

UNIVERSITY OF
SCIENCE AND TECHNOLOGY
OF HANOI

ADMINISTRATION OF COMPUTER SYSTEM
FINAL PROJECT REPORT
TEAM 10

KUBERNETES CLOUD SECURITY

VIETNAM FRANCE UNIVERSITY

BI11-099

Nguyễn Viết Hoàng

BA11-062

Nguyễn Ngọc Linh

BA11-004

Lại Hải Anh

Table of Contents

I. INTRODUCTION	2
II. METHODOLOGY	3
III. KUBERNETES SECURITY PRACTICES.....	6
IV. DISCUSSION AND CONCLUSION.....	6

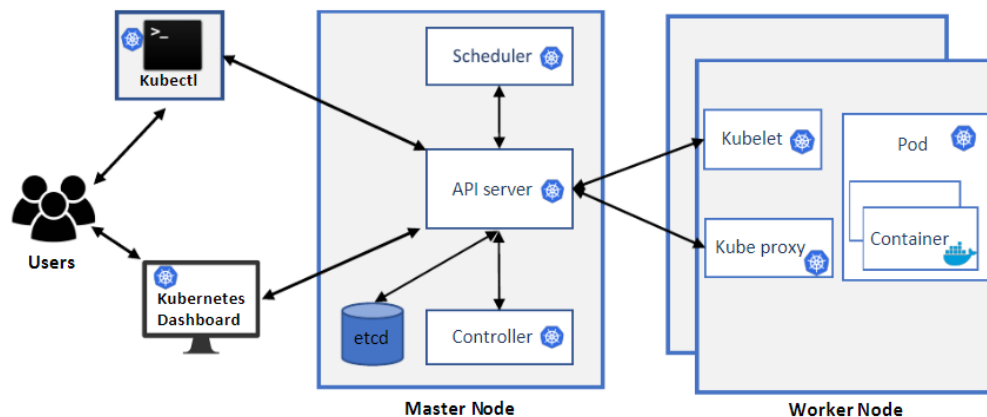
I. INTRODUCTION

1. Kubernetes:

Kubernetes is a system to run, manage, and coordinate containerized applications on a cluster of machines (1 or more) called a cluster. It can be configured to run applications and services as best suited as possible, adjust resources up or down, scale and update versions, and revoke updates when there are problems. .

Kubernetes follows the architecture of client-server architecture. Kubernetes users engage with the installation through the Kubernetes dashboard and the command-line tool 'kubectl.' The master node's primary role is to uphold the intended cluster configuration and oversee the worker nodes. Worker nodes, on the other hand, are employed for executing containerized applications within pods.

- Master node (control plane) include:
 - + **API server** (where API requests from users are received and processed) provides a RESTful interface to interact with Kubernetes and provides functionality for authentication, authorization, and access control to Kubernetes resources.
 - + **Etcd** is a distributed database used to store the state of a Kubernetes cluster. It stores and provides data such as information about pods, services, replicaset, and the status of other resources in the cluster.
 - + **Scheduler** decides which node the pod will be scheduled to run on cluster Kubernetes.
 - + **Controller manager** is responsible for coordinating various management tasks in the Kubernetes cluster.



- Worker node (kubelets):
 - + **Kubelet** is the link between Control Plane and Worker Node. Kubelet's main tasks include receiving requests from API Server to create, update and delete pods on the node, check the status of pods, monitor node resources and report back to Control Plane.

- + **Pod** can contain 1 or many different Containers. Containers in the same pod can interact with each other through localhost and access the same hosting.
- + **Kube proxy** is a network proxy. Responsible for determining how network requests to the pod are routed and forwarded. Implement a NAT mechanism to forward requests from the node's IP address to the pod's IP address.

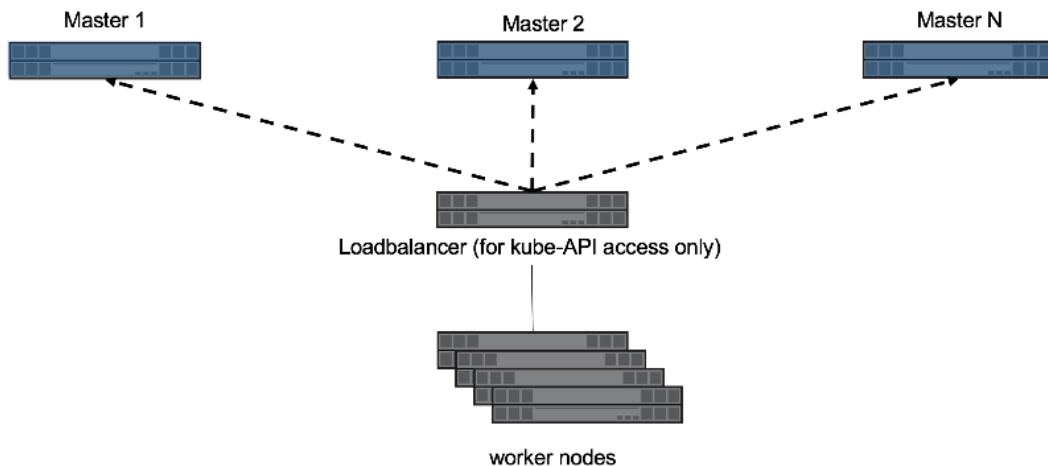
2. Kubernetes security:

While Kubernetes and containerization offer benefits like accelerated and scalable DevOps practices, they also introduce additional security risks. As the number of deployed containers increases, the attack surface expands, making it increasingly difficult to identify containers with vulnerabilities or misconfigurations. It is crucial to address these security risks proactively to maintain a robust and secure environment.

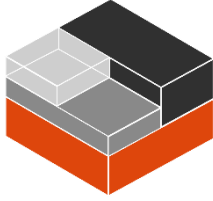





So, the solution for kubernetes security which we want to introduce here is Falco. Falco is a cloud-native security tool. By utilizing runtime insights, it offers almost instantaneous threat detection for workloads in the cloud, containers, and Kubernetes. Falco is capable of monitoring events from several sources, such as the Linux kernel, and enhancing them with metadata from other sources, such as the container runtime and the Kubernetes API server.

II. METHODOLOGY

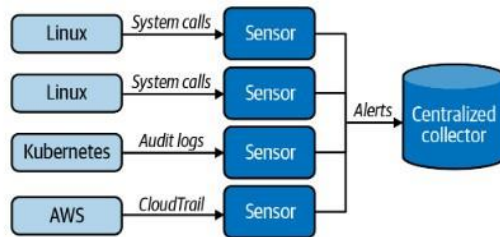
Scenario: K8s security for a small cloud storage provider (with architecture multiple master, compute)



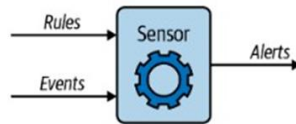
We setup for 20 nodes in total for small size service: 3 master nodes, 1 load balancer node, and 16 worker nodes in our network. Our tech stacks list is provided below.

<p>LXC for K8s nodes:</p> <ul style="list-style-type: none"> - LXC provides OS-level virtualization, which means sharing kernel with Host, thus resulting in lower system resource usage. - More manual work-arounds are need to make a functional LXC container K8s node than a full-powered VM K8s node (parsing kernel config, kernel modules, virtual device setup) 	
<p>Ansible to manipulate K8s nodes:</p> <ul style="list-style-type: none"> - Mass deployment and change over multiple workers with duplicated settings. - First we need to set up ssh and python on every host that we need to manage, and set up ssh key for convenience in deployment. 	 ANSIBLE
<p>Helm to manage package manager in K8s</p> <ul style="list-style-type: none"> - Easy and automated management (install, update, or uninstall) of packages for Kubernetes applications, and deploy them with just a few commands. - Offers Helm charts and repositories where you get everything necessary for deployment and its configurations. - Official Helm charts are up to date and maintained with new releases 	
<p>Longhorn to safely manage storage redundancy in K8s</p> <ul style="list-style-type: none"> - Highly available persistent storage (which is good for the platform that we will manage down below - nextCloud) - Easy incremental snapshots and backups - Cross-cluster disaster recovery - Longhorn supports ReadWriteMany (RWX) volumes, allowing a volume to be read and written by many nodes at the same time, which is a key requirement for certain types of applications 	
<p>ownCloud with oCIS to show high availability features and auto healing of K8s</p> <ul style="list-style-type: none"> - Later this service will run as cloud storage providers for multiple users - ownCloud Infinite Scale (oCIS) is a data platform providing tools to integrate, organize, share and govern data and metadata. The feature federated storage can be viewed here. 	
<p>Falco to watch for : security platform designed specifically for Kubernetes</p> <ul style="list-style-type: none"> - Falco can also be installed in a separated container (there is a Docker container image for Falco). 	

- Falco acts as a network of security cameras for your infrastructure, strategically positioned sensors that monitor activities and promptly notify you upon detecting malicious behavior. It relies on a collection of community-curated rules that define what constitutes harmful actions, offering the flexibility to customize or expand them to suit your specific requirements. While the alerts generated by your fleet of Falco sensors can technically remain on the local machines, in practice, they are often exported to a centralized collector for better management and analysis.



- Within the Falco sensor, there is an engine equipped with two inputs: a data source and a set of rules. As events flow in from the data source, the sensor diligently applies the rules to each event. When a rule finds a match with an event, an output message is generated as a result.



Falco in K8s:

- Falco is installed as DaemonSet (A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.)

Falco comes with a set of default rules that monitor the kernel for unexpected behavior such as:

- Privilege escalation using privileged containers
- Namespace changes using tools like setns
- Read/Writes to well-known directories such as /etc, /usr/bin, /usr/sbin, etc
- Ownership and Mode changes
- Unexpected network connections or socket mutations

Falco output support 3:

- Standard output
- File output
- Syslog output

<p>Falco plugins:</p> <ul style="list-style-type: none">- Can be created by using the Falco Go/C++ SDK to extend Falco.- Then the plugins can be distributed to registries. <p>Falco provides gRPC API (gRPC relies on HTTP/2, providing better performance and reduced latency, while REST uses HTTP/1.1)</p>	
---	--

III. KUBERNETES SECURITY PRACTICES

IV. DISCUSSION AND CONCLUSION