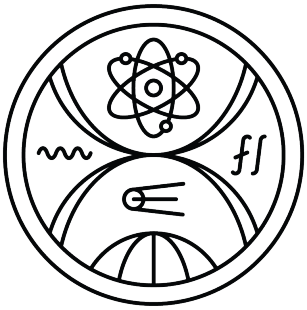
COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

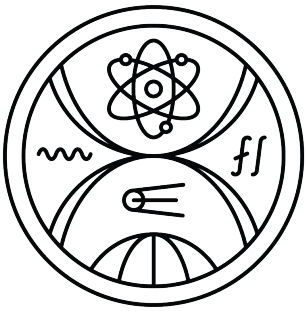


KUBERNETES SECURITY ASSESSMENT

Master thesis

2024 Bc. Pavel Semenov

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS



KUBERNETES SECURITY ASSESSMENT

Master thesis

Study program:	Applied informatics
Branch of study:	Applied informatics
Department:	Department of Applied Informatics
Supervisor:	prof. RNDr. Richard Ostertág, PhD.
Consultant:	Mgr. Ľubomír Firment

Bratislava, 2024 Bc. Pavel Semenov

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Pavel Semenov
Study programme: Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Diploma Thesis
Language of Thesis: English
Secondary language: Slovak

Title: Kubernetes security assessment

Annotation: Kubernetes has been gaining popularity rapidly in recent years as more and more enterprise solutions are subjected to cloud transformation and more companies are looking for the ways to increase development efficiency and reduce development costs. This brings new concerns from clients and stakeholders about the security of Kubernetes and its exposure to cyber-attacks.

Aim:

This thesis studies, compares and evaluates the state-of-the-art tools designed to discover vulnerabilities concerning the cluster configuration, running pods or cluster itself. Assessment is carried out in both local cluster setup predisposed with multiple vulnerabilities and real-world enterprise cloud infrastructure. Based on the assessment results we intend either to improve one of the existing tools or develop a Kubernetes security framework of our own, which will be able to provide better results in addressing the cluster security.

Literature:

V. B. Mahajan and S. B. Mane, "Detection, Analysis and Countermeasures for Container based Misconfiguration using Docker and Kubernetes", 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), 2022, pp. 1-6, doi: 10.1109/IC3SIS54991.2022.9885293. https://ieeexplore.ieee.org/document/9885293

D. B. Bose, A. Rahman and S. I. Shamim, "'Under-reported' Security Defects in Kubernetes Manifests", 2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS), 2021, pp. 9-12, doi: 10.1109/EnCyCriS52570.2021.00009. https://ieeexplore.ieee.org/document/9476056

Castillo Rivas, D.A., Guamán, D. (2021). "Performance and Security Evaluation in Microservices Architecture Using Open Source Containers". In: Botto-Tobar, M., Montes León, S., Camacho, O., Chávez, D., Torres-Carrión, P., Zambrano Vizuete, M. (eds) Applied Technologies. ICAT 2020. Communications in Computer and Information Science, vol 1388. Springer, Cham. https://doi.org/10.1007/978-3-030-71503-8_37

Clinton Cao, Agathe Blaise, Sicco Verwer, and Filippo Rebecchi (2022). "Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster". In Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES '22). Association for Computing Machinery, New York, NY, USA, Article 117, 1–9. https://doi.org/10.1145/3538969.3543810

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Computing Machinery, New York, NY, USA, Article 117, 1–9. https://doi.org/10.1145/3538969.3543810

Vedúci: RNDr. Richard Ostertág, PhD.
Konzultant: Mgr. Ľubomír Firment
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 07.12.2022

Dátum schválenia: 07.12.2022 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis advisor.

.....

Bratislava, 2024 Bc. Pavel Semenov

Acknowledgement

First, I would like to express my gratitude to Mgr. Ľubomír Firment for his guidance during the whole thesis and invaluable expertise in Kubernetes that made this thesis possible. I'd also like to thank my supervisor prof. RNDr. Richard Ostertág, PhD. for his insightful feedback.

Abstract

Kubernetes has been gaining popularity rapidly in recent years as more and more enterprise solutions are subjected to cloud transformation and more companies are looking for the ways to increase development efficiency and reduce development costs. This brings new concerns from clients and stakeholders about the security of Kubernetes and its exposure to cyber-attacks. This thesis studies, compares and evaluates the state-of-the-art tools designed to discover vulnerabilities concerning the cluster configuration, running pods or cluster itself. Assessment is carried out in both local cluster setup predisposed with multiple vulnerabilities and real-world enterprise cloud infrastructure. Based on the assessment results we intend either to improve one of the existing tools or develop a Kubernetes security framework of our own, which will be able to provide better results in addressing the cluster security.

Keywords: kubernetes, security, test, cloud

Abstrakt

Kubernetes v posledných rokoch rýchlo získava na popularite, pretože čoraz viac spoločností hľadá spôsoby, ako zvýšiť efektivitu vývoja a znížiť náklady na vývoj. Táto zvýšená popularita so sebou prináša väčšie vystavenie kybernetickým útokom a zvýšené obavy zainteresovaných strán o bezpečnosť Kubernetes. Cieľom práce je porovnať a zhodnotiť moderné nástroje určené na odhaľovanie zraniteľností týkajúcich sa konfigurácie klastra, bežiacich podov alebo aj samotného klastra. Posúdenie bude prebiehať na lokálnom klastri s prednasadenými viacerými zraniteľnosťami, ako aj v reálnej podnikovej cloudovej infraštruktúre. Na základe výsledkov hodnotenia máme v úmysle buď vylepšiť niektorý z existujúcich nástrojov, alebo vyvinúť vlastný bezpečnostný rámec pre Kubernetes, ktorý bude schopný poskytnúť lepšie výsledky pri riešení klastrovej bezpečnosti.

Kľúčové slová: kubernetes, bezpečnosť, testovanie, cloud

Contents

List of Figures

List of Tables

Terminology

Terms

- **CI/CD pipeline**
  CI/CD pipeline is a set of automatic tasks to be executed upon specific action resulting in ether failure on some of the pipeline stages or successful application deployment on the target environment. The aforementioned action might be a push into the source code repository or a manual pipeline run request. CI/CD pipeline usually includes build, test and deploy stages.
- **Cloud**
  Term "Cloud" is ususally used to describe an array of (remote, on-premise or hybrid) servers that operate as a single ecosystem used for various hosting services.
- **Cloud provider**
  A cloud provider is a company that offers cloud computing services, which include resources like storage, processing power, networking, databases, and software, delivered over the internet.

Abbreviations

- **K8s** - Kubernetes.
- **OS** - Operating System.
- **VM** - Virtual Machine.
- **CI/CD** - Continuous Integration/Continuous Deployment.
- **RBAC** - Role Based Access Control.
- **AWS** - Amazon Web Services.
- **TLS** - Transport Layer Security.
- **CSRF** - Cross Site Request Forgery.
- **XSS** - Cross Site Scripting.

- **OWASP** - The Open Worldwide Application Security Project.
- **TCP** - Transmission Control Protocol.
- **API** - Application Programming Interface.
- **NFS** - Network File System.
- **iSCSI** - Internet Small Computer Systems Interface.

Motivation

As the world's biggest corporations start grasping the power of the Cloud Computing, stakeholders are raising concerns regarding the security of the most popular and accessible Container Orchestration platform - Kubernetes. Opinion of the experts on this matter varies significantly and this paper aims to make a contribution to this dispute by determining how well can be Kubernetes cluster's security monitored.

We compare and evaluate the capabilities of the most popular security tools designed specifically for Kubernetes against official and unofficial Kubernetes security recommendations.

Chapter 1

Introduction

1.1 Containerization

When talking about the Kubernetes it is essential to be familiar with the technology of containerization. This chapter introduces the reader to the basics of the containerization. We start by giving a short definition, then we examine core concepts of the containerization such as container image and Docker. Then we compare it to the traditional means of application deployment. Finally, we examine the security on the container image level.

1.1.1 Overview

According to IBM, containerization is the packaging of software code with just the operating system libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. [2].

Although containers are built to be infrastructure-agnostic, there are still certain compatibility considerations to keep in mind. One significant factor is processor architecture. Containers built for a specific architecture family (e.g., arm64, amd64, or x86) are generally not cross-compatible with infrastructures based on different architectures. However, it is possible to build multi-architecture containers that support multiple processor architectures in a single image, enhancing the flexibility and portability of the containerized applications across diverse environments.

1.1.2 Container Image

Container images are software application packages, which are shipped with all required libraries, binaries and configurations. In another words, container images are lightweight and higly portable artifacts. Usually container images are stored in Con-

tainer Registries, either public (e.g. Dockerhub) or private. When an image transitions into the running state, it becomes a container.

Container images are comprised of multiple layers. At the base layer there is usually some lightweight Linux distribution. Then, each layer introduce a change in the environment, a change might be in the code or binary, runtimes, dependencies, and other filesystem objects required to run an application.

1.1.3 Docker

Docker is the most widely used containerization tool. It ships tools to build, run and store containers. Docker Engine is a collective name for the Docker build toolkit. First, there is a `dockerd` or Docker Daemon, which is a server running in the backend. Secondly, the user intercts with Docker CLI or Docker Client to build and run images. Docker Clients communicates with Docker Daemon through the Rest API served at the backend. Then, there is Docker Compose, which a simple orchestration tool for the containers. It can be used to compose a few containers into a system with a shared network and storage. Lastly, Dockerhub is a public container registry, where the bulk of the publicly available images are stored.

Images are built based on the Dockerfile, which is a set of instructions. Each instruction introduces a new layer to the image. Layers can then be smartly reused by the Docker Engine to build different images with similar bases. Listing 1.1 demonstates an example of the Dockerfile. Each Dockerfile starts with a `FROM` command, which specifies the base image to use for this container. The `FROM` keyword is followed by the base image name. In our case, `registry.redhat.io` is the registry address. It is followed by the repository name (`ubi8` in our case), which is separated from the registry name by a single slash and may be preceeded by the namespace. Lastly, tag follows repository name and separated with a colon from it. Both registry address and tag are optional. In case registry name is omitted, Docker will search for the image in the Dockerhub. If the tag is not provided, Docker will fetch `latest` tag.

```
FROM registry.redhat.io/ubi8:latest
RUN dnf install nodejs && \
    useradd -u 1000 -g 1000 -M node
COPY --chown=node:node src /app
WORKDIR /app/src
USER node
RUN npm ci
EXPOSE 3000
ENTRYPOINT ["npm", "run"]
```

Listing 1.1: A simple Dockerfile for a NodeJS app.

`RUN` command is used to run commands inside the container during the build phase. All artefacts generated by the `RUN` command stay inside the contaier and can be used during runtime. In our case we are installing the necessary binaries to run our application and creating a user to run the application. `COPY` command copies the specified resource from the local machine into the container. We also are changing the ownership for the copied files. `WORKDIR` command affects the commands after it, so that they are executed from the specified directory. `USER` command changes the current user. Last `USER` command inside the Dockerfile determines under which user the main process inside the container is running in the runtime. In our case we use our newly created `node` user. The default user for the container is `root`. `EXPOSE` command ensures that a specific port on the container is open for the external communication. Lastly, `ENTRYPOINT` is one of the few ways to define the main process of the container. When main process ends, the container stops.

1.1.4 Containerization vs Virtualization

Let us discuss why containerization is the internationally accepted enterprise solution nowadays and why do software arhitects tend to choose it over traditional virtualization solutions.

Figure 1.1 provides a side-by-side comparison of a Virutal Machine and a Cloud infrastructure, each with three applications deployed. We can see that each application on the container side is missing a "Guest OS" layer. Here lies the main advantage of the containers. Absence of the Guest Operating System provides a lot of advantages, which we discuss further below.

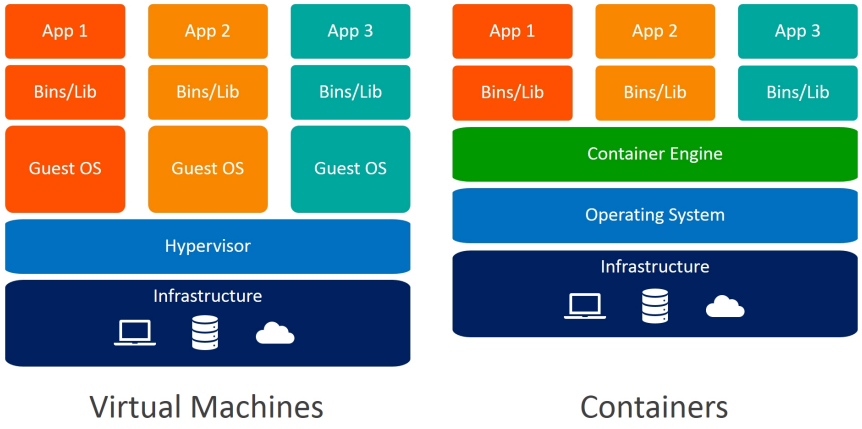


Figure 1.1: Side-by-side comparison of VM and container infrastructures.

The most important advantage of the containers is their resource efficiency. Containers only include the application code and its dependencies, which makes them very small compared to the Virtual Machines, which tend to be very bulky and grow in size

as development progresses. Containers share the host operating system kernel, so they consume significantly less CPU, memory, and storage than virtual machines, which require a full OS for each instance. This lightweight nature allows more containers to run on a single host, maximizing resource utilization and reducing overhead. Better resource efficiency means also lower costs for the user.

Then, startup speeds are significantly lower for the containers as they do not need to initialize the whole OS boot sequence. This feature also enables the scaling capabilities for the containers, allowing applications to respond quickly to changes in demand.

Additionally, the containers are more consistent than virtual machines. Packed with the required dependencies, they behave in the same way across different environments. As they are isolated from the OS, containers are almost immune to the compatibility issues. This gives them a strong portability advantage. They provide an abstraction that makes it easier to move workloads across various platforms.

Lastly, containers also lead when it comes to automation and CI/CD pipelines. Containers can be easily versioned, updated, and rolled back, allowing for smooth integration into CI/CD pipelines. This streamlines deployment, testing, and rollback processes, leading to faster development cycles. VMs can also be updated and rolled back, but the process is usually slower and more complex.

These are some of the most significant advantages of containerization. All of them contribute to fast build and deploy speeds, which also means high development speeds. While costing less money and providing a lot more flexibility, they become essential for successful enterprise software development. For large-scale development, test and production environment containerization has become an obvious choice over the virtualization.

1.1.5 Security Concepts

1.2 Kubernetes

In this section we will dive into the topic of Kubernetes. We start by shortly overviewing the Kubernetes architecure. Then we expand on the Kubernetes resources, their kinds and their role in the target application environment. Finally, we consider the security of Kubernetes.

1.2.1 Overview

Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes

builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community. [1].

IBM defines Kubernetes as an open source **container orchestration platform** for scheduling and automating the deployment, management and scaling of containerized applications. Today, Kubernetes and the broader ecosystem of container-related technologies have merged to form the building blocks of modern cloud infrastructure. This ecosystem enables organizations to deliver a highly productive hybrid multicloud computing environment to perform complex tasks surrounding infrastructure and operations. It also supports cloud-native development by enabling a build-once-and-deploy-anywhere approach to building applications. [3].

It is important to emphasize that the Kubernetes abstracts the actual machines (nodes) from the user. Nodes can be physical on-premises servers, or VMs that reside either on-premises or at a cloud provider. Kubernetes takes on the responsibilty of figuring out the deployment target for a particular application. That is, user only defines the desired state of the infrastructure using YAML or JSON configuration files. Kubernetes then creates all the workloads based on the applied configuration.

1.2.2 Kubernetes Architecture

While Kubernetes requires at least three nodes to run, there are some implementation designed for the local use, which emulate this concept (see Subsection 1.2.5). The master node is called control plane. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability. Worker nodes host the actual workload inside the cluster. Figure 1.2 provides a simple high-level overview of a typical Kubernetes cluster with a Control Plane and three worker nodes.

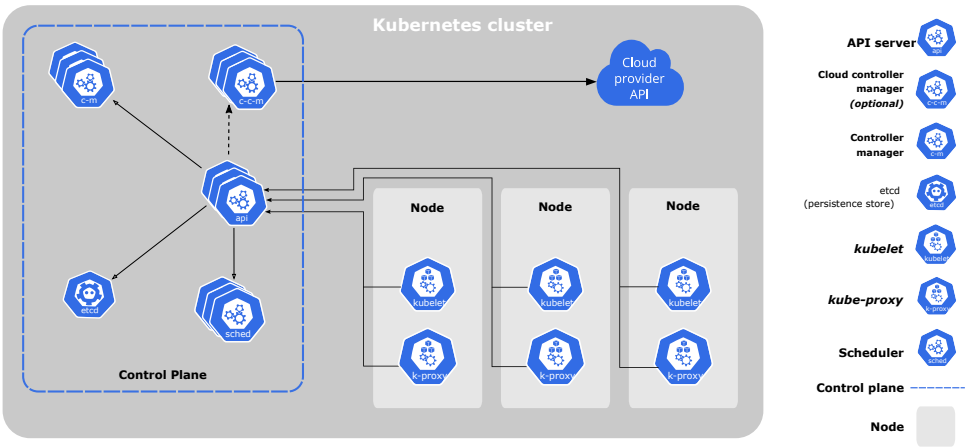


Figure 1.2: Kubernetes cluster architecure overview with its main components.

Control Plane overview

Control plane runs the following components:

- **kube-apiserver**
  Kube-apiserver exposes the Kubernetes API, which is acting as a frontend for the Kubernetes control plane.

- **etcd**
  Etcd is a key-value store, where all of the cluster data is stored.

- **kube-scheduler**
  Each time a new pod is created, it is passed to the kube-scheduler, which assigns the pod to the specific node to run on (based on individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines).

- **kube-controller-manager**
  Each Kubernetes resource has its own controller (e.g. NodeController, JobController, ServiceAccountController); all of them are compiled as one binary called kube-controller-manager.

- **cloud-controller-manager**
  Cloud-controller-manager embeds cloud-specific control logic. It differs depending on the cloud provider or can be absent completely, when running Kubernetes locally.

Worker Node overview

Each worker node has a kubelet and kube-proxy installed. Kubelet is an agent that manages runnning pods and containers. Kube-proxy is a network proxy that implements parts of the Kubernetes service concept. It maintains network rules on nodes, making in- outside-cluster communitcation possible.

Then, of course, each worker node has a set of running pods. A typical use would involve multiple running applications. Depending on the size of the node and the application resoucre consumption it can accomodate on average from one to a few dozens applications with various business purposes.

1.2.3 Kubernetes Resources

Kubernetes resources are fundamental components that define various entities within a Kubernetes cluster. Resources are objects that represent the desired state and configuration of the infrastructure, applications, and services running on the cluster. Ku-

bernetes provides a range of resources that enable developers and operators to define, manage, and scale containerized applications, network policies, storage requirements, and more. These resources are defined declaratively in YAML or JSON files, which makes infrastructure setup consistent and reproducible.

Among key Kubernetes resoucres are:

- **Pods**
  Pod is the atomic workload unit in the Kubernetes cluster. It encapsulates one or more containers that share the same network. It represents a single instance of a running application.

- **Deployments**
  Deployments are a higher level of abstraction for the Pods. They allow to define replica count and rollout/rollback strategy for the updates, which can be used to ensure availability for the application.

- **Services**
  Services provide a communication layer for the pods inside one cluster. Being an abstraction over the pods' network, they provide reliable access to the selected workloads, while serving as a Load Balancer.

- **ConfigMaps** and **Secrets**
  ConfigMaps and Secrets allow users to store data outside the workload. ConfigMaps are usually used to store non-sensitive information like environment and application configuration parameters. Secrets are a more secure resource designed for API keys, passwords and other sensitive data.

- **PersistentVolumes** and **PersistentVolumeClaims**
  These resources enable stateful applications to request and mount durable storage within a cluster, allowing data to persist independently of the Pod lifecycle.

Above are the most commonly used resources, which we also leverage in the practical part of the paper. Therefore, it is important that the reader understands the position and the purpose of each resource in the cluster infrastructure.

Workloads

Minimal computing units in Kubernetes are Containers, which are running in Pods. However, to simplify the management of Pods, Kubernetes has workload resources, which manage the set of Pods. They make sure the desired number of Pods of right kind are running to match the declaration.

Deployments and ReplicaSets are a good fit for stateless applications. Each pod in the Deployment is interchangebeable. Deployments are easily scalable and have built-in versioning and rollout mechanisms.

StatefulSet allows to create sets of stateful applications. They might share the same PersistentVolume and replicate data between each other.

DaemonSet defines Pods that provide node-local facilities. These might be fundamental to the operation of your cluster, such as a networking helper tool, or be part of an add-on.

Job and CronJob define tasks that run to completion and then stop. Jobs represent one-off tasks, whereas CronJobs recur according to a schedule.

Networking

Kubernetes networking model makes Pods look like VMs in the networking aspect. Pods on any nodes can communicate with each other without NAT. Containers inside the same Pod share its network meaning that they can reach each other using localhost.

Kubernetes networking addresses four concerns:

- Containers within a Pod use networking to communicate via loopback.
- Cluster networking provides communication between different Pods.
- The Service API lets you expose an application running in Pods to be reachable from outside your cluster.
- Ingress provides extra functionality specifically for exposing HTTP applications, websites and APIs.
- You can also use Services to publish services only for consumption inside your cluster.

Storage

Kubernetes does not ship a particular implementation of storage. However, it provides a range of resources that define the storage concept and supports different types of volumes. A Pod can use any number of volume types simultaneously. Ephemeral volume types have a lifetime of a pod, but persistent volumes exist beyond the lifetime of a pod. When a pod ceases to exist, Kubernetes destroys ephemeral volumes; however, Kubernetes does not destroy persistent volumes. For any kind of volume in a given pod, data is preserved across container restarts.

At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod. How that directory comes to be, the medium that backs it, and the contents of it are determined by the particular volume type used.

PersistentVolumes and PersistentVolumeClaims are the resources kinds most important to understand here.

- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

- A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany). Once and Many here refer to a number of Pods that can perform the read or write simultaneously.

1.2.4 Infrastructure Security

When we are talking about the infrastructure security, we must consider multiple layers. Going from the top to the bottom, we start with the security measurements taken on the Cloud provider side. In most cases this is not something we can affect and the security of different Cloud providers varies significantly. Unfortunately, this is out of scope of this paper, but all of the "big five" Cloud providers (AWS, Azure, Google Cloud, Alibaba and IBM) maintain high security standarts and security risks generally should not be a concern for their end users. Then, we get to the cluster itself. On the cluster level we must consider the security of the nodes, security of the cluster components and their configuration. At this layer we have already some space for the misconfigurations to appear. Here we can evaluate the security of the single components using some of the security scanners presented in Kubernetes security automation. Lastly, we get to the application level, where the security of the application code, Kubernetes resoucre configuration, libraries, dependencies and base images is the main concern. Again, this is the layer, where the developers have the most access to, thus, providing a lot of space for the possibility of a human error. In this paper we mostly work on this level when we do our research.

Officially, The Linux Foundation provides only some security guidelines for each of the layers. However, bundled with Kubernetes we get a few means to keep the security under control. Below we present an overview of the Kubernetes security recommendations, RBAC and Data security inside Kubernetes.

Kubernetes security recommendations

This section gathers the official security recommendations provided by the Kubernetes. They provide a list of concerns for each level of the cloud infrastructure. Cloud infrastructure can be viewed as a composition of four layers as displayed by the Figure 1.3.

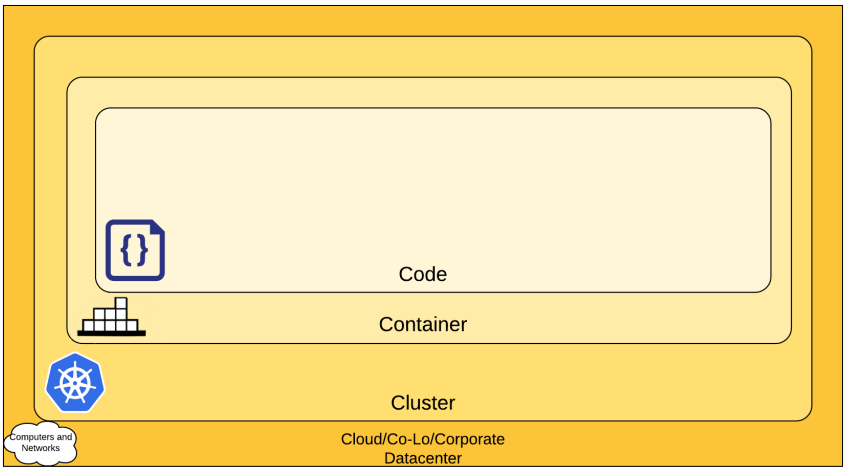


Figure 1.3: Four layers of the cloud infrastructure.

Each layer is built upon the previous one and its security depends on the security of the outer layers. It is, therefore, important to maintain high security standarts on base levels (Cloud, Cluster, Container).

1. **Cloud**

   Each cloud provider has its own security policies and guidelines. There are, however, some general infrastructure-level security best advice described by in the Table 1.1.

Area of Concern for Kubernetes Infrastructure	**Recommendation**
Network access to API Server (Control plane)	All access to the Kubernetes control plane is not allowed publicly on the internet and is controlled by network access control lists restricted to the set of IP addresses needed to administer the cluster.
Network access to Nodes (nodes)	Nodes should be configured to only accept connections (via network access control lists) from the control plane on the specified ports, and accept connections for services in Kubernetes of type NodePort and LoadBalancer. If possible, these nodes should not be exposed on the public internet entirely.
Kubernetes access to Cloud Provider API	Each cloud provider needs to grant a different set of permissions to the Kubernetes control plane and nodes. It is best to provide the cluster with cloud provider access that follows the principle of least privilege for the resources it needs to administer.
Access to etcd	Access to etcd (the datastore of Kubernetes) should be limited to the control plane only. Depending on your configuration, you should attempt to use etcd over TLS.
etcd Encryption	Wherever possible it's a good practice to encrypt all storage at rest, and since etcd holds the state of the entire cluster (including Secrets) its disk should especially be encrypted at rest.

Table 1.1: Security recommendations for the Cloud layer.

2. **Cluster**

   There are two cluster security concerns that could be addressed: securing the configurable cluster components and securing the applications running in the cluster.

   There are a few things to consider regarding the application security:

   - RBAC Authorization (Access to the Kubernetes API)
   - Authentication
   - Application secrets management (and encrypting them in etcd at rest)
   - Ensuring that pods meet defined Pod Security Standards
   - Quality of Service (and Cluster resource management)
   - Network Policies

- TLS for Kubernetes Ingress

3. **Container**
   Securing containers is a vast topic, which deserves its own chapter. There are, nevertheless, a few general recommendation provided by the Kubernetes, which you can find in the Table 1.2.

Area of Concern for Containers	**Recommendation**
Container Vulnerability Scanning and OS Dependency Security	As part of an image build step, you should scan your containers for known vulnerabilities.
Image Signing and Enforcement	Sign container images to maintain a system of trust for the content of your containers.
Disallow privileged users	When constructing containers, create users inside of the containers that have the least level of operating system privilege necessary in order to carry out the goal of the container.

Table 1.2: Security recommendations for the Container layer.

4. **Code**
   When it comes to code, the developers have the most flexibility to design secure applications. There are a lot of issues to address, which may vary significantly from application to application depending on its purpose, architecture and framework base. Kubernetes documentation gives a handful of recommendations regarding this topic, which are displayed below in the Table 1.3.

Area of Concern for Code	Recommendation
Access over TLS only	If your code needs to communicate by TCP, perform a TLS handshake with the client ahead of time. With the exception of a few cases, encrypt everything in transit. Going one step further, it's a good idea to encrypt network traffic between services. This can be done through a process known as mutual TLS authentication or mTLS which performs a two sided verification of communication between two certificate holding services.
Limiting port ranges of communication	This recommendation may be a bit self-explanatory, but wherever possible you should only expose the ports on your service that are absolutely essential for communication or metric gathering.
3rd Party Dependency Security	It is a good practice to regularly scan your application's third party libraries for known security vulnerabilities. Each programming language has a tool for performing this check automatically.
Static Code Analysis	Most languages provide a way for a snippet of code to be analyzed for any potentially unsafe coding practices. Whenever possible you should perform checks using automated tooling that can scan codebases for common security errors.
Dynamic probing attacks	There are a few automated tools that you can run against your service to try some of the well known service attacks. These include SQL injection, CSRF, and XSS. One of the most popular dynamic analysis tools is the OWASP Zed Attack proxy tool.

Table 1.3: Security recommendations for the Code layer.

Role Based Access Control

Data security

1.2.5 Other Kubernetes Implementations

Openshift

Rancher Desktop

1.3 Kubernetes security automation

This section introduces the reader to the topic of the security automation inside the Kubernetes cluster. We discuss different security tools, their place in the Cloud Infrastructure and examine their usage patterns. Additionally, we explain why were the

specific tools chosen for our research.

1.3.1 Overview

1.3.2 Selection criteria

To perform our assessment we have chosen from a variety of Kubernetes security scanners. Though the area is still relatively new, there is a variety of tools with different purposes available on the market. We made our choice based on the following criteria:

- **free-to-use**
  We do include some proprietary tool testing further in our research as an additional comparison, however, for the main part we only use free tools. Kubernetes itself is distributed under Apache License 2.0, which means it is inherently free to use. The ability to adopt these tools without financial constraints enables wider adoption, thus, contributing to the community-driven innovations.

- **open-source**
  Again, we are sticking to the open-source nature of the Kubernetes. By selecting open-source tools, this research ensures that each tool's codebase is transparent and can be reviewed by security experts. This transparency increases trust in the tools' effectiveness, as the community can spot, disclose, and even patch any vulnerabilities in the software. Another advantage is the customization of the open-source software as the companies can adapt the tools to their specific Kubernetes security needs.

- **designed specifically for Kubernetes**
  Designed to be used in the Kubernetes environment specifically, these tools should offer features like scanning container images for vulnerabilities, but also monitoring network policies, securing Kubernetes configuration files, and identifying misconfigurations within clusters. Tools built specifically for Kubernetes are more efficient, as they are optimized to address the distinct aspects of the platform, making security management more effective.

- **has an active community support**
  Tools with active communities tend to have more frequent updates, faster response times for bug fixes, and a wide range of contributors who bring diverse insights to improve functionality and security. The world of the software security is changing rapidly and an active community means that the tool is up-to-date with the most recent events. A thriving community also means that users can easily access support on the community forums.

Based on the aforementioned criteria we ended up choosing and testing the following tools:

- Trivy
- kube-bench
- kube-hunter
- Kubescape

In the next chapters we closely examine each selected tool and explain how it is matches our selection criteria. Additionally, we compare them to each other and highlight their strong and weak sides.

1.3.3 Trivy

1.3.4 Kube-bench

1.3.5 Kube-hunter

1.3.6 Kubescape

Bibliography

[1] The Linux Foundation. Kubernetes official web. https://kubernetes.io/, Accessed: 4.11.2024.

[2] Ian Smalley (IBM) Stephanie Susnjara (IBM). What is containerization? https://www.ibm.com/topics/containerization, Accessed: 4.11.2024.

[3] Ian Smalley (IBM) Stephanie Susnjara (IBM). What is kubernetes? https://www.ibm.com/topics/kubernetes, Accessed: 4.11.2024.