Muftau Raheem

# IMPLEMENTING A SECURED CONTAINER WORKLOAD IN THE CLOUD

Master's thesis

Master of Engineering, Cybersecurity

2021

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland
University of Applied Sciences

| Author (authors) | Degree title | Time |
|---|---|---|
| Muftau Raheem | Master of Engineering, Cybersecurity | February 2021 |

**Thesis title**

Implementing a Secured Container Workload in the Cloud

69 pages
1 page of appendix

**Commissioned by**

**Supervisor**

Vesa Kankare (Senior Lecturer, XAMK)

**Abstract**

The objective of this thesis was to research the different security controls to consider when implementing container workloads in the cloud, the current threats affecting container workloads in the cloud, the mitigation to the threats and finally find the cloud security recommendations to secure the workloads.

The action-based research methodology was used, where the researcher is actively involved in ensuring that the commissioner is aware of the risks associated with implementing the container workloads in the cloud and implementing the recommended guidelines presented in this study to mitigate the risks. Information security guidelines, benchmarks, scientific and industry whitepapers have been used in the research.

The research shows that securing of container workloads must be with a “shift left” approach in which security is included at every stage of the container lifecycle. A set of threats and associated mitigations were identified to be critical to securely deploy container workloads in the cloud. This research also examined the different hardening techniques of the k8s cloud-managed service and the applicable security standards for organisations to consider when deploying container workloads in the cloud. The result of this thesis will serve as a guideline for the commissioner and other organisations planning to implement a secured container workload in the cloud.

**Keywords**

Containers, Cloud, Container Security, Docker, Container images, immutable infrastructure, Kubernetes, Registries, Micro-segmentation, Shift-left, Sidecar

**CONTENTS**

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ACR | Azure Container Registry |
| AKS | Azure Kubernetes Service |
| API | Application Programmable Interface |
| ASM | Anthos Service Mesh |
| AWS | Amazon Web Services |
| C2 | Command and Control |
| CCAT | Cloud container attack tool |
| CCM | Cloud controller manager |
| CD | Continuous Deployment |
| CI | Continuous Integration |
| CIS | Centre for information security |
| CLI | Command Line Interface |
| CM | Configuration Management |
| CMEK | Customer-Managed Encryption Keys |
| CMS | Container Management Solution |
| CNI | Container Networking interface |
| COS | Container-Optimized OS |
| CRUD | Create, Read, Update, Delete |
| CS | Continuous Security |
| CSA | Cloud Security Alliance |
| CSO | Container-specific OS |
| CSP | Cloud Service Provider |
| CSPM | Cloud Security Posture Management |
| CVE | Common Vulnerabilities and Exposures |
| CWP | Cloud Workload Protection |
| DAP | Dynamic Access Provider |
| DB | Database |
| DevOps | Developer and Operations |
| DNS | Domain Name Service |
| DR | Disaster Recovery |
| ECR | Elastic Container Registry |

| | |
|---|---|
| EKS | Elastic Kubernetes Service |
| GCP | Google Cloud Platform |
| GCR | Goole Container Registry |
| GKE | Google Kubernetes Engine |
| HA | High Availability |
| IAM | Identity and access management |
| IBMS | Identity based micro-segmentation |
| IP | Internet Protocol |
| K8s | Kubernetes |
| MiTM | Man in the Middle |
| mTLS | Mutual TLS |
| NVD | National vulnerability database |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| OWASP | Open Web Application Security Project |
| PaaS | Platform as a Service |
| PCI | Payment Card Industry |
| PE | Privilege Escalation |
| PLP | Principle of least privilege |
| RBAC | Role-Based access control |
| RO | Read-only |
| SA | Service Accounts |
| SCM | Source Code Management |
| SDK | Software Development Kit |
| SIEM | Security Information and Event Management |
| SM | Service Mesh |
| TLS | Transport Layer Security |
| UEBA | User and entity behaviour analytics |
| VM | Virtual Machines |
| VS | Vulnerability and Compliance |

# 1 INTRODUCTION

Computing architecture has transitioned across different stages in today's modern computing infrastructure. The first of the stages being considered was client-server architecture which is based on a single application in a host operating system (OS) running on bare metal, the next transition was the virtualization architecture which is mainly comprised of the foundational bare metal with a hypervisor, guest OS and the running applications. (Palo-Alto. 2019).

An individual virtual machine running in a virtualized architecture requires its OS, libraries, dedicated resources, and applications, this, in turn, creates a bottleneck for VMs allowed to run on a server. (Sultan et al. 2019). The third stage which is the focus of these studies is the containers, containers are made up of bare metals, host OS, container engines, binaries, or libraries and finally the code application. Containers provide a means to bundle an application's code with the needed dependencies to run smoothly on any platform and computing environment. It, therefore, solves the portability problem by ensuring that the applications can run successfully as the packaged application is passed from one environment to another.

By 2022, 75 percent of global organisations will have containerised applications within the production infrastructure compared to less than 20 percent in 2019. Furthermore, the container management industry is expected to grow from €383.58million in 2020 to €777.37million by 2024 and the public cloud container orchestration services will be having the largest share of this revenue. (Moore 2020). Hence, it becomes critical to have these studies about securing the container workloads in the Cloud. Although there are numerous benefits in embracing containers and moving away from the virtualization architecture, the associated threats become a major challenge that requires adequate consideration. Security controls must be integrated to mitigate the threats throughout the lifecycle of the container.

# 2 RESEARCH

## 2.1 Research Objectives

The identity of the commissioning organization for this thesis has been kept secret and therefore referred to CompX. CompX is undergoing a digital transformation of migrating legacy applications to containerized microservices deployed in the Cloud. CompX operates in a highly regulated business area where it must be compliant with several IT security standards and national regulations.

Container technology is new to the commissioner and this study presents the opportunity to research into the security controls to be implemented when deploying container workloads in the cloud. The digital transformation is not unique to CompX because several organisations are re-architecting applications utilizing the cloud-native services for scaling, high availability, portability, consistency, and the immutability of a containerized microservice infrastructure using DevOps agile deployment model.

The main objective of this thesis is to research and identify the security controls required to securely deploy container workloads in the cloud using a cloud-managed orchestrator solution such as the Elastic Kubernetes Service (EKS) from Amazon, Google Kubernetes Engine (GKE) from Google and Azure Kubernetes Service (AKS) from Azure. The research will identify the controls to secure the containers at every stage of its lifecycle and identify the threats commonly used by adversaries to target containerized applications in the cloud. The result of this thesis will serve as a set of recommended guidelines for the commissioner to securely implement container workloads in the cloud.

## 2.2 Research Questions

In other to achieve the objectives of this thesis with better clarity, the research questions which this thesis aims to answer as listed below.

- What security controls should be implemented to secure container workloads in the cloud?

- What is the current threat landscape of containers and how can they be mitigated?
- What cloud security controls can be utilized to secure the container workloads deployed in the cloud?

## 2.3 Research Method

The action research method has been used in these studies. The focus was on “problem-solving in whatever way is appropriate” (Smith 2017). It involves finding the current practice, researching the available standards and adopt the recommended practices to the environment. This involves active cooperation between the researcher and the commissioning organisation. Also, this method is primarily achieved as “learning by doing” (O'Brien 2001).

The action research is one of the three research synthesised when qualitative and quantitative research methodologies are combined. The uniqueness of the action research is that the researcher participates in the change process and ensure the realization of the research. The researcher primarily acts as equipment for gathering data. (Kananen 2015, 57).

The data gathering method in action research is qualitative research which involves participatory observation allowing the researcher to be the subject phenomenon. The observation is complemented by interviews and discussions to ensure the correct interpretation of the observations. (Kananen 2015, 57). The researcher has found this method effective with the commissioner’s stakeholders in ensuring that the expected changes, as found during this research, are implemented.

## 3 APPLICATION CONTAINER TECHNOLOGY

The application container technology is OS-level virtualization for deploying and running microservice applications without utilizing an entire VM compute resource. Containers are likened to an application running as a process with isolation on an OS within its own address space. The containerised running applications are packaged with all the needed dependencies and libraries. (Zhang et al. 2018).

As shown in figure 1, the traditional deployment involves the running of applications on physical servers. All the applications share the same resource with no means of resource allocation for the applications. Solving the resource allocation issue requires allocating a dedicated physical server for an individual app. This poses a scalability challenge and resource wastage with more cost. (Kubernetes.io 2020b).

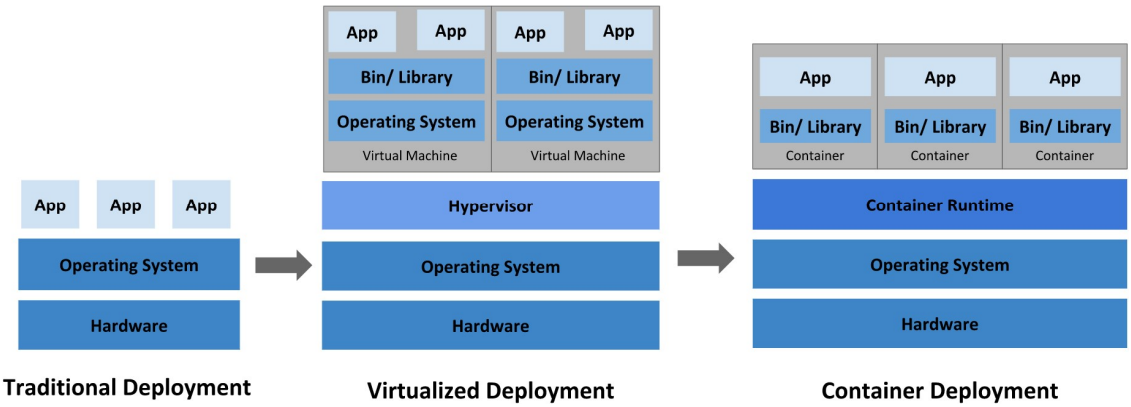


Figure 1 Workload Abstractions (Kubernetes.io 2020b)

The virtualized infrastructure evolved as a solution to the challenges of the traditional deployment by allowing multiple virtual machines (VMs) with allocated compute resources to run on a physical server, thereby providing better resource utilization and scalability. Each VM has its OS and provides some security by isolating applications between the VMs without allowing easy access to information. (Kubernetes.io 2020). The use of containers provides some benefits to infrastructures such as agile application deployment with CI/CD, resource

isolation with predictable application performance and application consistency across environments.

Containers are lightweight and portable because the applications share a single OS as well as decoupled from the base infrastructure making them interoperable across different OS and Cloud environment. (Kubernetes.io 2020b). Multiple containers run on the same physical server using features such as Linux Control Groups (Cgroups) and namespaces. As shown in Figure 2, Cgroups are used to assign resources such as CPU, memory, and network to the containers. (Zhang et al. 2018).

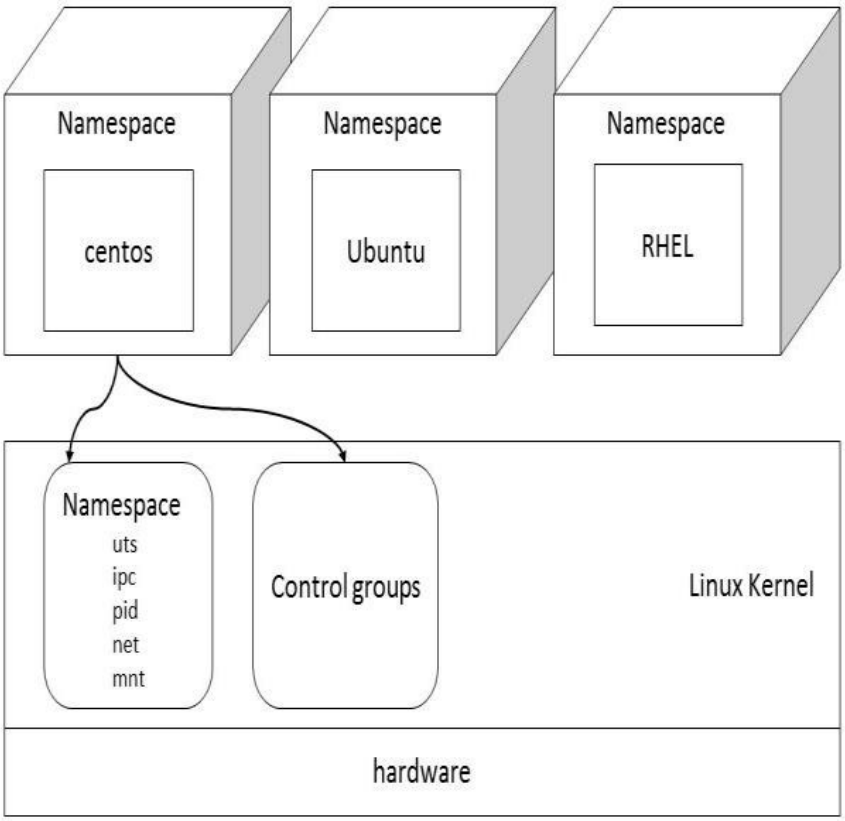


Figure 2 Namespaces and Cgroups (Joy. 2015).

Figure 3 shows the granularity of application workloads with a shorter lifespan as they evolve from one deployment model to another. Containers have the shortest lifespan that supports the DevOps culture of several deployment iterations in a day. (MacDonald & Croll 2020). There are different types of container

technologies apart from Docker that includes java containers, Unikernels, LXD, OpenVZ, Rocket containers (RKT), Hyper-V containers (Wadsworth. 2016) and many more. This focus of this thesis is on Docker and using a cloud-managed container orchestrator service, Kubernetes.

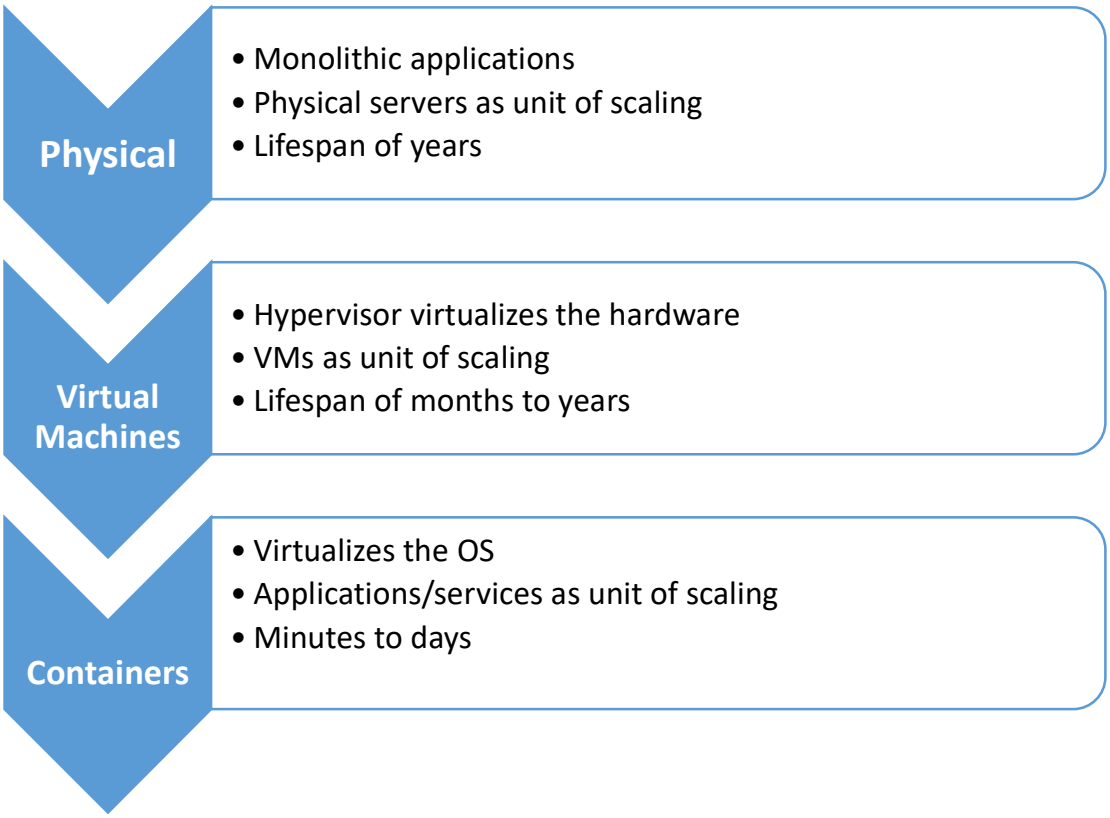


Figure 3 Lifespan of application workloads (MacDonald & Croll. 2020).

### 3.1 Container Technology Terminologies

Containers continue to be a popular technology among application developers due to its robustness in building and packaging an application with its dependencies suitable for different environment and deployment targets (Johnston 2018). It is important to understand the foundational elements of containers to ensure adequate security controls are applied at every stage of the container lifecycle. A summary of some of the terminologies is presented in subsequent sections of this chapter.

### 3.1.1 Image

Container images are the lightweight foundational element of the containers because they are the files with the needed configurations, libraries, and the code to efficiently run an application with the desired result. The containers become instances of the images and every instance will be having same foundational dependencies (Brady et al. 2020). The docker images are composed of different image layers, with each layer depicting a set of instruction in the docker file. All entries in the file are read-only (RO) except the last line which usually signifies what command to run in the container layer. (Docker 2020a). Figure 4 below shows the layered composition of a container image.

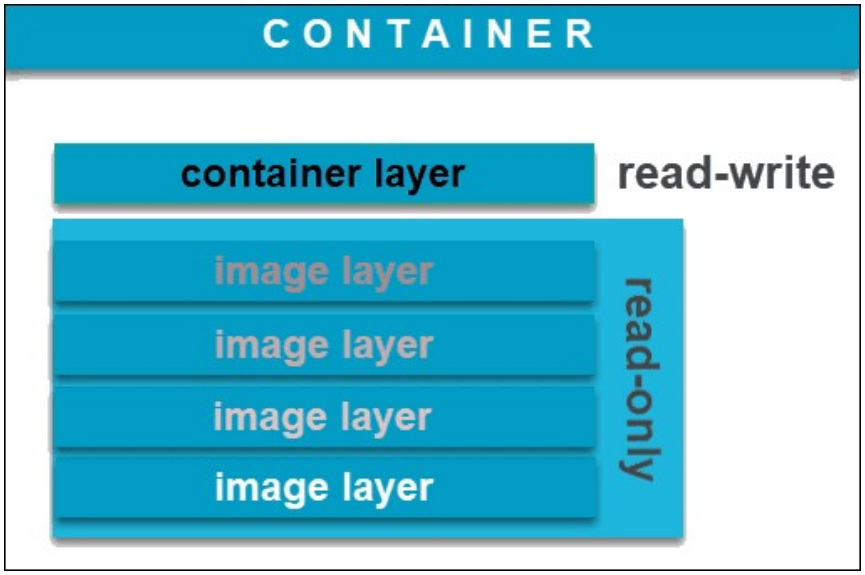


Figure 4 Container Image Layers (Simi 2019)

### 3.1.2 Docker

Dockers are the de-facto Platform as a Service (PaaS) solution for rapidly building, testing, deploying, and sharing containers (Brady et al. 2020). It is the default container runtime on some cloud service providers (CSP) managed Kubernetes offerings which include Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS) and Microsoft Azure Kubernetes Service (AKS) (Foster 2020). In summary, Figure 5 shows docker supports the creation of containers while orchestrators like Kubernetes manages the containers

(Bytemark 2019). Other orchestrators include apache mesos, docker swarm, fleet and docker-compose (Pavlik & Mercl 2018).

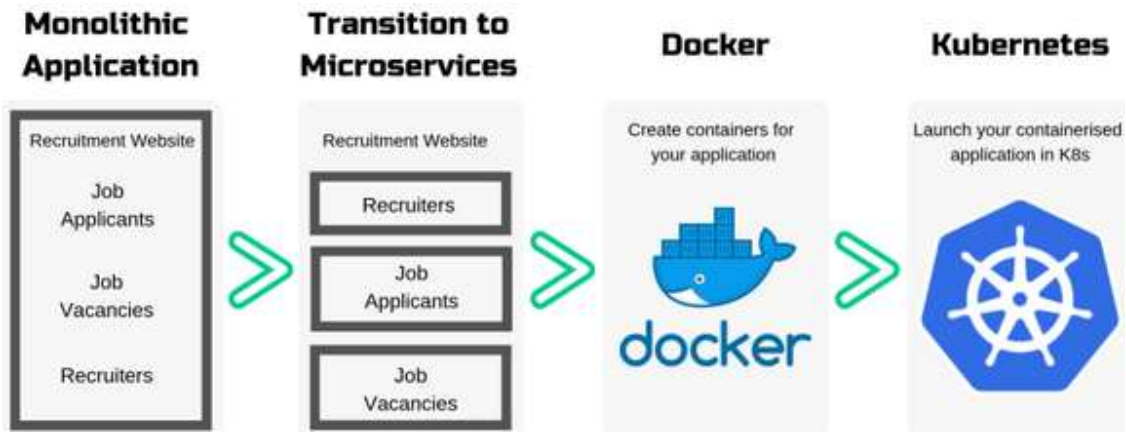


Figure 5 Relationship between Dockers and Kubernetes (Bytemark 2019)

### 3.1.3 Kubernetes

Kubernetes sometimes referred to as k8s is the container management solution (CMS) used for the orchestration of Docker containers across multi-host installations. (RedHat 2018). It supports container-as-a-service by abstracting the application orchestration from underlining infrastructure resource and as-a-service automation helping with provisioning, scaling, and auto-healing. Kubernetes is constantly being developed with new features and critical patches for discovered vulnerabilities.

Kubernetes is becoming the container orchestration standard due to its interoperability and its command line is the current industry standard which is *kubectl*. K8s nodes have a container runtime software that runs the container applications using either Docker, containerd and container runtime interface. (Kubernetes.io 2020a). “The most powerful orchestrator is Kubernetes in this time. Kubernetes allows you to run and manage containers, regardless of the hardware” (Pavlik & Mercl 2018).

Kubernetes has some standard components distributed between the master and worker nodes. These components include the API server, controller manager, scheduler, kube-dns, metric-server, etcd, kubelet, and kube proxy.

Within a cloud infrastructure, some components are managed by the CSP and some are managed by the CSP customers, the details of the shared responsibility model are presented in section 4 of container security in the cloud. Figure 6 below contains some of the different elements of a container workload.

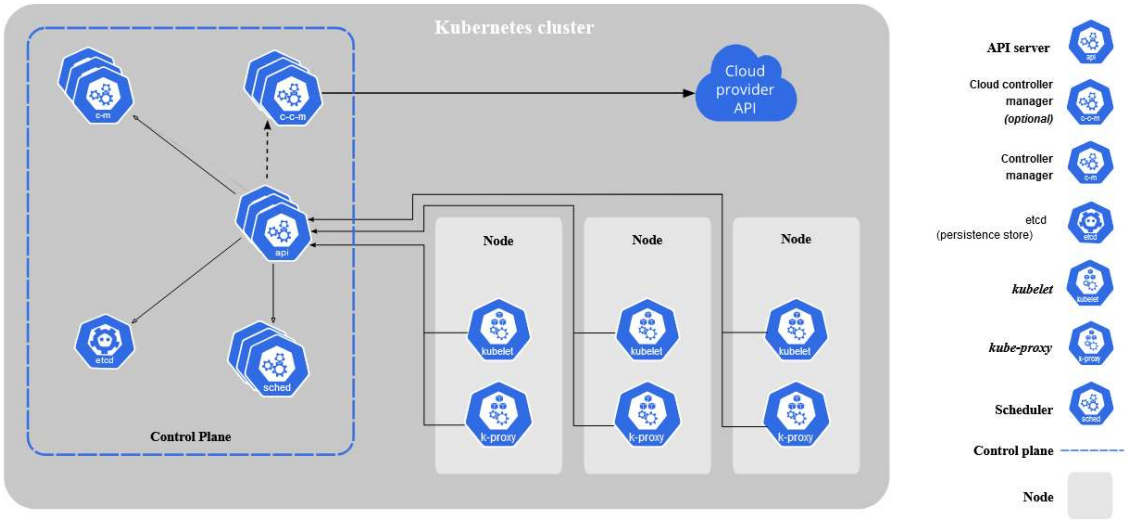


Figure 6 Kubernetes Standard components (Kubernetes.io 2020a)

The functions of the respective components as applicable to the master and worker nodes is presented in table 1.

Table 1 Kubernetes Components on Master and Worker nodes

| Component | Master Node functions | Worker Node functions |
|---|---|---|
| **API Server** | • Processes requests and updates etcd.<br>• Performs authentication and authorization.<br>• It provides the entry point for the cluster. | Not Applicable |

| | | |
|---|---|---|
| **Controller Manager** | • It Is the daemon process implementing the control loops built into Kubernetes such as deployment updates. | Not Applicable |
| **Cloud Controller Manager (CCM)** | • The CCM enables linking the K8s with the CSP's API by logically separating the components that interact with the cloud from those that only require interaction with the k8s cluster | Not Applicable |
| **etcd** | • It is a HA key-value store for the Kubernetes cluster | Not Applicable |
| **Scheduler** | • It decides where pods are run based on pre-defined properties e.g affinity groups, resources, labels etc. | Not Applicable |
| **Kubelet** | Not Applicable | • This is an agent on every worker node.<br>• Ensures that all pods are healthy.<br>• It registers the node with the API server. |
| **Kube Proxy** | Not Applicable | • This is also an agent on the workers.<br>• It acts as a network proxy and load balancer for k8s services. |

### 3.1.4 Kubernetes Constructs and Objects

These are entities used to depict the state of the cluster and can be expressed in the YAML format. The objects can describe what the container applications are running, the nodes where the applications are running, the assigned resources to the containers and the configured policies such as high availability and restart policies (Kubernetes.io 2020a). The construct includes Pods, nodes, cluster, services, and namespaces.

In Kubernetes, the pods are the smallest units that exist. It represents an instance of a running process within a cluster. A pod has one or more container and the containers within a pod automatically communicate because they share network and storage resource irrespective of their nodes (Kubernetes.io 2020a).

Nodes are the elastic compute resources that run the containerized applications. The nodes host the pods, the pods can migrate to an available node to ensure the application is responsive. The nodes are managed from a cluster, and each node runs the needed services to support the Docker containers for the cluster. Kubernetes has a node master-slave architecture; the slave is otherwise referred to as the worker node (Bytemark 2019).

The master nodes in a Kubernetes cluster controls the pod deployments and worker nodes. (Bytemark 2019). It maintains the state of the running application and the container images. The worker nodes are the actual compute resources that host the deployed pods. Clusters are sets of nodes running the pods. For every cluster, there is a master node and one or more worker nodes. The containers within a Kubernetes cluster are abstracted across the cluster and not bound to a specific node (RedHata).

Services are an essential component of every pod. It is the abstract way of exposing the running applications on a set of pods (Kubernetes.io 2020a). Kubernetes namespaces are the logical groupings of the cluster resources that act as a virtual cluster within a Kubernetes cluster.

The list below shows the available namespaces from a newly created Kubernetes cluster. Figure 7 and 8 show different K8s objects.

```
# kubectl get namespaces
NAME              STATUS   AGE
default           Active   3m3s
kube-node-lease   Active   3m4s
kube-public       Active   3m4s
kube-system       Active   3m4s
```

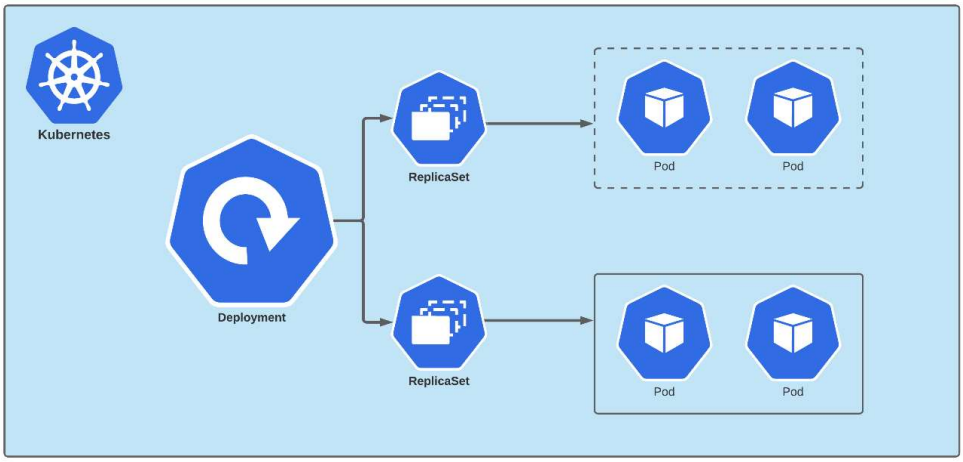


Figure 7. k8s objects (Maharjan 2020)

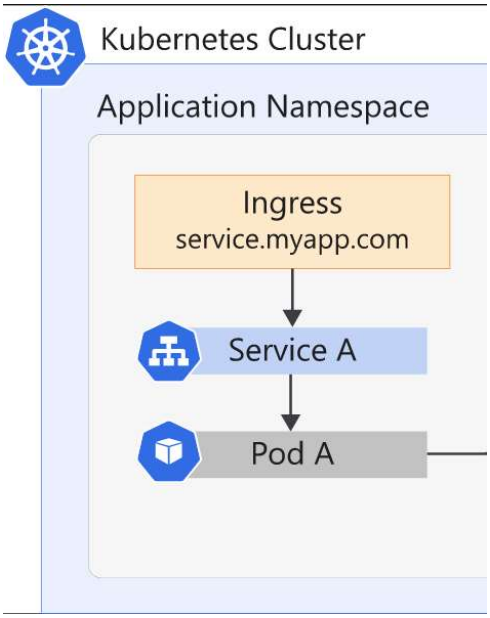


Figure 8. k8s objects (Lee & Hogenson 2020).

# 4 CONTAINER SECURITY IN THE CLOUD

Deploying containers in the cloud using the CSP's managed service requires the understanding of the shared responsibility model of the cloud infrastructure. The containers managed service is a PaaS solution which means that the CSP is responsible to secure the underlying infrastructure and the customer secures the running application. Despite the CSP ensuring the security of the infrastructure, there is still a lot of security controls that need to be managed by the application owner when running applications in the cloud. Some of these include running the latest Kubernetes version, latest operating system versions, securing the container images, ensuring secure image storage/access in the registry, container network security policies, container runtime security, monitoring and vulnerability management.

A shared responsibility model of the Kubernetes cloud-managed service such as GKE, EKS and AKS is represented in figure 9, showing that all components of the control plane are the full responsibility of the CSP. The organisations consuming the CMS must ensure the security of the pods throughout their lifecycle.

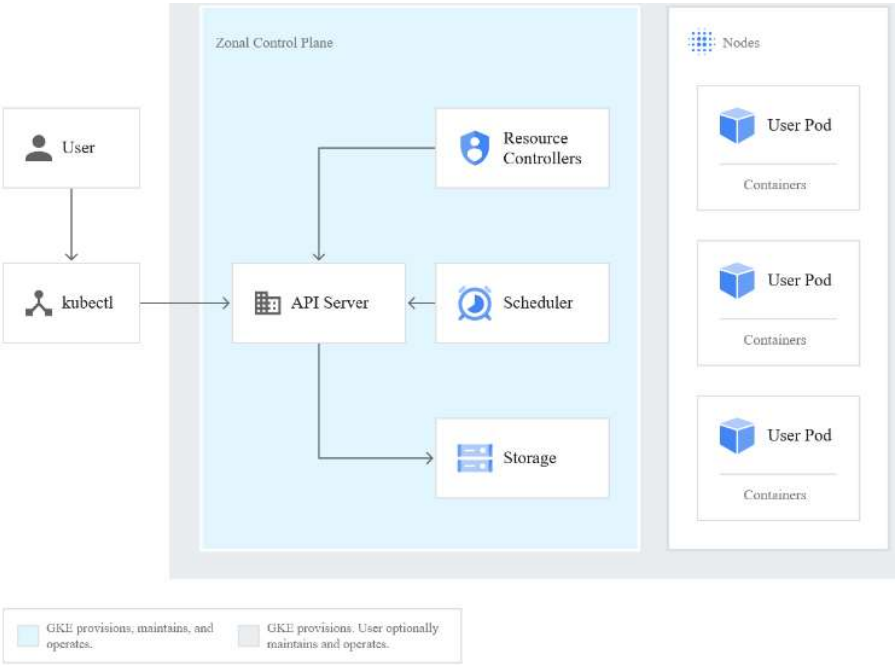


Figure 9. Share responsibility model of the k8s CMS (Google 2020c)

The table 2 below shows the shared responsibility model of a container-managed service depicting the underlying infrastructure being the responsibility of the CSP.

Table 2 Shared responsibility model of CMS in the cloud (Kaczorowski 2019)

| Container PaaS Layer | Customer Responsibility | CSP Responsibility |
|---|---|---|
| Content | * | NA |
| Access policies | * | NA |
| Usage | * | NA |
| Deployment | * | NA |
| Web application security | * | NA |
| Identity | NA | * |
| Operations | NA | * |
| Access and Authentication | NA | * |
| Network Security | NA | * |
| Guest OS | NA | * |
| Audit Logging | NA | * |
| Network | NA | * |
| Storage with encryption | NA | * |
| Hardened Kernel | NA | * |
| Boot | NA | * |
| Hardware | NA | |

### 4.1 Container Images security

The containers images are the core component in the containerized web application, as such the integrity of the image must be protected by ensuring that non-vulnerable and exploitable images are built, stored, and deployed within the

infrastructure. Securing the container images means reducing the attack surface in the container lifecycle by detecting vulnerabilities, configuration errors and the security policy violations (StackRox 2019). Figure 10 shows the different security threats associated with the building of the base images from the source code to the deployment.

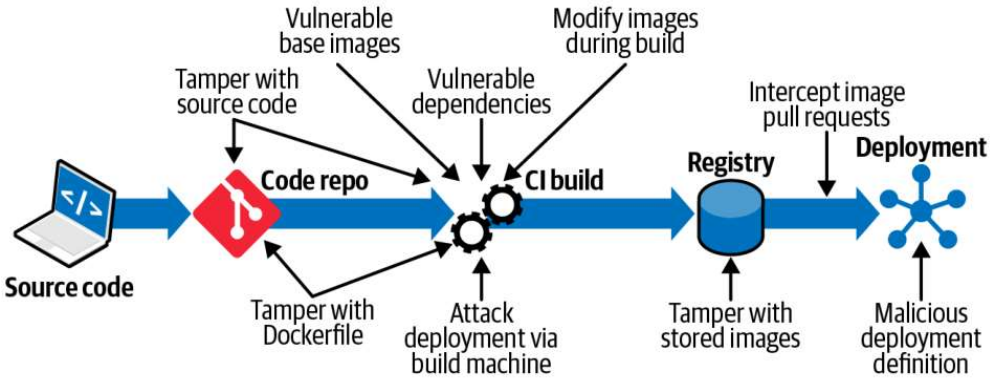


Figure 10. Base Image security threats (Rice 2020)

The items discussed in this section presents the security recommendation towards having a secured image for container workloads and the implementation of these controls will help mitigate threats.

### 4.1.1 The building of Base Images

With the availability of different images from the public registries, it is easier to "grab" images from publicly available registries. This poses a immense risk to the organization as the actual origin of the image and its dependencies are not known, likewise if there are exploits embedded in the image. It is recommended to have full control of the build process by building the image from their private registry (Bernstein 2018).

Securing the build of the base images from the early phase creates a "shift left" approach towards securing the container workloads. The base image is often built from a set of commands specified in the Dockerfile, it is often a common practice to use the "docker build" command which in turn invokes the docker daemon process running as root and grants the possibilities of running other

docker privileged commands which could be a security risk within an uncontrolled cloud infrastructure. The latest release of Docker, version 19.03.14 in December 2020, Docker introduces the rootless mode of the docker daemon that allows a non-root user to execute the Docker daemon and containers inside the user's namespace (Docker 2020b). Using the rootless mode helps to satisfy the CIS docker recommendation of ensuring the images are created with a non-root user and the containers running as a non-root user (CIS Docker 2019).

However, there are still security risk of being able to execute other Docker commands which are not relevant to the image build process, this presents an opportunity for a malicious user or an attacker to poison the trust chain of the container lifecycle. Other solutions such as Kaniko, Bazel, podman and buildah (Abbassi 2019) gives the flexibility of creating an image for container workloads without using the Docker daemon process (Rice 2020)

The image build process should include a security assessment that identifies vulnerabilities in the image components and every layer of the base image. Introducing this control ensures that the identified vulnerabilities and available fixes are applied before the base image is marked as a container "golden" image for the workloads (StackRox 2019).

It should however be noted that traditional vulnerability management tools are limited in terms of visibility to identify the vulnerabilities of each layer of the container images. Some of these limitations include asset management, keeping up with the ephemeral nature of the containers, providing vulnerability details of the libraries being used, e.g., identifying the vulnerability in a webserver but not an underlining library composition. Some open-source solutions capable of a deep dive to provide the vulnerability of the container images and configuration include Anchore engine, Clair project and Dagda (Moyle 2020).

### 4.1.2 CI/CD/CS Pipeline

Automation is a key part of the container deployment in the Cloud and so should be the security. The continuous integration (CI) involves the continuous

monitoring of the source code management (SCM) for new commits or changes and a set of predefined tasks are initiated based on the commit. In continuous delivery, Infrastructure as code images is built, tested, and deployed. The end goal of the continuous delivery is a status showing the image could be deployed by storing it in the registry (Sanz et al. 2018.)

The continuous deployment (CD) uses the result of the continuous delivery to trigger an automated deployment and capable of a rollback operation based on a configuration management integration. RBAC must be implemented to ensure authorized accounts can trigger a job on the pipeline (Sanz et al. 2018).

The continuous security (CS) should be dynamic and keep up with the rapidness of the CI/CD and its threat landscape. The pipeline must include admission controls that validate the built images, stored images, and images that are deployed meet the expected compliance benchmarks such as the CIS standard for Docker or the benchmark for k8s CMS as well as an acceptable level of risk (Sanz et al. 2018).

The continuous security must implement an admission control that checks the images before being transformed into a running container, the admission control should include the following at the least:

- image scanning for vulnerabilities and malware
- ensuring that image is pulled from an authorized registry.
- checking that the image meets the defined security policies.
- trusted images should only be used. (Rice 2020)

With the admission control in place, the pass or fail control could then include in the CI/CD pipeline for an automated CS. Some of these include:

- A failed vulnerability and compliance (VC) scan would result in a failed build.
- The image failed VS scan should prevent image deployment to workloads.
- A failed VS scan on container runtime should alert for remediation.

(Hausenblas & Rice 2018)

An example security policy rule would be to fail a build pipeline if there are high and critical VC issues that already have a fix. Different policies could be applied

based on the criticality of the infrastructure, for example, the development environment having a different security policy from the production environment.

Figure 11 shows the integration of an automated security assessment to the CI/CD pipeline which includes applying security policies during the image build, storage and towards the runtime. Anchore engine is an example open-source tool that could be integrated with the popular CD tool, Jenkins. (Jenkins 2018). On the commercial side, Prisma Cloud from Palo-Alto among others can also be used to achieve this implementation.

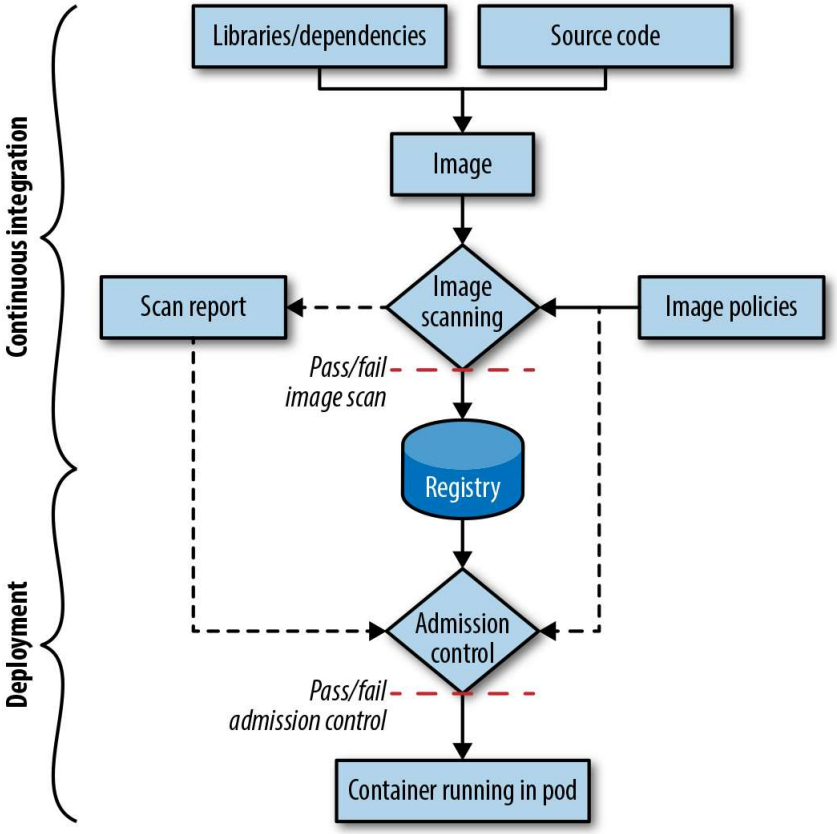


Figure 11. CI/CD Pipeline with automated security data flow. (Hausenblas & Rice 2018)

Figure 12 below shows an example configuration of a continuous security policy for an acceptable vulnerability risk level for container images using Prisma Cloud

compute module.  On the other hand, figure 13 shows the acceptable compliance risk level. The VC policies will fail or pass the build if the set threshold is met.


Figure 12. CI/CD pipeline vulnerability Security policy

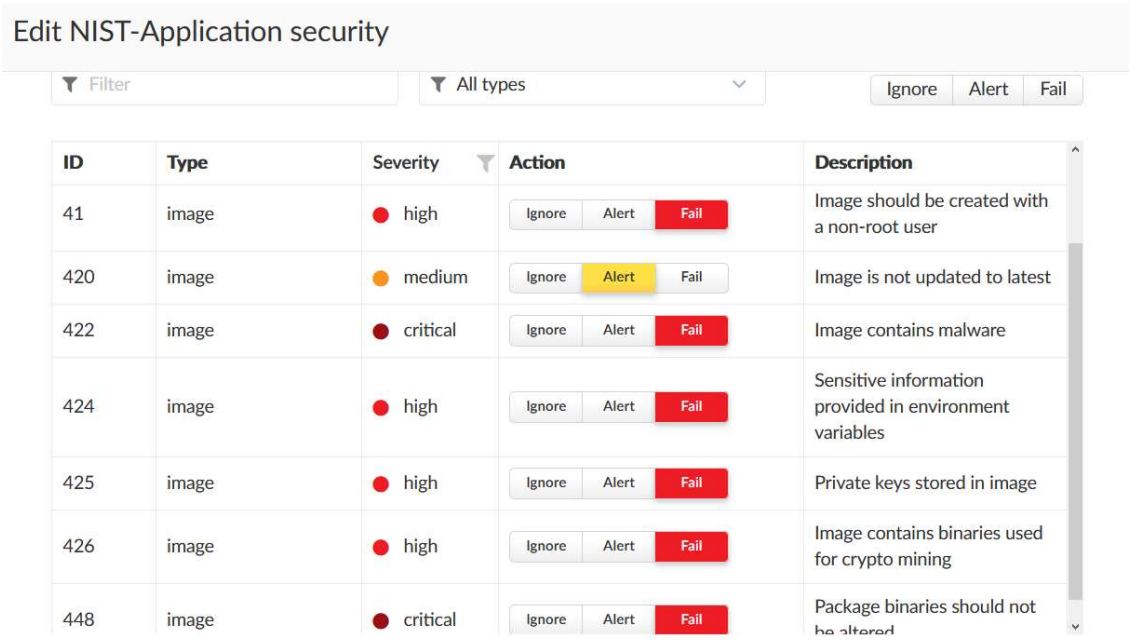


Figure 13. CI/CD Compliance Security policy using Prisma Cloud

Figure 14 shows a summary output of the pipeline from Jenkins. Build #9 failed because there were 2 compliance issues of high and medium severity. The

security violations show that the image was created with a root user.  Build #8 in the screenshot passed when the policy was adjusted to alert about the policy violations and pass the build.


Figure 14 Jenkins Image build output

The below output shows the detailed output of the scan and figure 15 shows the identified vulnerabilities from the image. The full pipeline code is in appendix 1.

```
Started by user user
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /opt/bitnami/jenkins/jenkins_home/workspace/prisma_pi
peline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Preparation)
[Pipeline] echo
Preparing
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {(Build)
[Pipeline] sh
+ echo FROM ubuntu:14.04
[Pipeline] sh
+ echo MAINTAINER AF <af@*.com>
[Pipeline] sh
+ echo RUN mkdir -p /tmp/test/dir
[Pipeline] sh
+ docker build --no-cache -t dev/ubun2:test .
```

```
Sending build context to Docker daemon   85.5kB

Step 1/3 : FROM ubuntu:14.04
 ---> df043b4f0cf1
Step 2/3 : MAINTAINER AF <af@*.com>
 ---> Running in ff29b61c1043
Removing intermediate container ff29b61c1043
 ---> b7b94befd51c
Step 3/3 : RUN mkdir -p /tmp/test/dir
 ---> Running in b01ec9b2cf08
Removing intermediate container b01ec9b2cf08
 ---> ac96d3384183
Successfully built ac96d3384183
Successfully tagged dev/ubun2:test
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {(Scan)
[Pipeline] prismaCloudScanImage
[PRISMACLOUD] Scanning images on master
[PRISMACLOUD] Waiting for scanner to complete
[PRISMACLOUD] /opt/bitnami/jenkins/jenkins_home/workspace/prisma_pipeline/t
wistcli4569411900994248309 images scan ubun* --docker-
address unix:///var/run/docker.sock --ci --publish --details --
address https://*.twistlock.com:443/121212 --ci-results-file prisma-cloud-
scan-results.json
[prisma_pipeline] $ /opt/bitnami/jenkins/jenkins_home/workspace/prisma_pipe
line/twistcli4569411900994248309 images scan ubun* --docker-
address unix:///var/run/docker.sock --ci --publish --details --
address https://*.twistlock.com:443/121212 --ci-results-file prisma-cloud-
scan-results.json
[Pipeline]}
[Pipeline] // stage
[Pipeline]}
[Pipeline] // node
[Pipeline] End of Pipeline
[Checks API] No suitable checks publisher found.
ERROR: Build failed
Finished: FAILURE
```

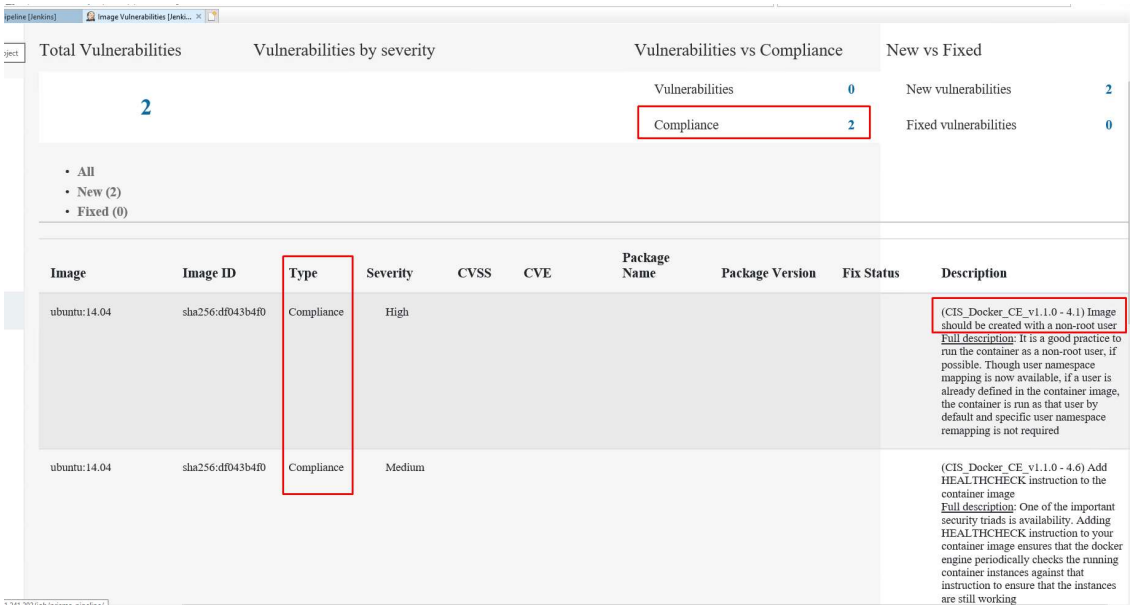


Figure 15. Image build security policy violations on Jenkins

## 4.2 Securing Container Registry

Images are stored in registries. To this thesis, the registries being considered are those of the Cloud providers such as the Google container registry (GCR), Amazon Elastic container registry (ECR) or the Azure container registry (ACR). The storage and retrieval of the images in the registry are referred to as pushing and pulling, respectively.

### 4.2.1 Container Registry Authentication and Authorisation

Granting access to the container registry should follow a least-privilege, administrative accounts with write as well as delete roles should not be used for regular tasks on the registry. The individual identity entities building the images and pushing to the registry will be assigned credentials with the permissions to push/pull to the registry, the identity credential should have short-lived token access for few hours according to the organizational policies.

The CI/CD build pipeline should be assigned a service account subject to an access-token rotational policy usually 90 days according to CIS benchmark. This service account would only need the pull permission. Where possible, namespaces should be used to group and share registry resources with relevant

teams within the project (Field et al. 2018). Private registries must always be used to mitigate the risk of unauthorised access to the built images.

### 4.2.2 Continuous Vulnerability Assessment

As vulnerabilities are constantly discovered and threat actors always ready to exploit, so is the criticality of continuous vulnerability assessment of images in the registry. The vulnerability source at least gets the common vulnerability exposure (CVE) from the national vulnerability database (NVD).

All images pushed to the registry must be scanned for vulnerability upon upload based on the image digest which identifies the image and tracks its vulnerability changes. After the initial assessment, a continuous and regular scan of the image should be scheduled for an updated image vulnerability status based on the threat intelligence from the vulnerability sources.

Most of the Cloud providers provide the container registry vulnerability scanning as a managed service for the cloud-native solution (Google 2020e).
In addition to the scanning of the images, it is recommended to audit the age of the container images stored in the registry. Older images with vulnerable dependencies should be identified, patched, deleted, or recreated using the latest libraries. Ensuring these activities are completed helps to reduce the attack vectors from the Cloud infrastructure.

### 4.2.3 Registry Encryption

The data at rest in the registries are recommended to be encrypted to protect against access to sensitive data on disks, modification, and unauthorized access (Dissanayake & Mistry 2020). Most Cloud providers offer encryption for data at rest which means the CSP completely manages the complete lifecycle of the keys viz, creation, rotation, and deletion of the keys.
Encrypting the container registry using the customer-managed encryption keys (CMEK) allows fulfilling the security and compliance requirement while having control of how the encryption keys are used. Access to the registry can also be

managed with the encryption keys; a registry with disabled encryption key means access is forbidden until allowed. In addition to encrypting the data at rest, data in transit during pull and push activities should also be encrypted between the registries and trusted endpoints.

### 4.2.4 Trusted Images

Images are easily transferrable from non-Cloud infrastructure to the Cloud due to the portability and availability of the container images on public image repositories. This presents a risk of allowing untrusted, malicious, and vulnerable container images within the Cloud infrastructure. In ensuring that trusted images are deployed on the workloads every there should be a centrally managed inventory to identify every image and repositories, this provides control to manage the allowed repositories and images.

In addition to this, identifying the images by names alone is not enough. Images should have a unique identification either using their hashes or other unique metadata. With the unique identification, an enforcement policy that ensures that every container workload only runs the trusted images from the approved registry. Image integrity check and continuous monitoring policy should be in place to guarantee the maintenance of the images and compliance of the images as the vulnerabilities and requirements changes (Souppaya et al. 2017).

According to MITRE ATT&CK, the persistence phase in a cloud attack could include adversaries implanting malicious container images within the infrastructure. Using tools such as Cloud Container Attack Tool (CCAT), attackers can plant backdoors in container images and create a reverse web shell to their command and control (C2). These are some of the potential threats facing the cloud container workloads. (MITRE ATT&CK. 2020a).

### 4.3 Container Orchestrator Security

Container orchestrators are the tools used in the management and automation of container deployments and regular tasks. Common container orchestrators providing the framework to manage the containers and other microservice include Kubernetes, Docker Swarm, Docker Compose, Fleet and Apache Mesos. They provide the management of the container tasks, some of which is listed below:

- Service scalability
- HA and DR management
- Rapid deployment and provisioning
- Allocation of resources to the containers
- Life cycle and configuration management
- Securing container communication
- Container workload scaling
- Scheduling and configuration
- Traffic routing and load balancing
- Container health monitoring
- Multi-cloud or multi-platform service development

(Pavlik & Mercl 2018)

Considering the scope of the roles the orchestrators in the management of container workloads it is important to consider and ensure its security. Kubernetes is one of the popular container orchestrators, the focus of the research in this chapter will be on the CSP managed services of Kubernetes and specifically the GKE with applicability to other CSPs.

#### 4.3.1 Run the latest version

One of the fundamental controls to have is to ensure that the latest version of Kubernetes is deployed as well as have an upgrade policy. By default, most of the CSPs do not offer the latest version of Kubernetes for the container workloads, leaving the responsibility to the users. Table 3 presents the available versions of Kubernetes and the corresponding offerings in the Cloud from Amazon, Google, and Microsoft (Foster 2020)

Table 3. Cloud Kubernetes versions and upgrade

| | GKE | EKS | AKS |
|---|---|---|---|
| **Default version** | 1.16 | 1.17 | 1.17 |
| **Supported versions** | 1.14 to 1.18 | 1.14 to 1.17 | 1.16 to 1.19 |
| **Control-plane upgrade** | CSP Managed automatically | Requires user to upgrade | Requires user to upgrade |

### 4.3.2 Private Cluster

Access to the control plane and nodes should be restricted. By default, access to the control is set to allow anyone access over the internet to make connections to the control plane, this default configuration should never be used. The first of ten phases of cloud workload attack tactic according to the MITRE ATT&CK cloud matrix is initial access with a technique of exploiting public-facing application. It means adversaries will try all possible exploits to access the hack value. (MITRE ATT&CK 2020b).

During this research, a search from Shodan for publicly available Kubernetes control plane reveals over 13000 workloads being available on the internet and some even presenting the login console as shown in figure 16. Hence, users must take these configurations critical, review the default configurations and continuously review the Cloud security posture management (CSPM) of the entire cloud infrastructure. Figure 17 shows the top 10 countries with publicly accessible Kubernetes control plane.

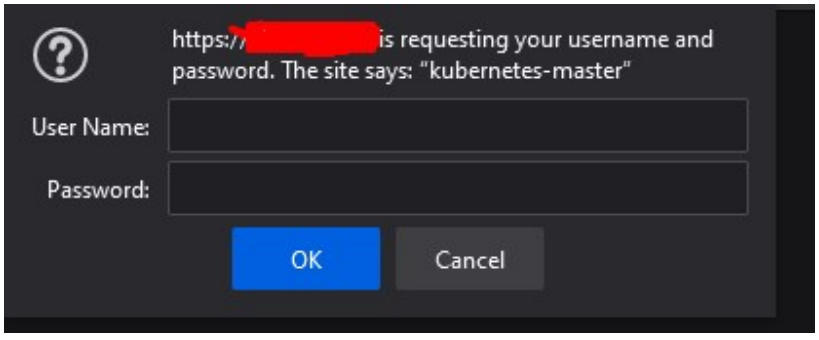


Figure 16. Login Prompt of a publicly exposed Kubernetes control plane

**Top Countries**

| | |
|---|---|
| 1. United States | 7,401 |
| 2. Germany | 1,621 |
| 3. Ireland | 1,554 |
| 4. Singapore | 613 |
| 5. China | 531 |
| 6. United Kingdom | 418 |
| 7. India | 378 |
| 8. Australia | 301 |
| 9. Japan | 222 |
| 10. France | 145 |

Figure 17 Top countries of internet-facing control plane (Shodan. 2020)

Figure 18 shows how prevalent the misconfigurations are within the cloud infrastructure. This research shows most of the responding clusters over the internet are present in the AWS Cloud (Shodan. 2020). This does not mean that the Cloud itself has the issue, rather it is the misconfiguration of the organisation utilizing the services because the control plane security has not been considered. In 2018, Tesla's Kubernetes infrastructure was infiltrated due to having a password-less k8s console and with privilege escalation storage buckets with sensitive data were accessed. (RedLock 2018).

Misconfigurations like this are common in the cloud. A typical example of this was an Accenture data leak due to publicly accessible Cloud storage containing decryption keys and other sensitive corporate and customer data (Ashok 2017).


Figure 18 Top CSPs with a publicly reachable Kubernetes control plane (Shodan 2020).

Figure 19 shows that Kubernetes is the most available API endpoint during the findings and presents how critical it is to protect the container workloads.

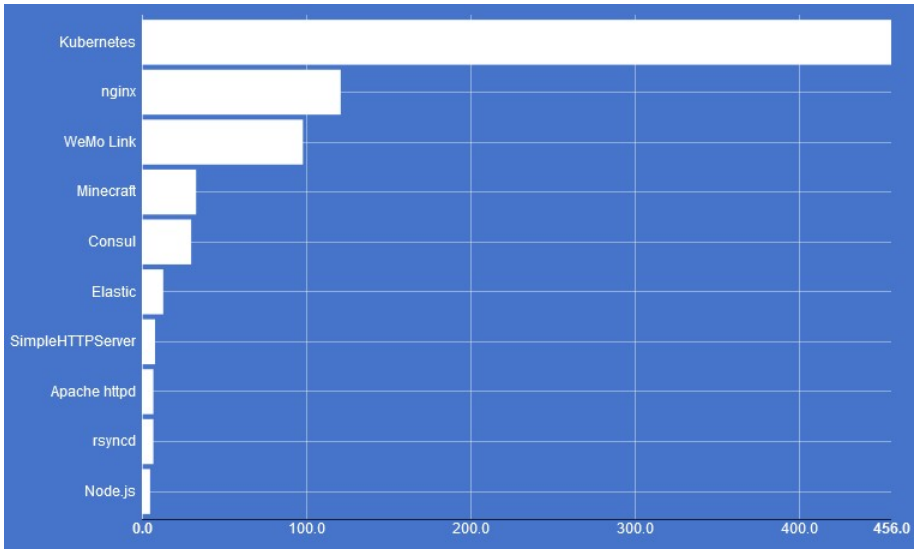


Figure 19 Kubernetes as the top publicly available service (Shodan 2020)

When deploying container workloads in the Cloud, it is highly recommended to create private clusters as a mitigation to the publicly exposed Kubernetes control planes. At the very least, create a whitelist of the authorized networks to the public endpoints of the cluster. Table 4 shows how to create private clusters in AWS, Azure and GCP using the CSP's respective SDK command-line interface. As an example, creating a private GKE cluster with access to the control plane endpoints only allowed from the RFC1918 range requires adding the following flag:

--enable-ip-alias

--enable-private-nodes

--enable-private-endpoint

--enable-master-authorized-networks (Google. 2020d)

Table 4 Creating Private Kubernetes Cluster in the Cloud

| GCP SDK configuration | AWS SDK Configuration | Azure SDK Configuration |
|---|---|---|
| gcloud container clusters create <private-cluster-name> \<br>--enable-master-authorized-networks \<br>--enable-ip-alias \<br>--enable-private-nodes \<br>--enable-private-endpoint \<br>--no-enable-basic-auth \ | aws eks update-cluster-config \ --region <region-code> \<br>--name <my-cluster> \<br>--resources-vpc-config endpointPublicAccess=<false>,endpointPrivateAccess=<true><br><br>(AWS. Amazon EKS cluster endpoint access control) | az aks create -n <private-cluster-name> -g <private-cluster-resource-group> --load-balancer-sku standard --enable-private-cluster<br><br>(Microsoft. 2020) |

### 4.3.3 Container-centric OS

There are inherent attack surfaces to every OS allowing for exploits. In containers, there is no need to have additional attack surface and it is highly recommended to use a Container-specific OS (CSO) which reduces the likelihood of exploiting host OS vulnerability thereby compromising the workloads. The host OS to be used must not allow the containers to mount directories on the host's filesystem or tampering of the filesystem.

The use of CSO reduces the attack risks associated with share kernel as there is not the availability of the application library/package managers as present in the general-purpose OS and improper user access rights (Souppaya et al. 2017).

By default, GKEs runs the Container-Optimized OS (COS) which has a minimal OS footprint such as read-only filesystem, file-system integrity check, locked-down firewall and audit logging. (Google. 2020a).

### 4.3.4 Role-Based Access Control (RBAC)

The criticality of the principle of least privilege (PLP) cannot be overemphasized in any infrastructure and Kubernetes is not an exception. Using PLP reduces the blast radius of an attack in case of a compromise. (Hausenblas & Rice. 2018). RBAC in Kubernetes involves some elements which are subject, resources, verbs, roles, and role bindings as depicted in figure 20.

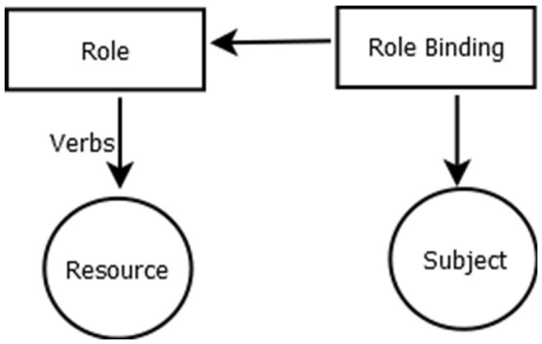


Figure 20 RBAC fundamentals in Kubernetes

The subjects are the entities requesting access to the Kubernetes API and the subjects could be users, groups, or service accounts (SA). The resources are the actual API available for use within the cluster while the verbs are the sets of actions specified in a role defining the permitted operations on the resources by the subjects. The summary of the verbs is CRUD (Create, Read, Update and Delete) actions and the Role bindings attach a role to the subject.

## 4.4 Securing the Container Runtime

Images transitions to container runtime after being deployed. At the runtime, there are new security concerns as new threats are discovered either due to misconfiguration or vulnerabilities in the container application libraries and dependencies. A survey by StackRox in 2019 shows that most organisations find it challenging securing the container runtime as shown in figure 21. (StackRox.

2019). This section describes some of the risks to be considered in the container runtime.

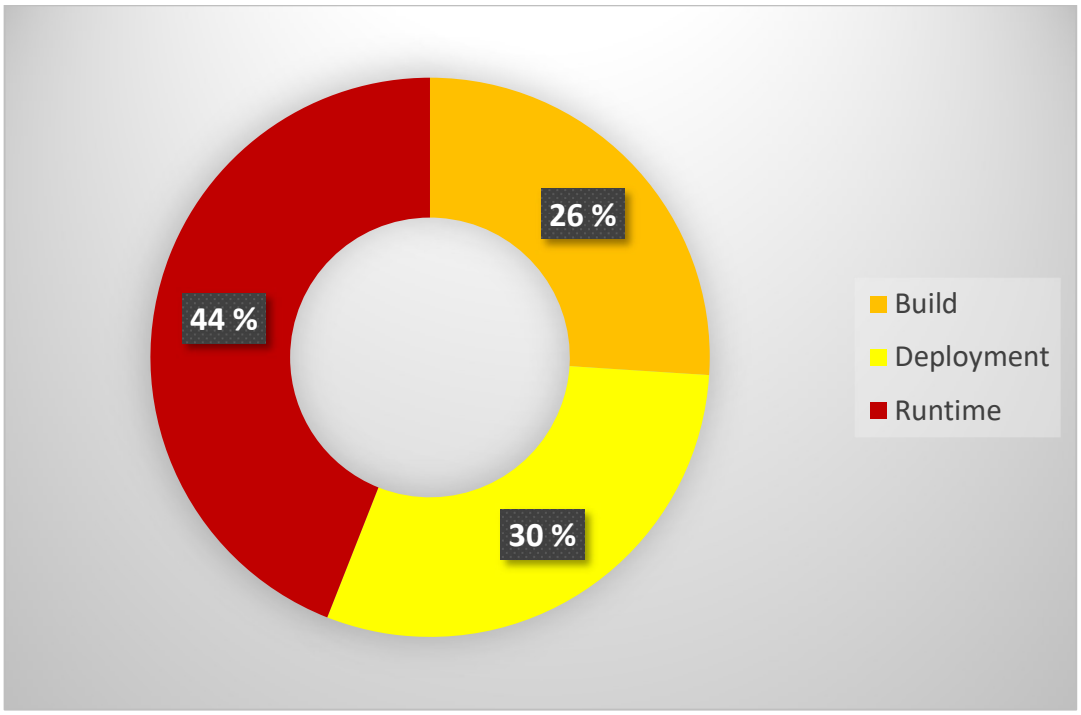


Figure 21 Container lifecycle security concerns (StackRox 2019)

### 4.4.1 Secret Management

Every application always requires some form of credentials to function as expected. The credentials otherwise known as secrets could be access tokens to make some API calls or even database (DB) credentials for some frontend application. It is a common practice to see application developers hardcoding these secrets to the application.

The PLP must be applied to the secrets granted to the containers and different set of secrets should be applied to different environments. For instance, the development secret must not be the same as the production environment. In addition to the PLP, secrets must be encrypted at rest using CMEK as well as in transit using TLS.

The secrets could be stored using the native Kubernetes storage or some other third-party solutions. By default, Kubernetes stores its secret with other configurations in the etcd, which is open-source which poses as a distributed and reliable key-value store for critical data of distributed systems. (Etcd Authors. 2020). The etcd values are stored as base64 encoding which could be easily decoded, for example using the echo command on a Linux OS as seen below.

```
echo YmFzZTY0IGRlY29kZXI= | base64 --decode
```

Organisations should ensure in addition to the encryption provided by the CSP, customer-managed encryption keys (CMEK) are also used to encrypt the workloads. Encryption key rotation policies should be applied to ensure that there are controls in place to rotate keys or revoke in case of a comprise. (Hausenblas & Rice. 2018). The use of third-party secret management solutions such as Conjur and Hashicorp vault are more secure because they separate secrets from the container workloads, provides a single secret management pane for all applications and prevents committing secrets to SCM. (CyberArk Conjur).

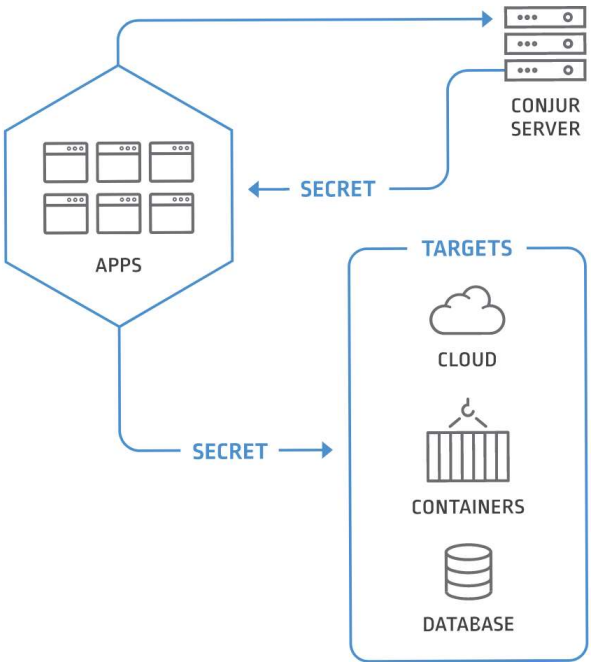


Figure 22 Third-party secret management solution workflow (CyberArk Conjur)

### 4.4.2 Micro-segmentation

The limitation of network traffic by dividing networks into small segments thereby reducing blast radius in a situation of a malicious attack or a breach referred to as micro-segmentation. It requires the implementation of a distributed firewall regulating access to the network traffic according to pre-defined security rules based on each resource. (Mujib & Sari 2020).

This leads to the concept of identity-based micro-segmentation (IBMS), implementing IBMS to secure the workloads eliminates the assumption that IP based network reachability means network authorization as shown in Figure 23.

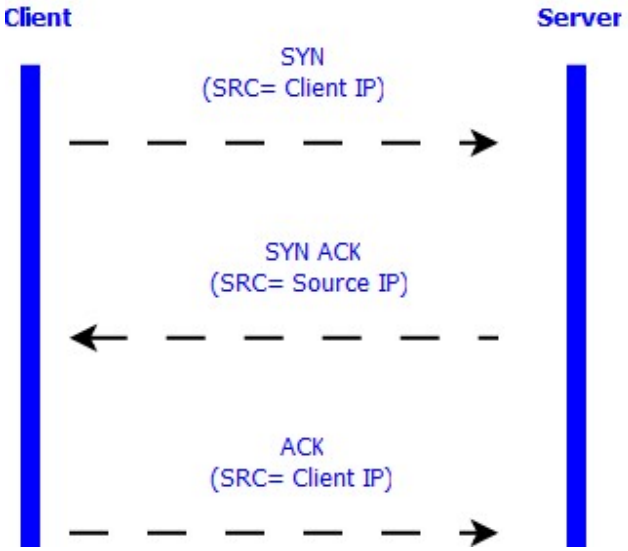


Figure 23 Traditional IP Connection request (Palo-Alto 2020a).

IBMS will apply several known metadata about a workload to provide a set of cryptographically signed identity dynamically learned from the cloud-native sources such as the system information, cloud provider and the container orchestrator. (Palo Alto 2020). Table 5 shows some cryptographic unique identity for the cloud-native identity source.

Table 5 identity-based micro-segmentation (Palo Alto. 2020).

| Cloud-native identity source | Cryptographic unique identity |
|---|---|
| System Information | OS<br>services<br>Hostnames |
| Cloud Provider | IAM roles<br>Other CSP metadata |
| Container Orchestrator | Kubernetes service accounts<br>Namespaces<br>Docker images<br>App labels |

Figure 24 shows the 3-way handshake of the IBMS in which the cryptographic identity is used to first authenticate client and server workloads. Based on the matching attributes, the network is authorised. The nonce is the arbitrary value used only once within the lifetime of a cryptographic session. (Rogaway. 2002).

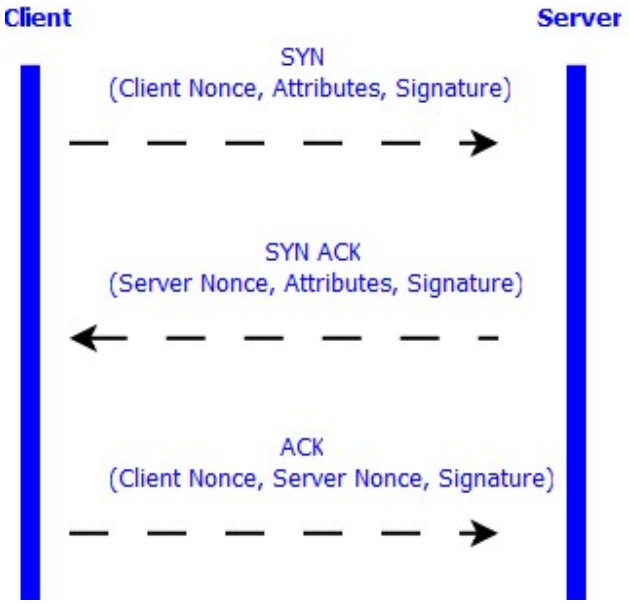


Figure 24. IBMS Connection Request (Palo Alto .2020)

By default, k8s allows Intra pod traffic within a cluster without restrictions. To limit traffic, Kubernetes uses network policies to declaratively configure how pods communicate with one another using a combination of allowed pods, allowed namespaces and IP block. (kubernetes.io 2020a).

K8s implements micro-segmentation with network plugins such as calico, Canal, Cilium, CNI-Genie etc. Most public clouds such as AWS, GCP and Azure supports some of the Kubernetes network plugins such as calico that helps to implement the rules defined in the network policies. (Calico 2019). Defining the rules on the k8s nodes is a time-consuming task to do manually that is not worth the effort as the containers are ephemeral which means the rules will have to be recreated every time the containers are created. (Rice 2020).

Moreover, a k8s cluster in a production environment will have more than one node, which means all nodes within the cluster must have all the rules defined for them. Leveraging on the k8s network policy objects provides the easy management and automation of the rules. The network policies could also be dynamically using container-native tools capable of learning about the normal container workloads network traffics.

Some best practices for using micro-segmentation to protect the cloud container workloads is having a default deny rule, default-deny egress, limit pod-to-pod traffic by allowing only pods with the right label to communicate and finally, allow only predefined ports for each container. (Rice 2020).

### 4.4.3 Service Mesh

Service mesh (SM) in k8s provides secure communication for the services of a container cluster. It provides mutual TLS (mTLS) encrypting communications within container workload infrastructure and protecting the infrastructure from MiTM attack. Unlike the network policies operating from OSI layer 3 to 4, service mesh secures the container workloads from OSI layer 5 to 7 which is the session to application layer respectively. (Rice 2020).

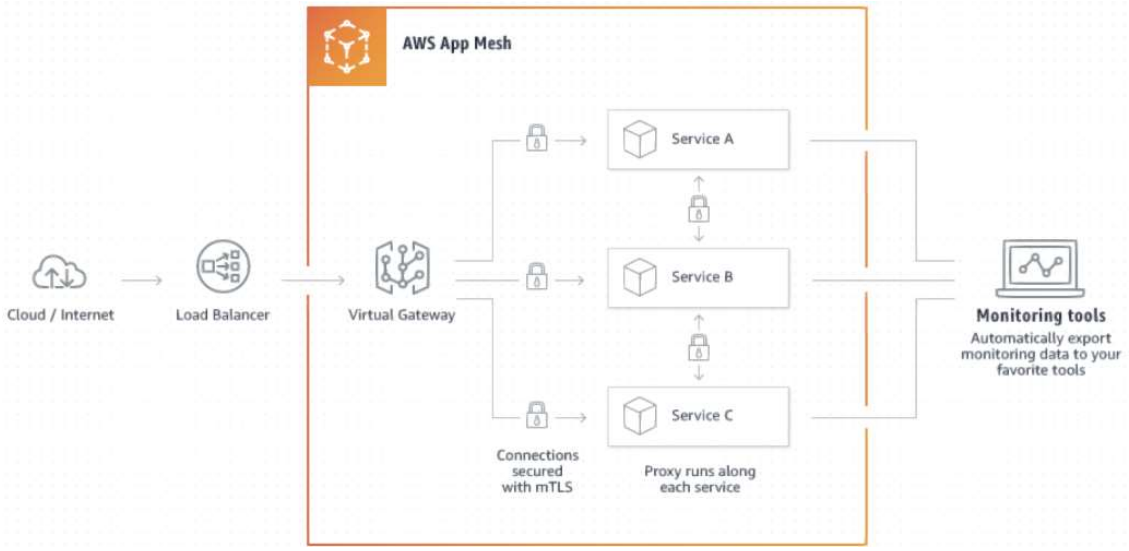


Figure 25 Service Mesh mTLS (Amazon AWS App Mesh)

In addition to securing the communication between the services with mTLS, which gives authentication and authorization for the services, SM provides the possibility of creating policies that are enforced across the infrastructure. (Google 2020d). For Cloud-managed k8s services, there are managed service mesh solutions such as GCP's Anthos service mesh (ASM) and AWS App Mesh. SM is made of proxies called sidecars because they run alongside the services. All the requests between the microservices are routed through the proxies for security, traffic management and monitoring.

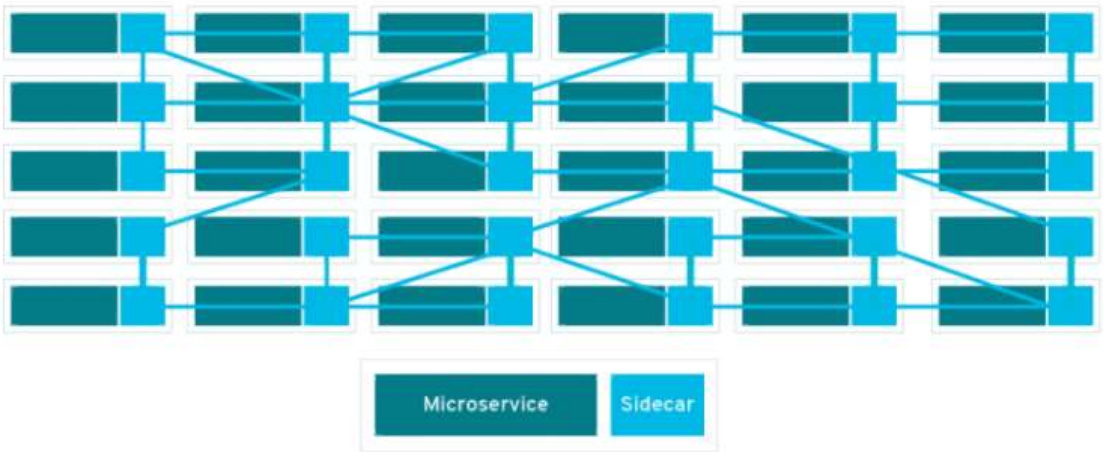


Figure 26 Service Mesh sidecars (RedHat. What is service mesh?)

If service mesh will be used, it is recommended to have some controls that ensure that the sidecars are present in all the containers and possibly have a complementary solution to restrict the traffic flow between the containers as well as external IP addresses or domains.

### 4.5 Threats and mitigation

Microsoft Azure security team presented the first Kubernetes attack matrix using the MITRE ATT&CK framework. It shows threat landscape, tactics, and the techniques an adversary could use to exploit a Kubernetes infrastructure. (Weizman 2020b)

This matrix includes forty techniques with nine tactics that an attacker can use to compromise a container workload as shown in Figure 24. Usually, the mitigations for the threats have overlapping security controls, organisations should ensure policies are in place to secure the container workloads within the Cloud.

| Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion | Credential Access | Discovery | Lateral Movement | Impact |
|---|---|---|---|---|---|---|---|---|
| Using Cloud credentials | Exec into container | Backdoor container | Privileged container | Clear container logs | List K8S secrets | Access the K8S API server | Access cloud resources | Data Destruction |
| Compromised images in registry | bash/cmd inside container | Writable hostPath mount | Cluster-admin binding | Delete K8S events | Mount service principal | Access Kubelet API | Container service account | Resource Hijacking |
| Kubeconfig file | New container | Kubernetes CronJob | hostPath mount | Pod / container name similarity | Access container service account | Network mapping | Cluster internal networking | Denial of service |
| Application vulnerability | Application exploit (RCE) | | Access cloud resources | Connect from Proxy server | Applications credentials in configuration files | Access Kubernetes dashboard | Applications credentials in configuration files | |
| Exposed Dashboard | SSH server running inside container | | | | | Instance Metadata API | Writable volume mounts on the host | |
| | | | | | | | Access Kubernetes dashboard | |
| | | | | | | | Access tiller endpoint | |

Figure 27. Kubernetes attack Matrix (Weizman 2020b)

The attacks on the container workloads could be resolved either within Kubernetes itself, at the CSP or other cloud-native tooling such as the container registries.20 of the 40 threats can be mitigated by implementing security controls

in Kubernetes itself, 6 techniques could be mitigated implementing controls in toolchains, 3 mitigated at the CSP and 11 mitigated with a set of controls in k8s and at the CSP. (Dang 2020). The details of the individual technique are presented in this chapter.

### 4.5.1 Initial Access

This tactic refers to an attacker gaining access to the GKE, AKS or the EKS clusters deployed using the cloud account credentials. The compromise of a cloud account means the possibility of gaining access to the management plane. Table 6 details the techniques and recommended mitigation for the initial access tactic.

Table 6 Initial Access Mitigation (Dang. 2020)

| Technique | Mitigation | security controls |
|---|---|---|
| **Using Cloud Credentials** | • Avoid the use of shared accounts<br>• Configure Cloud IAM to restrict access to privileged credentials.<br>• Implement PLP | CSP |
| **Compromised images in the registry** | • Only use private registries<br>• Implement Trust image policy<br>• Developers to use only approved base image.<br>• Limit access to the registry<br>• Continuous VC scans of the registry | Tools (Registry) |
| **kubeconfig file** | • Always use the latest version of Kubernetes clients<br>• Authenticate to the API server using a third-party DAP | Kubernetes |

| | | |
|---|---|---|
| **Application vulnerability** | • Implement workload identity<br>• Scan images for vulnerabilities<br>• Limit external access to pods with network policies<br>• Implement admission controls to prevent high/critical severity images from deployment | CSP<br>K8s |
| **Exposed Dashboard** | • Use private endpoint to dashboard<br>• Delete/disable dashboard if not required<br>• Restrict ingress traffic to the dashboard | K8s |

### 4.5.2 Execution

This tactic allows the attacker to run the malicious codes inside the cluster and table 7 shows the techniques used to exploit container workloads and the possible mitigations to secure the container workloads in the cloud.

Table 7 Execution phase mitigation (Dang 2020).

| Technique | Mitigation | security controls |
|---|---|---|
| **Exec into containers** | • Implement RBAC with PLP access to pods<br>• Delete unnecessary processes from containers<br>• Run pods with RO file system | K8s |
| **Bash/cmd inside a container** | • Limit access to the workloads | CSP |
| **New container** | • Implement RBAC to create pods | K8s |

| | | |
|---|---|---|
| **Application exploit** | • Scan images for vulnerabilities<br>• No container should have code execution vulnerabilities | K8s |
| **SSH server running inside a container** | • Policy to prevent SSH server process in containers<br>• Have monitoring and audit all SSH servers running in the containers | K8s and Tooling |

### 4.5.3 Persistence

In the persistence phase, the attackers establish backdoors to ensure that persistence access to the cluster is maintained if initial connectivity access is lost. The techniques involved in this stage is presented in table 8.

Table 8 Persistence Mitigation (Dang 2020).

| Technique | Mitigation | security controls |
|---|---|---|
| **Backdoor container** | Implement RBAC to create pods and abstractions that create pods | K8s |
| **Writable hostPath mount** | • In CSP limit node lifetime to 24hrs and automatically provision new nodes as a replacement.<br>• Policy to limit/disallow host mount | CSP<br>K8s |

| | | |
|---|---|---|
| | • Make required host paths as RO | |
| **Kubernetes CronJob** | RBAC with PLP to create pods and jobs | K8s |

### 4.5.4 Privilege Escalation (PE)

The PE tactics involve the techniques used by the attacker to have an elevated right within the clusters. Some of the access rights could be gaining access to the nodes from the containers, privilege elevation in the container and possibly access to cloud infrastructure, especially in environments without RBAC or misuse of cloud service accounts.
The techniques and the possible mitigations for the PE tactic are presented in table 9.

Table 9 Privilege Escalation mitigation (Dang 2020).

| Technique | Mitigation | security controls to configure |
|---|---|---|
| **Privileged container** | • Restrict the use of running privileged container workloads<br>• If any implement RBAC and network policies to restrict network access to the privileged workloads. | K8s |
| **Cluster-admin binding** | • RBAC to limit admins with access to cluster-admin role and admins able to create role bindings. | K8s |

| | | |
|---|---|---|
| | • Avoid the use of cluster-admin role and grant groups with granular permissions with PLP | |
| **hostPath mount** | • In CSP limit node lifetime to 24hrs and automatically provision new nodes as a replacement.<br>• Implement a policy to prevent host mount in container workloads<br>• Use pod security policies to specify the allowed file path to be mounted | CSP<br>K8s |
| **Access Cloud resources** | • Implement workload identity where applicable.<br>• limit node lifetime to 24hrs and automatically provision new nodes as a replacement.<br>• Implement a policy to prevent host mount in container workloads<br>• Use pod security policies to specify the allowed file path to be mounted | CSP<br>K8s |

### 4.5.5 Defense Evasion

In the defense evasion, the attacker uses different tactics to prevent being detected and hide all trails leading to its activities. Details of the techniques and possible mitigation are presented table 10 below.

Table 10 Defense Evasion Mitigation (Dang 2020).

| Technique | Mitigation | security controls to configure |
|---|---|---|
| **Clear container logs** | • Restrict host mounts<br>• In CSP limit access to the cluster workloads and implement PLP<br>• Implement a SIEM system to collect logs or have persistence storage. | K8s<br>CSP<br>Tool |
| **Delete K8s events** | • Limit workload access to the nodes.<br>• Enable the logging supported features | K8s<br>CSP |
| **Pod/container name similarity** | • Implement RBAC for Pod creations APIs<br>• Use PLP for access to the cluster | K8s |
| **Connect from a Proxy server** | • Restrict access to the k8s API server | CSP |

### 4.5.6 Credential Access

In this tactic, the attacker starts using techniques to steal sensitive credentials such as passwords, service tokens, service accounts, secret key stores in the

cluster, the cloud or application credentials. Table 11 shows the recommended mitigations.

Table 11 Credential Access mitigation (Dang 2020).

| Technique | Mitigation | security controls |
|---|---|---|
| **List k8s secrets** | • Use namespaces to limit the scope of a secret<br>• Implement secret rotation.<br>• Store secrets outside of the container workloads | K8s |
| **Mount service principal** | • limit node lifetime to 24hrs and automatically provision new nodes as a replacement.<br>• Use pod security policies to specify the allowed file path to be mounted. | CSP<br>K8s |
| **Access container service account** | • Service accounts should have RBAC with PLP.<br>• Avoid the use of the cluster-admin role | K8s |
| **Applications credentials in configuration files** | • Ensure no secrets are written to manifest files.<br>• Scan files to identify secrets stored in the SCM | K8s<br>Tools |

### 4.5.7 Discovery

In this phase, the attacker explores ways to identify more resources within the k8s cluster thereby allowing an attacker to make lateral movement within the container workload infrastructure. By default, k8s does not restrict traffic between the pods, hence making the discovery exercise and enumeration easier to do. Some of the discovery would include getting instance metadata (such as SSH pub keys, network configs,) and running pods using https://<NODE IP>:10255/pods/. With network policies, access to the kubelet port can be restricted as follows:

```
apiVersion: crd.projectcalico.org/v1
kind: GlobalNetworkPolicy
metadata:
  name: deny-access-to-kubelet-port
spec:
  types:
  - Egress
  egress:
  - action: Deny
    protocol: TCP
    destination:
     nets:
     - 0.0.0.0/0
     ports:
     - 10255
    source: {}
```

Other recommended mitigations to the discovery tactic are in table 12 below.

Table 12 K8s Discovery mitigation (Dang 2020).

| Technique | Mitigation | security controls |
|---|---|---|
| **Access the K8s API server** | • Restrict users and SA with access to the k8s API server<br>• Use private clusters | K8s<br>CSP |

| | | |
|---|---|---|
| | • Access to external clusters to be limited to only trusted IPs | |
| **Access Kubelet API** | • Implement network policies to block pod access to the Kubelet port | K8s |
| **Network mapping** | • Implement micro-segmentation to restrict traffic between pods. | K8s |
| **Access Kubernetes dashboard** | • Don't use the Kubernetes dashboard, rather use the cloud console dashboard | K8s |
| **Instance API** | • Enable metadata concealment or workload identity<br>• Implement egress network policy to restrict to the cloud metadata services | CSP |

### 4.5.8 Lateral Movement

The lateral movement tactics focus on navigating through the container clusters to gain access to other resources within the cluster, underlying node and ultimately other cloud resources within the infrastructure.
For instance, every workload in AKS has a service principal (SP) used for the creation and management of the resources used for cluster operations. The SP is stored in /etc/kubernetes/azure.json, so an attacker with access to the credentials can access and modify the resources. Table 13 shows the recommended mitigation actions.

Table 13 Lateral Movement Mitigation (Dang 2020)

| Technique | Mitigation | security controls |
|---|---|---|
| **Access cloud resource** | • limit node lifetime to 24hrs and automatically provision new nodes as a replacement.<br>• Enable metadata concealment or workload identity<br>• Implement a policy to prevent host mount in container workloads<br>• Use pod security policies to specify the allowed file path to be mounted | CSP<br>K8s |
| **Container service account** | • Implement RBAC with PLP for SA and role bindings | K8s |
| **Cluster internal networking** | • Implement network policies to restrict inter pod network traffic | K8s |
| **Applications credentials in configuration files** | • Ensure no secrets are written to manifest files.<br>• Scan files to identify secrets stored in the SCM | Tools<br>K8s |
| **Writable volume mounts on the host** | • In CSP limit node lifetime to 24hrs and automatically | K8s<br>CSP |

| | | |
|---|---|---|
| | provision new nodes as a replacement.<br>• Policy to limit/disallow host mount<br>• Make required host paths as RO | |
| **Access K8s dashboard** | • Do not use the Kubernetes dashboard, rather use the cloud console dashboard | K8s |
| **Access tiller endpoint** | • If using Helm, upgrade to the latest version of Helm | Tools |

### 4.5.9 Impact

The last tactic of the Kubernetes attack matrix is the impact where the attacker finally launches the offensive to destroy the entire container workloads, cause a denial of service and obtain other hack values from the infrastructure. In mid-2020, there was a large-scale campaign where attackers have impacted the cloud container workloads by compromising them for cryptojacking attack. (Weizman 2020a). Recommended mitigation techniques are contained in table 14.

Table 14 Impact Mitigation (Dang 2020).

| Technique | Mitigation | security controls |
|---|---|---|
| **Data Destruction** | • Actively monitor the audit logs | K8s<br>CSP |

| | | |
|---|---|---|
| | • Implement RBAC with PLP to SIEM<br>• Restrict access to k8s nodes and configuration APIs from CSP | |
| **Resource Hijacking** | • Restrict access to k8s cluster nodes<br>• RBAC with PLP to create pods<br>• Do not deploy public images in production without scanning for VC | K8s<br>CSP |
| **Denial of service** | • Always deploy the latest version of k8s available from the CSP<br>• Only permit allowed connections between pods.<br>• Restrict access to the cluster API and internally consumed services to only allowed IP.<br>• Implement effective monitoring and alert security solution | K8s<br>CSP |

# 5 CONTAINER SECURITY STANDARDS AND PUBLICATIONS

Cybersecurity consists of several standards with different sets of policies used to protect critical infrastructures across different industries. The recommended policies are sets of proven guidelines that could help secure the infrastructure. Also, organisations have different standard compliance requirements needed by law. For the container security and the findings done in this thesis work, the examined standards and publication are from NIST, CIS, CSA, PCI and publications from IEEE and leading container security vendors such as StackRox, Aquasec, Palo Alto, Microsoft and Oracle just to mention a few.

## 5.1 NIST Special Publications

The NIST special publication 800-190 is a comprehensive set of guidelines to secure application container technologies. It presents the threats and the countermeasures to consider when planning, implementing, and maintaining a container technology infrastructure. (Souppaya et al. 2017.)

The publications focus on the risks associated with container images, the image registries, host OS, orchestrator, and the container runtime. Organisations are responsible to implement the countermeasures using any of the security tools with the technological capabilities to address the risks described in the special publication 800-190. Some of the solutions providing the CWPP include Aquasec, Twistlock, Neuvector, StackRox, Sysdig, Anchor and many others.

## 5.2 CIS

CIS publishes several benchmarks to configure the infrastructure from the OS, Cloud providers, network devices, browsers etc. It has also published recommended best practices for docker and Kubernetes virtualization.
The CIS benchmark for Kubernetes provides a set of general guidelines to securely configure the open-source Kubernetes infrastructure.
For users consuming the Kubernetes CMS such as GKE and EKS, the security of container workloads is a shared responsibility between the users and the CSP.

To ensure that the right set of controls are implemented, the applicable CSP Kubernetes benchmark such as CIS GKE benchmark and CIS EKS benchmark should be applied as they are a subset of the CIS Kubernetes Benchmark addressing the specific risks of using the Kubernetes Cloud Managed services. (Google. 2020b).

Security solutions that support the k8s CMS benchmarks should be deployed for an automated audit and continuous compliance monitoring. A manual check could as well be done using the open-source kube-bench for the worker nodes as follows; kube-bench node --benchmark cis-1.5

### 5.3 PCI Secure Cloud Computing Guidelines

The PCI Cloud computing guidelines prescribed standards to fulfilled by organisations accepting payment cards. It has defined some minimum requirements to deploy application container workloads. One of the policies relating to secret management is required to ensure that secrets are not stored within the cluster.

“Access controls to both the orchestration framework and the containers themselves such that different workloads do not have access to keys, identity tokens and other sensitive information used by other containers in the cluster” (Cloud Special Interest Group. 2018). Other recommended policies have been discussed in this research work.

### 5.4 Cloud Security Alliance

The Cloud security alliance, CSA have created a set of guidelines across 14 different domains to ensure a secured Cloud architecture, adequate governance in the Cloud and Operating the Cloud securely. (Mogul et al. 2017). It is a highly recommended standard for cloud workloads.

# 6 RESULTS AND DISCUSSION

This research seeks to answer the three research questions identified at the beginning of this thesis as specified in chapter 2.1. The research questions are listed below with the discussion and findings to the questions are presented in the topics are contained in this chapter. The applicable security controls from the results presented in this section are also implemented by the commissioner.

- What security controls should be implemented to protect the container workloads in the Cloud?".
- What is the current threat landscape of containers and what are the recommended mitigation?
- What cloud controls can be utilized to secure the container workloads deployed in the cloud?

## 6.1 What security controls should be implemented to protect container workloads in the cloud?

Securing the container workloads is not a "press one-button" approach to security, it requires a well-planned approach to secure the containers irrespective of the orchestration tool. This research shows that container security must be part of their entire lifecycle; from the build to running phase and every element in-between.

The building of base images must comply with the industry standard and hardened using the recommended benchmark such as the CIS benchmark. Having a hardened base image is not enough because threats and vulnerabilities are always evolving, so organisations must have continuous security as part of the CI/CD to ensure that vulnerability and compliance are always monitored.

The challenge most organisations face is using processes and tools not designed for securing container workloads in the cloud, processes such as patch management and tools to be adopted must adapt rapidly a containerized infrastructure and able to interpret the cloud constructs and provide more context about any vulnerabilities or misconfigurations.

Establishing a list of trusted images to be used for deployment, implementing RBAC with PLP and ensuring that all data at rest and in transit are always encrypted will further strengthen the security posture of the container workloads in the cloud.

In the runtime, employing a defense-in-depth approach helps further secure the containers. Limiting access between containers with the use of micro-segmentation is a recommended practice to ensure that container workload is only allowed to connect with specified namespaces and ports. The runtime must be continuously monitored for any deviation from set security policies and immediately remediated based on the associated risk factor.

It is highly recommended that organisations implementing container workloads should at the minimum ensure that the infrastructure follows the NIST special publication 800-190 and the CIS benchmark for the adopted k8s cloud-managed service.

### 6.2 What are the current threat landscape of containers and recommended mitigation?

Attackers are constantly exploiting the vulnerabilities associated with containers either in the runtime or images stored in registries such as a malicious image being downloaded over 5million times from docker hub (Goodin. 2018). The main container threats in the cloud include application vulnerability, misconfigured cloud IAM, insufficient RBAC with PLP, secret exposure, malicious container images, overly permissive network, and insecure orchestrator configurations.

This study shows that the threat landscape of the containers is no different from the non-containerized applications concerning the tactics which are shown from in the matrix from chapter 4.5. There is however some peculiarity with the containers especially for the orchestrator being considered in this research which is the Kubernetes.

The uniqueness includes the presence of a control plane being the central management system to control the operations of all the nodes and containers in a container cluster. An adversary with access to a misconfigured and vulnerable control plane can exploit and impact the normal operations of the cluster.

Most of the identified threats can be mitigated by implementing effective RBAC with PLP, using micro-segmentation and having a cloud-native runtime security solution. Security minded organisations should build their compliant images and use private registries instead of utilizing publicly available images.

### 6.3 What cloud security controls can be utilized to secure the workloads deployed in the cloud?

Deploying in the cloud comes with its own risk and the required level of applicable security control to be implemented in the cloud depends on the deployment model being adopted, either IaaS or PaaS. The k8s cloud managed services are the PaaS services, which means the security of the underlining infrastructure is the responsibility of the CSP based on the shared responsibility model.

When deploying container workloads in the cloud, IAM is a critical foundation element that must be configured with RBAC. IAM services such as GCP “cloud IAM Conditions” provides a possibility to define the required set of permissions for the human and service accounts needed by the cluster.

It was interesting to see how k8s clusters are publicly available on the internet with the majority of them available on the public cloud, this shows that the use of private clusters should be defined as a guardrail in the cloud. In addition to this and reducing the attack surface. Besides, some cloud providers by default create a publicly accessible load balancer with firewalls allowing access from 0.0.0.0/0, this should be reviewed to ensure that the services are not publicly exposed by creating private load balancers. The research shows using minimalistic OS should be favored over the full feature OS for the container workloads.

### 6.4 Recommended future research.

Container technologies and especially using Kubernetes as orchestrator continues to evolve and develop new features and tools to manage them. The threat to the containerized infrastructure is also evolving so is the security standards to protect containers and the use of the cloud also adds additional complexity to the security of the workloads if not configured properly.

This research did not examine in detail the specifics of different cloud provider when deploying the container workloads in the cloud and this presents an opportunity for research and learning on the capabilities of different cloud providers to securely deploy container workloads. An additional research proposal is identifying the specific monitoring metrics to effectively implement situation awareness for a containerized infrastructure in the cloud.

## REFERENCES


Abbassi, P. 2019. Building Container Images with Podman and Buildah. WWW document. Available at: https://www.giantswarm.io/blog/building-container-images-with-podman-and-buildah [Accessed 05 December 2020].

Amazon AWS App Mesh. Amazon Web Services. Available at: https://aws.amazon.com/app-mesh [Accessed 05 November 2020].

Ashok, A. 2017. Accenture Data leak: 'Keys to the Kingdom' left exposed via multiple unsecured cloud servers. International Business Times., 11 October 2017. Available at: https://www.ibtimes.co.uk/accenture-data-leak-keys-kingdom-left-exposed-via-multiple-unsecured-cloud-servers-1642653 [Accessed 05 December 2020] .

AWS. Amazon EKS cluster endpoint access control. WWW document. Available at: https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html [Accessed 15 December 2020].

Bernstein, B. 2018. Container Image Registry Security Best Practices. WWW document. Available at: https://thenewstack.io/container-image-registry-security-best-practices/ [Accessed 02 December 2020].

Brady, K., Moon, S., Nguyen, T. & Coffman, J. 2020. Docker Container Security in Cloud Computing. IEEE. 10th Annual Computing and Communication Workshop and Conference (CCWC)

Bytemark. 2019. Kubernetes Terminology: Glossary. WWW document. Available at: https://docs.bytemark.co.uk/article/kubernetes-terminology-glossary/ [Accessed 26 April 2020].

Calico. 2019. Managed Public Cloud. Calico Project. WWW document. Available at: https://docs.projectcalico.org/getting-started/kubernetes/managed-public-cloud/ [Accessed 05 December 2020].

CIS Docker 2019. CIS Docker Benchmark v1.2.0. WWW document. Available at: https://learn.cisecurity.org/l/799323/2020-07-02/yz62 [Accessed 26 April 2020].

Cloud Special Interest Group. 2018. PCI Security Standards Council. Information Supplement: PCI SSC Cloud Computing Guidelines Version 3.

CyberArk Conjur. Secret management solution. WWW document. Available at: https://www.conjur.org/ [Accessed 11 December 2020].

Dang, W. 2020. Protecting Kubernetes: Kubernetes Attack Matrix and How to Mitigate its Threats. WWW document. Available at: https://www.stackrox.com/post/2020/06/protecting-against-kubernetes-threats-chapter-1-initial-access/ [Accessed 15 December 2020].

Dissanayake, L. & Mistry, P. 2020. Introducing Amazon ECR server-side encryption using AWS Key Management System. WWW document. Available at: https://aws.amazon.com/blogs/containers/introducing-amazon-ecr-server-side-encryption-using-aws-key-management-system/ [Accessed 29 December 2020].

Docker. 2020a. About Storage Drivers. WWW document Available at https://docs.docker.com/storage/storagedriver/ [Accessed 04 December 2020].

Docker. 2020b. Release notes. WWW document. Available at: https://docs.docker.com/engine/release-notes/ [Accessed 10 December 2020].

Etcd Authors. 2020. ETCD WWW document. Available at: https://etcd.io [Accessed 29 November 2020].

Field, J., Lepow, D., Moat, L., Macy, M., Jenks, A. & Connock, J. 2018. Best Practices for Azure Container Registry. WWW document. Available at: https://docs.microsoft.com/en-us/azure/container-registry/container-registry-best-practices [Accessed 02 December 2020].

Foster M, 2020. EKS vs GKE vs AKS – Evaluating Kubernetes in the Cloud. WWW document. Available at: https://www.stackrox.com/post/2020/10/eks-vs-gke-vs-aks/ [Accessed 10 November 2020].

Goodin, D. 2018. Backdoored images downloaded 5million times finally removed from Docker Hub. WWW document. Available at: https://arstechnica.com/information-technology/2018/06/backdoored-images-downloaded-5-million-times-finally- [Accessed 11 January 2021].

Google. 2020a. Container-Optimized OS. WWW document. Available at: https://cloud.google.com/container-optimized-os [Accessed 29 November 2020].

Google. 2020b. CIS Benchmarks. WWW document. Available at: https://cloud.google.com/kubernetes-engine/docs/concepts/cis-benchmarks [Accessed 04 November 2020].

Google. 2020c. GKE Cluster Architecture. WWW document. Available at: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture [Accessed 90 January 2021].

Google. 2020d. Hardening your cluster's security. WWW document. Available at: https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster [Accessed 29 November 2020].

Google. 2020e. Vulnerability Scanning. WWW document. Available at: https://cloud.google.com/container-analysis/docs/vulnerability-scanning [Accessed 29 November 2020].

Hausenblas, M. & Rice, L. 2018. Kubernetes Security: Operating Kubernetes Clusters and applications safely. 1st Edition. O'Reilly Media. https://shodan.io/report/He7efzH9. [Accessed 04 December 2020].

Jenkins. 2018. Anchor Container Image Scanner. WWW document. Available at: https://wiki.jenkins.io/display/JENKINS/Anchore+Container+Image+Scanner+Plugin [Accessed 10 December 2020].

Johnston, M. & Newcomer, K. 2018. What is Container Security? WWW document. Available at: https://www.redhat.com/en/topics/security/container-security [Accessed 26 April 2020].

Joy, A. 2015. Performance Comparison between Linux Containers and Virtual Machines. IEEE. International Conference on Advances in Computer Engineering and Applications Ghaziabad, 2015, 342-346.

Kaczorowski, M. 2019 Exploring Container Security: The shared responsibility model in GKE. Google GCP. WWW document. Available at: https://cloud.google.com/blog/products/containers-kubernetes/exploring-container-security-the-shared-responsibility-model-in-gke-container-security-shared-responsibility-model-gke [Accessed 09 November 2020].

Kubernetes.io. 2020a. Kubernetes components. WWW document. Available at: https://kubernetes.io/docs/concepts/overview/ [Accessed 10 December 2020].

Kubernetes.io. 2020b. What is Kubernetes? Kubernetes. WWW document Available at https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Accessed 26 April 2020].

Lee, T. & Hogenson, G. 2020. How Bridge to Kubernetes works. WWW document. Available at:. https://docs.microsoft.com/en-us/visualstudio/containers/overview-bridge-to-kubernetes?view=vs-2019 [Accessed 10 January 2021].

MacDonald, M. & Croll, T. 2020. Market Guide for Cloud Workload Protection Platforms. WWW document. Available at: https://www.gartner.com/doc/reprints?id=1-1ZF2YMQ5&ct=200707&st=sb [Accessed 10 May 2020].

Maharjan, Y. 2020. How rolling and rollback deployments work in Kubernetes. WWW document. Available at: https://medium.com/@yankee.exe/how-rolling-and-rollback-deployments-work-in-kubernetes-8db4c4dce599 [Accessed 10 January 2021].

Microsoft. 2020. Create a private Azure Kubernetes Service Cluster. WWW document. Available at: https://docs.microsoft.com/en-us/azure/aks/private-clusters [Accessed 11 November 2020].

MITRE ATT&CK. 2020a. Cloud Matrix. WWW document. Available at: https://attack.mitre.org/matrices/enterprise/cloud/ [Accessed 29 November 2020].

MITRE ATT&CK. 2020b. Implant Container Image. WWW document. Available at: https://attack.mitre.org/techniques/T1525/ [Accessed 05 December 2020].

Mogul, R., Arlen, J., Gilbert, F., Lane, A., Mortman, D., Peterson, G. & Rothman, M. 2017. Security Guidance for Critical Areas of Focus in Cloud Computing v4.0.

Moore, S. 2020. Gartner Forecasts Strong revenue growth for global container management software and services through 2024. WWW document. Available at: https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co [Accessed 10 May 2020].

Moyle. Ed. 2020. Benefits of Open Source container vulnerability scanning. WWW document. Available at: https://searchcloudsecurity.techtarget.com/tip/scan [Accessed 10 November 2020].

Mujib, M. & Sari, R. Performance Evaluation of Data Center Network with Network Micro-segmentation. 2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE) IEEE Xplore.

O'Brien, R. 2001. An Overview of the Methodological Approach of Action Research. Available at https://www.web.ca/arfinal.html#_Toc26184651 [Accessed 25 April 2020].

Palo-Alto. 2019. Container security for dummies. WWW document. Available at: https://www.paloaltonetworks.com/resources/guides/container-security-for-dummies [Accessed 4 April 2020].

Palo-Alto. 2020. Identity-Powered Micro-segmentation: Going beyond network boundaries to protect cloud-native applications. ebook-081720.

Pavlik, J. & Mercl, R. 2018. University of Hradev Kravlove. The Comparison of Container Orchestrators. Available at: https://www.researchgate.net/publication/323417485_The_Comparison_of_Container_Orchestrators

RedHat. 2018. Container and Images. OpenShift 3.0.WWW document. Available at: https://docs.openshift.com/enterprise/3.0/ [Accessed 04 December 2020].

RedHata. What is a Kubernetes Cluster? WWW document. Available at: https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-cluster [Accessed 10 November 2020].

RedHatb. What is Service Mesh? WWW document. Available at: https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh [Accessed 13 November 2020].

RedLock. 2018. CSI Team. Lessons from the Cryptojacking Attack at Tesla. Blog. Available at: https://redlock.io/blog/cryptojacking-tesla [Accessed 05 November 2020].

Rice, L. 2020. Container Security: Fundamental Technology Concepts that protect Containerized Applications. 1st Edition. O'Reilly.

Rogaway.P. 2002. Nonce-Based Symmetric Encryption. University of California, Davis. Available at: https://rdcu.be/ceXdJ

Sanz, J., Carter. E. & Anderson, K. 2018. Running Containers in Production for dummies. Ebook. Sysdig. Available at: https://sysdig.com/resources/ebooks/running-containers-in-production-for-dummies/ [Accessed 10 October 2020].

Shodan. 2020. Kubernetes-online20201220. WWW document. Available at: https://shodan.io/report/He7efzH9 [Accessed 15 November 2020].

Simi, S. 2019. Docker Image Vs Container: The Major Differences. WWW document. Available at: https://phoenixnap.com/kb/docker-image-vs-container [Accessed 25 April 2020].

Smith, M. 2017. What is action research and how do we do it. The encyclopedia of pedagogy and informal education. WWW document. Available at: [https://infed.org/mobi/action-research/. Accessed on [Accessed 25 April 2020].

Souppaya, M., Morello, J. & Scarfone, K. 2017. National Institute of Standards and Technology. Special publication 800-190. Application Container Security Guide. 2017. PDF document. Available at: https://doi.org/10.6028/NIST.SP.800-190 [Accessed 25 April 2020].

StackRox. 2019. Container and Kubernetes Security: An Evaluation Guide. PDF document. Available at: https://security.stackrox.com/container-security-evaluation-guide.html [Accessed 11 November 2020].

Sultan, S., Imtiaz, A. & Tassos, D. 2019. Container Security: Issues, Challenges, and the Road Ahead. IEEE Access 7: 52976–96. Available at: https://doi.org/10.1109/ACCESS.2019.2911732.

Wadsworth, R. 2016. Beyond Docker: Other Types of Containers. WWW document. Available at: https://www.contino.io/insights/beyond-docker-other-types-of-containers [Accessed 23 December 2020].

Weizman, Y. 2020a. Misconfigured Kubeflow workloads as a security risk. WWW document. Available at: https://www.microsoft.com/security/misconfigured-kubeflow-workloads [Accessed 30 December 2020].

Weizman, Y. 2020b Threat matrix for Kubernetes. 2020. WWW document. Available at: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/ [Accessed 30 December 2020].

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L. & Zhou, W. 2018. A Comparative Study of Containers and Virtual Machines in Big Data Environment. IEEE 11th International Conference on Cloud Computing, San Francisco.

**LIST OF FIGURES**

## LIST OF TABLES

## APPENDIX

### Appendix 1 VC scanning code for Image build in CI/CD

The code snippet below assumes that the CI/CD pipeline environment is based on Jenkins with the integration of Prisma Cloud Jenkins plugin for the vulnerability and compliance scanning.

```
node {
   stage('Preparation') {
       // for display purposes
       echo "Preparing"
   }
#Building of the container image based on ubuntu
   stage('Build') {
       // Build an image for scanning
       sh 'echo "FROM ubuntu:14.04" > Dockerfile'
       sh 'echo "MAINTAINER Aqsa Fatima <aqsa@twistlock.com>" >> Dockerfile
'
       sh 'echo "RUN mkdir -p /tmp/test/dir" >> Dockerfile'
       sh 'docker build --no-cache -t dev/ubun2:test .'
   }
#Scan of the image after build before pushing to the registry
    stage('Scan') {
    prismaCloudScanImage ca: '',
        cert: '',
        dockerAddress: 'unix:///var/run/docker.sock',
        ignoreImageBuildTime: true,
        image: 'ubun*',
        key: '',
        logLevel: 'info',
        podmanPath: '',
        project: '',
        resultsFile: 'prisma-cloud-scan-results.json'
}
#Publish result of the scan
    stage('Publish') {
        prismaCloudPublish resultsFilePattern: 'prisma-cloud-scan-
results.json'
    }
#Push image to registry after passing admission control
    stage('push to registry') {
        ...
        ...
    }

}
```