 4.2 Ideas for future work

<Text in 11-point size and 1.5 line spacing.>

# List of References

*<It is expected that the list would reflect the breadth and depth of scholarly research undertaken by the student during the course of the project.>*

Used:

<https://gitlab.com/nanuchi/k8s-in-1-hour>

look into these:

given book security section

# Appendix A Self-appraisal

<This appendix must contain everything covered under the ’self-appraisal’ criterion in the mark scheme. Although there is no length limit for this section, 2-4 pages will normally be suﬃcient. The format of this section is not prescribed, but you may like to consider the following sections and subsections.>

Have I…

* Considered how my work could be extended?
* Included mature personal reflection which leads to lessons learnt?
* Suggested how the problems encountered might be avoided?
* Considered legal, social, ethical and professional issues, with justification if one or more are not relevant?



## A.1 Critical self-evaluation

## A.2 Personal reﬂection and lessons learned

## A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

### A.3.1 Legal issues

<Discussion of legal issues>

### A.3.2 Social issues

### <Discussion of social issues>

### A.3.3 Ethical issues

### <Discussion of ethical issues>

### A.3.4 Professional issues

<Discussion of professional Issues>

# Appendix B External Materials

<https://kubernetes.io/docs/home/>

<https://minikube.sigs.k8s.io/docs/start/>

<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

<https://www.mongodb.com/docs/>

<https://gitlab.com/nanuchi/developing-with-docker/-/tree/feature/k8s-in-hour/app>

<https://github.com/aquasecurity/kube-bench/>