Threat Detection for Containers

Essentials to Securing Threats for Containerized Cloud-Native Applications

BORIS ZAIKIN
SENIOR SOFTWARE CLOUD ARCHITECT, IBM/NORDCLOUD GMBH

CONTENTS

Nowadays, containerized solutions are the de-facto standard in cloud-native application development. Tools like Docker, containerd, CRI-O, and Kubernetes are leading the charge in the world of container operation systems. Millions of development and architecture teams choose a container-based solution to help build their products, and distinguished cloud providers offer numerous services based on Kubernetes, Docker, and other container orchestration platforms.

With this increase in container adoption, security and threat management is more critical than ever. According to the Cloud Native Computing Foundation and the NSA/CISA guidelines:

- The majority of security issues and threats are found as 56% of developers are currently not even scanning their containers!
- Gartner projects that more than 70% of companies will be running containerized applications by 2023.
- A 2022 Red Hat report reveals that at least 59 percent of all Kubernetes security issues are related to misconfigurations.

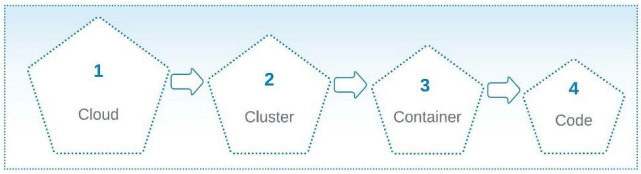
This Refcard will address key security and threat detection topics for containerized environments, including:

- An introduction to well-known container orchestration platforms like Docker, containerd, and CRI-O and their corresponding security challenges
- An overview of cloud-native security and compliance standards
- An introduction to threat detection tool selection criteria
- Essentials to threat detection for containers, including OWASP and Kubernetes security guidelines
- A secure cloud architecture example based on Azure, AWS, and Google Cloud

OVERVIEW OF THE CLOUD-NATIVE SECURITY APPROACH

Cloud-native security for containers represents four layers of security boundaries (also known as the 4Cs). These layers consist of code, container, cluster, and cloud. See the 4Cs represented in Figure 1 below.

Figure 1: The 4Cs (Cloud, Cluster, Container, Code) of cloud-native security



Security at the speed of light

Get cloud clarity with the only data-driven cloud security platform

We deliver end-to-end visibility and automated insight into risk across multicloud environments.

Our **Polygraph® Data Platform** uses data, analytics, and machine learning to automatically find the truth of risks known, and unknown.

100:1
reduction in critical alerts

80%
faster threat investigations

2-5
security tools consolidated

Book a demo today >

LACEWORK

CODE

As demonstrated in Figure 1, code is the deepest layer, as you not only enforce security within the code but also in the cloud, clusters, and container layers.

However, the code should not contain backdoors for vulnerabilities, for example:

- All communication should be done over TLS or mTLS
- Scan your dependencies and provide static analysis for your code
- Limit access to all well-known ports

CONTAINER

Containers and their container environment are fundamental parts of cloud-native security solutions. Now, applications are based not only on Docker but also on containerd, CRI-O, and other similar platforms.

There are several common security rules that can be applied to container platforms:

- Scan your container(s) for vulnerabilities and incorporate security scanning tools
- Use the principle of a least privilege
- Isolate users and container runtime
- Disable root permissions
- Introduce resource limits

CLUSTER

Container orchestrators are essential to security since they manage all application containers and services. Kubernetes is a widely used container orchestration platform, and its vulnerabilities are subject to a long list of security guidelines, specifically:

- RBAC and authentication strategies
- Application secrets management
- Network policies
- Ingress over TLS
- Pod Security Standards

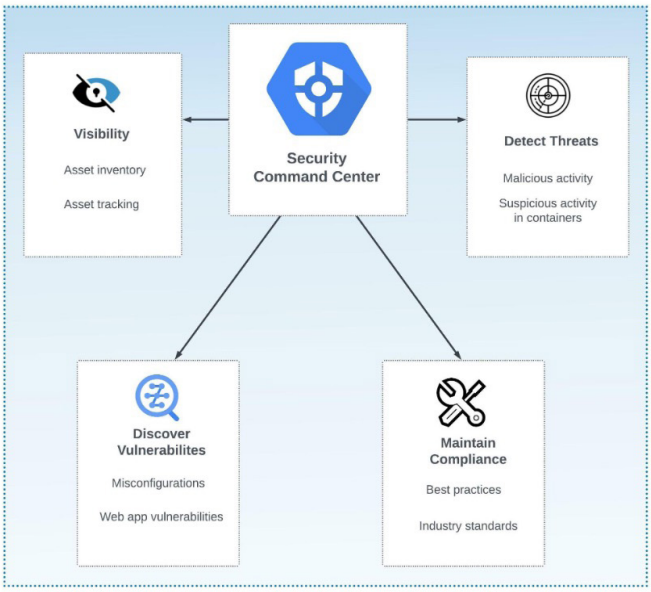
CLOUD AND INFRASTRUCTURE

All well-known cloud providers have security guidelines and resources to protect application clusters. For example, Azure has powerful platforms like Sentinel and Defender for Cloud that provide logic for threat detection within your infrastructure.

AWS' Security Hub is a cloud security management service that provides automated security verification for resources in AWS. All security verifications and checks are based on the AWS Foundational Security Best Practices standard.

Lastly, Google Cloud has security threat detection as a part of the Security Command Center. The Security Command Center is a centralized vulnerability and threat reporting service. It not only contains threat detection but also vulnerability scanning options.

Figure 2: Google Cloud Security Command Center



Additionally, open-source and third-party cloud security tools are powerful options to consider, especially when dealing with a multi-cloud or hybrid cloud environment.

Apart from cloud security, keep an eye on your infrastructure. If you use Kubernetes, then you need to focus on key aspects, like:

- Access to the Kube control plane, nodes, or public networks — and especially cluster networking
- Set up a permission strategy for your Kubernetes cloud provider
- Adopt the principle of least privilege
- Limit access to the etcd cluster and provide encryption over TLS

There's more to cover here as we learn about cloud and infrastructure security in the following section.

COMMON CLOUD SECURITY ARCHITECTURES

Cloud infrastructure is a fundamental part of cloud-native applications. In this section, let's explore some of the major cloud providers that offer full-stack security and threat detection options.

AWS SECURITY HUB

AWS Security Hub is a security service that contains options to aggregate, prioritize, and manage alerts from different AWS services.

AWS Security Hub can work with:

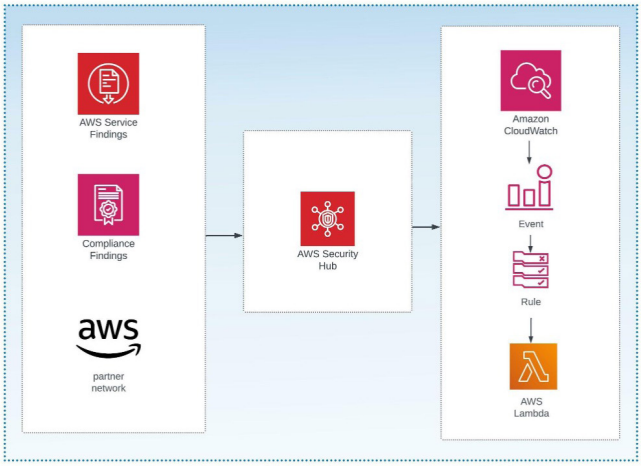
- Amazon S3 buckets
- S3 buckets with sensitive data
- Amazon Machine Images (AMIs)

- AWS service accounts
- Amazon EC2 instances

Additionally, the AWS Security Hub helps with automation by automating telemetry remediation and defining custom actions like emails, SMS, or even ticketing systems. The best advantage of AWS is its consolidated view of all AWS accounts across AWS regions.

Let's take a look at a simple AWS architecture with the AWS Security Hub detection system:

Figure 3: AWS Hub Security Detection System



The architecture above demonstrates the AWS Security Hub integrated with Cloud Watch. The Cloud Watch rule triggers the event that is then integrated with a Lambda function. Therefore, a particular function will handle the AWS Security Hub events.

The AWS Security Hub creation and configuration can be easily automated with the following Terraform module:

```
module "security_hub" {
    source        = "clouddrove/security-hub/aws"
    version       = "1.0.1"
    name          = "security-hub"
    security_hub_enabled = true

    #member account add
    enable_member_account = true
    member_account_id     = "123344847783"
    member_mail_id        = "example@mail.com"

    #standards
    enabled_standards = [
    "standards/aws-foundational-security-best-
practices/v/1.0.0",
    "ruleset/cis-aws-foundations-benchmark/v/1.2.0"
    ]
    #products
    enabled_products = [
```

CODE CONTINUES IN THE NEXT COLUMN

```
    "product/aws/guardduty",
    "product/aws/inspector",
    "product/aws/macie"
    ]
    }
```

With this module, developers can configure their member account, implement a ruleset (like: `aws-foundational-security-best-practices`), and enable AWS products.

AZURE DEFENDER FOR CONTAINERS

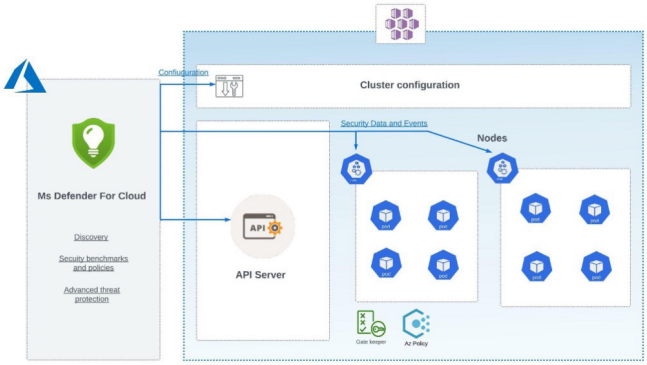
Azure Defender for Containers is a complex cloud-native security service. It contains container monitoring, container registry guiding services, and the AKS cluster security toolset.

Azure Defender contains AKS security hardening features. It allows you to set up Defender agents directly into the worker nodes and into the control plane so that you can run security and threat detection. The Defender has a large database of threats and security holes. As a result, the Defender has integrated the UI dashboard into the Azure Portal. In the dashboard, one can monitor their cluster security issues and remediate security checks. To check it out, visit the Azure Defender documentation.

When securing containers with Azure Defender, use the Agents' auto-provisioning capabilities. To do so, enable the log analytics and vulnerability assessment extensions in the auto-provisioning window.

Refer to the cloud deployment architecture of Azure Defender in Figure 4 below. It is deployed via the Defender profile, which is based on eBPF technology.

Figure 4: Azure Defender for AKS cluster



This architecture includes the following components:

- `DeamonSet` (`azuredefender-collector-ds-*`, `azuredefender-publisher-ds-*`) is a set of containers that collects security events and telemetry of the cluster environment
- Deployment (`azuredefender-publisher-ds-*`) apps that publish the collected data to the Defender's back end

Deploy the Defender for containers using the Bicep template:

```
 targetScope = 'subscription'

var enableSecurityCenterFor = [
  'KubernetesService'
]
resource securityCenterPricing 'Microsoft.
Security/pricings@2018-06-01' = [for name in
enableSecurityCenterFor: {
  name: name
  properties: {
    pricingTier: 'Standard'
  }
}]
```

As shown above, the Defender is still named as the security center. The template is quite easy and contains the section `enableSecurityCenterFor`. This section contains services, which Defender will be enabled for, and the section `securityCenterPricing`. This section is the Defender resource definition.

GOOGLE CLOUD APPARMOR

And last but not least is AppArmor — a security module based on the Linux Kernel security. It restricts processes that run in the operating system and is based on the security profile.

With a security profile, you can restrict network communication and activity, files, and storage. For Docker containers, you can use the default security profile and run the default Docker profile using the following command:

```
 docker run --rm -it debian:jessie bash -i
```

This code will restrict access to `/proc/sysrq-trigger` when read.

Create your own security profile using the example code below:

```
cat > /etc/apparmor.d/no_raw_net <<EOF
#include <tunables/global>

profile no-ping flags=(attach_disconnected,mediate_
deleted) {
  #include <abstractions/base>

  network inet tcp,
  network inet udp,
  network inet icmp,

  deny network raw,
  deny network packet,
  file,
  mount,
}
EOF
```

This code block restricts access to *raw* and *packet* networks and allows the access to the `tcp`, `udp`, and `icmp` protocols.

It is important to know the services and features offered by Azure, AWS, and Google Cloud because they allow us to build an efficient, secure containerized architecture.

ABOUT CONTAINER SECURITY AND THREAT DETECTION

All container engines are based on Container Runtime Engines (CREs). One of the most widely used CREs is Docker. However, Docker may not always be a good choice in terms of security and Kubernetes-ready solutions. There are other options like containerd or CRI-O.

Let's take a quick look at these options.

CONTAINERD

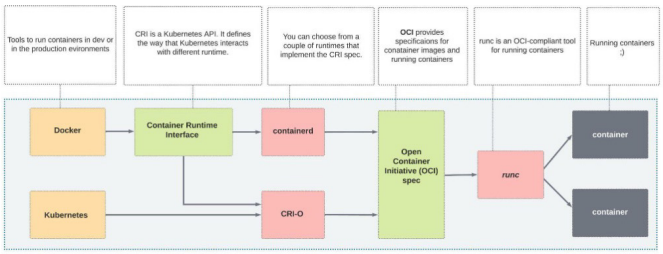
Since Docker is a monolithic tool that contains CLI, storage management, complex networking, permission logic, etc., these tools create a significant overhead and attack surface for exploits in Kubernetes.

Taking these issues into account, Docker extracted the container runtime into a separate project called containerd. Containerd is much smaller than Docker and offers low-level storage for managing image transfer logic. Now, containerd is part of the Cloud Native Computing Foundation (CNCF).

CRI-O

CRI-O goes even deeper than containerd, providing native and lightweight CRI. CRI-O doesn't contain any additional layers running in Kubernetes. Kubelet talks directly to the CRI-O runtime to pull images. See Docker, containerd, and CRI-O layers in Figure 5 below:

Figure 5: Docker, containerd, and CRI-O Layers



SECURITY ASPECTS OF DOCKER, CONTAINERD, AND CRI-O

When it comes to Docker, containerd, and CRI-O, a lot of attacks and threats to containerized environments are based on the same scenarios. For example:

- Spiking CPU, RAM, and disk usage to attack neighboring containers
- Using root privileges for a container to expose host networks, thus applying privileged flags

- Hardcoding secrets in images or in the Kubernetes configuration
- Linux Kernel vulnerabilities
- Using a malicious or fake image that contains a so-called man-in-the-middle attack

Since Docker has a monolithic architecture and a lot of layers, it is vulnerable to all of the scenarios mentioned above. For example:

- CVE-2020–8554 — A vulnerability that allows attackers to obtain access to the cluster by creating ClusterIPs service.
- Siloscape — Malware inside Windows containers. The Silocape creates a backdoor to the entire Kubernetes cluster, including sensitive data and CPU, GPU, and storage resources
- CVE-2018–15664 — Gives access to the Docker system with root permission

Containerd is a much more lightweight engine; it does not contain many layers from Docker. However, it has Linux features such as `audit_write`, `mknod`, `net_raw`, and `sys_chroot`. Containerd provides a backdoor to the attack surface like poisoning images or container escape for host network containers.

CRI-O doesn't contain layers like Docker because dev teams exclude all unnecessary Linux features that provide backdoors. However, CRI-O also contains some vulnerabilities that can result in an out-of-memory (OOM) condition within the *cgroup*, which results in containers and images without TLS connection.

Remember: Threats can always be reviewed in the CVE reports.

ESSENTIALS TO THREAT DETECTION FOR CONTAINERS

Teams can mitigate or completely remove threats and security vulnerabilities with the following core concepts:

DOCKER NETWORKING

Docker's networking is a complex part of the Docker infrastructure, and it is essential to understand how it works. It's important to know what Docker Network Drivers are, for example:

- Bridge
- Host
- Overlay

By default, one container network stack does not have access to another container. However, if you configure a bridge or host to accept traffic from any other containers or external networks, you can create a potential security backdoor for an attack. You can also disable inter-container communication using a set flag — `icc=false` within Docker daemon.

SET UP RESOURCE LIMITS

It's important to set up memory and CPU limits for your Docker container because Docker has a container that does not provide this option by default. This principle is an effective way to prevent DoS attacks.

For example, you can set up a memory limit to prevent your container from consuming all memory. The same applies to CPU limits. Additionally, there is an option to set up resource limits on a Kubernetes level — this will be covered in greater detail in the *Kubernetes Hardening Guidelines* section below.

AVOID SENSITIVE DATA IN CONTAINER IMAGES

This principle is quite important to move all sensitive data out of the container. You can use different options to manage your secrets and other sensitive data:

- Docker secrets allows you to store your secrets outside of the image
- If you run Docker containers in Kubernetes, use Secrets to store your passwords, certificates, or any other sensitive data
- Use cloud-specific storage for sensitive data — for example, Azure Key Vault or AWS Secrets Manager

VULNERABILITY AND THREAT DETECTION TOOLS

Vulnerability scanning tools are an essential part of detecting images that may contain security pitfalls. Moreover, you can also integrate properly selected tools into the CI/CD process. Third-party vendors and some common open-source tools offer this sort of functionality. Some examples of these open-source tools can be found in the *About Container Security and Threat Detection* section.

CONTAINER REGISTRIES

To protect your images, create an additional security layer and use images from protected registries. Here are a few examples of open-source registry platforms:

- Harbor — An open-source registry with integrated vulnerability scanning. It is based on security policies that apply to Docker artifacts.
- Quay — An open-source image registry that scans images for vulnerabilities. Powered by RedHat, Quay also offers a standalone image repository that allows you to install and use it internally in your organization. Below, we will dive into how it scans for vulnerabilities within containers.

PRINCIPLE OF LEAST PRIVILEGE

This principle means that you should not execute containers using admin users. Instead, you should create users that have admin access and can only operate with this particular container. Groups can also add users there. Read more about it in the Docker Engine Security

documentation. Below is an example of how to create the user and group.

```
RUN groupadd -r postgres && useradd --no-log-init -r
-g postgres postgres
USER postgres
```

Also, alongside creating users and groups, please be sure to use the official verified and signed images. To find and check images, use `docker trust inspect`. Find more options and tools in the section below, *Threat Detection Tool Selection Criteria for Containers*.

LINUX SECURITY MODULE

To enforce security, implement the default Linux security profile and do not disable the default one. For Docker, you can use AppArmor or Seccomp. Security context rules in Kubernetes can also be set up with `allowPrivilegeEscalation`. This option controls the privileges that the container possesses. Also, the `readOnlyRootFilesystem` flag sets the container root filesystem to read-only mode.

STATIC IMAGE VULNERABILITY SCANNING

Now that we know how threat detection tools can work together and the strategies that we can use, let's define what it means to adopt static image vulnerability scanning, secret scanning, and configuration validation.

Static security vulnerability scanning is based on the Open Container Initiative (OCI) format. It validates and indexes the images against well-known threats and vulnerability information sources, like CVE Tracker, Red Hat security data, and Debian Security Bug Tracker.

Static security vulnerability scanning mechanisms can be used for scanning several sources, like:

- Container image
- Filesystem and storage
- Kubernetes cluster

Static image vulnerability scanning can also scan misconfigurations, secrets, software dependencies, and generate the Software Bill of Materials (SBOM). An SBOM is a combination of open-source, third-party tools and components within your application that contains license information of all components. This information is important for quick identification of security risks.

Below, we've included a list of open-source tools that cover the logic explained above. This is a representation of only a few of the many tools that can be used:

- Clair — A security vulnerability scanning tool with an API for development integration purposes. It also creates your own divers to extend and customize Clair functionality. Clair has several Indexer, Matcher, Notifier, or combo models.
- Trivy — A security container scanner based on the CVE threat database. Trivy can be installed on your PC or in the Kubernetes nodes, using `npm`, `apt-get`, `brew`, and other package managers, like:
  - *apt-get install trivy*
  - *yum install trivy*
  - *brew install aquasecurity/trivy/trivy*

To execute image scanning, run the following command:

```
$ trivy image app-backend:1.9-test
```

Another key to static image vulnerability scanning is the Security Content Automation Protocol (SCAP). SCAP defines security compliance checks for your infrastructure based on Linux.

OpenSCAP is a tool that includes complex security auditing options. It allows you to scan, edit, and export SCAP documents, consisting of the following components and tools:

- **OpenSCAP Base** – For vulnerability and configuration scans
- **`oscap-docker`** – For compliance scans
- **SCAP Workbench** – A graphical utility to facilitate the execution of typical OpenSCAP tasks

Below, you can see an example on how to run the validation process of the SCAP content:

```
oscap ds sds-validate scap-ds.xml
```

CONFIGURATION VALIDATION

Static image validation is an important part of the threat detection process. However, static image scanning cannot detect the misconfiguration in the YAML and JSON configuration, and it may cause outages in complex YAML configs. Therefore, having an easy and automated method involves configuration validation and scanning tools like Kubeval. We can introduce configuration validation that can solve issues with static configuration and simplify the automation process.

As an example, we will incorporate Kubeval, an open-source tool used to validate one or more Kubernetes configuration files, often used locally as part of a development workflow and/or within CI/CD pipelines.

To run the validation, use the following example:

```
$ kubeval my-invalid-rc.yaml
WARN - fixtures/my-invalid-rc.yaml contains an
invalid ReplicationController - spec.replicas:
Invalid type. Expected: [integer,null], given:
string
$ echo $?
1
```

SECRETS SCANNING

The main goal of secrets scanning is looking into the container images, infrastructure-as-code files, JSON log files, etc. for hardcoded and default secrets, passwords, AWS access IDs, AWS secret access keys, Google OAuth Key, SSH keys, tokens, and more.

MANAGEMENT CONSOLE AND SENSOR AGENTS

A management console is a UI tool that builds a security infrastructure overview and provides vulnerability and threats reports. On the other hand, sensor agents are a tool that scans cluster nodes and gathers security telemetry. Thus, sensor agents report telemetry to the management console.

For example, to install sensor agents within the AKS cluster, adhere to the following:

```
helm repo add deepfence https://deepfence-helm-
charts.s3.amazonaws.com/threatmapper
helm show readme deepfence/deepfence-agent
helm show values deepfence/deepfence-agent

# helm v2
helm install deepfence/deepfence-agent \
    --name=deepfence-agent \
    --set managementConsoleUrl=x.x.x.x \
  --set
deepfenceKey=C8TtyEtNB0gBo1wGhpeAZICNSAaGWw71B
SdS2kLELY0
```

To install the management console within your AKS cluster, use the following command:

```
 helm repo add deepfence https://deepfence-helm-
charts.s3.amazonaws.com/threatmapper

helm install deepfence-console deepfence/deepfence-
console
```

The installation completes two operations:

- Downloads the Helm package from: https://deepfence-helm-charts.s3.amazonaws.com/threatmapper
- Installs the Helm package directly to the cluster namespace.

Both commands are quite simple and can be easily integrated into an IaC process.

KUBERNETES HARDENING GUIDELINES

Because Kubernetes is the most popular orchestration platform, it requires frequent security tweaks. Thus, the National Security Agency (NSA) created the K8s Hardening Guidelines. Let's explore the most common security rules according to the NSA's hardening guidelines:

NETWORKING AND NETWORK POLICIES

Understanding how the Kubernetes networking model works is critical to setting up proper network communication between pods and pretending to create open ports or direct access to the nodes. The Network Policy can help organize this communication.

SECURE INGRESS AND EGRESS TRAFFIC TO YOUR POD

When securing Ingress and Egress traffic to your pod, it's helpful to deny all traffic and then start opening the ports one by one. Using a service mesh like Istio can also be helpful since it adds additional service layers, automates traffic, and helps with monitoring. However, be careful when using the service mesh as it may add further complexity.

STRONG AUTHENTICATION AND AUTHORIZATION POLICIES

- **Enforce Transport Layer Security** — Transport Layer Security (TLS) should be used for communication between Kubernetes cluster services.
- **Adopt RBAC** and the **principle of least privilege**.
- **Restrict access to Kubelet** — Enable authentication and authorization to use this tool; only admins should have access to Kubelet.

USE LOG AUDITING

Enable Kubernetes control plane auditing to provide full information to the security team. Often, this can work by forwarding logs to various monitoring platforms.

VALIDATE AND CHECK FOR MISCONFIGURATIONS

To check for misconfigurations, adopt an automation approach and a configuration scanning tool. The CNCF project Open Policy Agent can be used to create security policies to prevent creating misconfigurations.

For example, deny running the containers with the latest tags:

```
package Kubernetes

import data.kubernetes
import data.lib as l

check03 := "K8S_03"

exception[rules] {
      make_exception(check03)
      rules = ["usage_of_latest_tag"]
}

deny_usage_of_latest_tag[msg] {
       ubernetes.containers[container]
      [image_name, "latest"] =  ubernetes.split_
image(container.image)
```

CODE CONTINUES ON THE NEXT PAGE

```
        msg =  ubern("%s: %s in the %s %s has an
image, %s, using the latest tag. More info: %s",
[check03, container.name,  ubernetes.kind, image_
name,  ubernetes.name, l.get_url(check03)])
}
```

More examples of policies can be found in the Open Policy Agent policies.

IMPLEMENT INTRUSION DETECTION AND A SECURITY INFORMATION SYSTEM

Intrusion detection systems are an important part of Kubernetes security. These systems check the host's behavior for any dubious activity and/or anomaly traffic, signaling alerts for the administrators.

THREAT DETECTION TOOL SELECTION CRITERIA FOR CONTAINERS

Tools and platforms are a fundamental part of the threat detection foundation. There are many native cloud providers and open-source tools that contain threat detection options, including services such as:

- Static image vulnerability scanning
- Configuration validation
- Secrets scanning

The question is how to choose the proper security tool for your architecture. Let's review several requirements and strategies for selecting the proper threat detection tool:

THREAT DETECTION TOOL CRITERIA FOR CONTAINERS	
CLOUD CRITERIA	**RECOMMENDATION**
Cloud provider (e.g., AWS, Google Cloud, and Azure)	Many cloud providers offer native solutions, providing easy access for users, though additional security tooling is recommended for end-to-end visibility and protection.
On-premises/custom cloud	On-premises (or bare metal) requires more custom solutions. One of the best strategies here is to combine tools that cover static image vulnerability scanning and configuration validation.
Hybrid (cloud and on-premises)	Hybrid scenarios may include the following strategies: • Partially cover threat detection with existing cloud security services and add tools for secret scanning and configuration validation to reinforce the security architecture. • Save money by selecting a set of open-source tools for static image vulnerability scanning, configuration, and secret validation.

Let's have a look at the following example of an on-premises hybrid scenario where we build a model architecture with a combination of open-source tools and services.

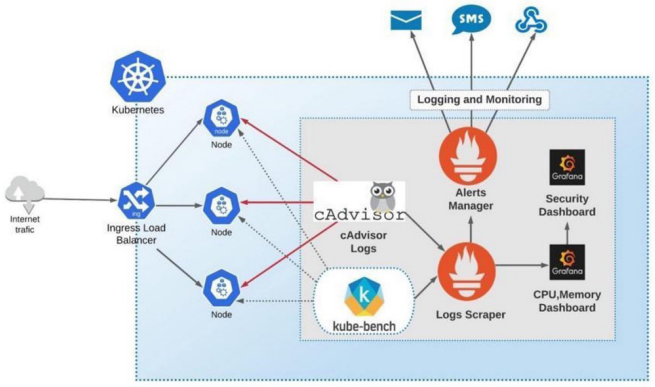
THREAT DETECTION ARCHITECTURE EXAMPLE

As an example of using open-source security tools, we will kick off the example with a Kubernetes cluster. It can be deployed using a custom cloud provider (or on-premises environment).

Our example architecture contains the following open-source platforms:

- Cadvisor — Used for detecting vulnerabilities.
- Grafana — Optimal for building and configuring dashboards and alerts by importing all components together.
- Prometheus — A powerful and open-source option for monitoring CPU, GPU, memory, images, and other metrics.
- Kube-bench — Detects threat and security issues for your Kubernetes cluster. Its security check is based on the CIS Kubernetes Benchmark.

Figure 6: Example threat detection architecture



To configure the **kube-bench**, use YAML files. For example, the YAML file is listed below. It is used to run the validation process against a single pod:

```
---
apiVersion: batch/v1
kind: Job
metadata:
  name: kube-bench
spec:
  template:
    metadata:
      labels:
        app: kube-bench
    spec:
      hostPID: true
      containers:
        - name: kube-bench
```

CODE CONTINUES ON THE NEXT PAGE

```
          image: docker.io/aquasec/kube-bench:v0.6.8
          command: ["kube-bench"]
          volumeMounts:
            - name: var-lib-etcd
              mountPath: /var/lib/etcd
              readOnly: true
            - name: var-lib-kubelet
              mountPath: /var/lib/kubelet
              readOnly: true
            - name: var-lib-kube-scheduler
              mountPath: /var/lib/kube-scheduler
              readOnly: true
            - name: usr-bin
              mountPath: /usr/local/mount-from-host/
bin
              readOnly: true
            - name: etc-cni-netd
              mountPath: /etc/cni/net.d/
              readOnly: true
            - name: opt-cni-bin
              mountPath: /opt/cni/bin/
              readOnly: true
…
      restartPolicy: Never
      volumes:
        - name: var-lib-etcd
          hostPath:
            path: "/var/lib/etcd"
        - name: var-lib-kubelet
          hostPath:
            path: "/var/lib/kubelet"
…
```

Let's save this document to the file `job.yaml` and run it with the apply command:

```
$ kubectl apply -f job.yaml
```

You can find the complete version here.

CONCLUSION

In this Refcard, we've presented a complete guide to threat and vulnerability detection mechanisms for containerized cloud-native applications. From security baselines for cloud-native environments to hardening guidelines for Kubernetes, this Refcard provides everything needed to start building secure containerized architectures for cloud-native applications.

WRITTEN BY BORIS ZAIKIN,
SENIOR SOFTWARE CLOUD ARCHITECT, NORDCLOUD/IBM

I'm a Certified Software and Cloud Architect who has solid experience designing and developing complex solutions based on the Azure, Google, and AWS clouds. I have expertise in building distributed systems and frameworks based on Kubernetes and Azure Service Fabric. My areas of interest include enterprise cloud solutions, edge computing, high load applications, multitenant distributed systems, and IoT solutions.

DZone

600 Park Offices Drive, Suite 300
Research Triangle Park, NC 27709
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.