CyberSecurity MAGAZINE

Security and Observability for Cloud Native Platforms Part 2

cybersecurity-magazine.com/security-and-observability-for-cloud-native-platforms-part-2/

David Soldani, Hami Bour and Saber Jafarizadeh September 9, 2022

Security and Observability for Cloud Native Platforms Part 1

Threat Landscape and Overall Security Framework

There are several possible routes to attacking a containerized deployment, and one way to map them is to think of the potential attack vectors at each stage of a container's life cycle.

The life cycle starts with the application code written by a developer. This code, as well as the third-party dependencies on which it relies, may contain flaws known as *vulnerabilities*. There are thousands of vulnerabilities that have been published, and if they exist in an application, an attacker may have the ability to exploit them. Examples of vulnerabilities are *secret exposure* and application (including CNF microservices) traffic in *plane text*, which can be intercepted and altered.

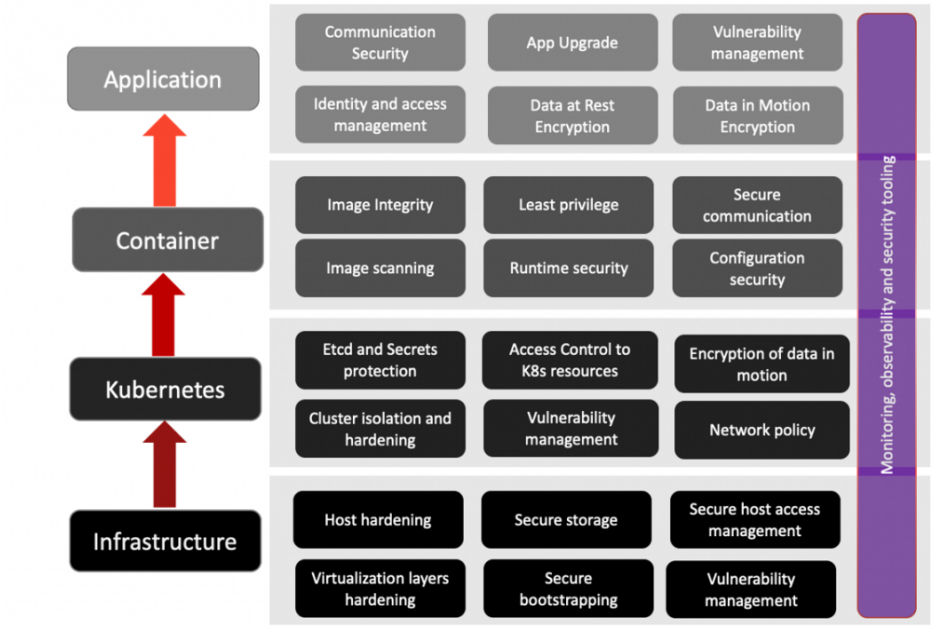
After the code has been created, it gets built into a *container image*. When you are configuring how a container image will be built, there are plenty of opportunities to introduce vulnerabilities that can be exploited later to attack the running container. These include configuring the container to run as the *root user*, giving it the privilege to escape the container to the host through vulnerabilities or volume mount.

Other potential risks are unauthorised access via Kubelet API, K8s API or proxy, and unauthorised access to machines/VMs and etcd API.

Moreover, on the workers node and master node, the unencrypted traffic to and from a Pod could be intercepted, modified, and injected.

Securing the cloud native platform requires to protect the physical and virtual infrastructure, as well as the container orchestrator, container, and application (including network microservices, e.g., in the case of Open vRAN).

An outline of the corresponding security mechanisms, monitoring, observability, and security tooling is depicted in the figure below.



Let's take a closer look at how the infrastructure security, security of workload deployment at each stage, and security within DevOps process are achieved. The next sections will then focus on the main aspects of Kubernetes security.

Infrastructure Security

The infrastructure of our system can be protected by deploying the following safeguards and security measures, but not be limited to.

Hardening Virtualisation Layers

The Peripheral Component Interconnect (PCI) passthrough feature enables you to access and manage hardware devices from a virtual machine. When PCI passthrough is configured, the PCI devices function as if they were physically attached to the guest operating system. To reduce the risk of direct memory access and hardware infection, PCI passthrough should be disabled by default. Furthermore, should you use an emulator, such as, e.g., Quick EMUlator (QEMU), for imitating other operating systems, this process should also be hardened by using mandatory access controls like sVirt, SELinux, or AppArmor, and ensure that *iptables* have the default policy, filtering network traffic.

Secure Bootstrapping

An automated provisioning process should be employed for all cloud nodes, including compute, storage, network, service, and hybrid nodes to ensure that nodes are provisioned consistently and correctly, and facilitate security patching, upgrading, bug fixing, and other critical changes.

Host Hardening

It comprises the *choice of operating system* (utilization of modern immutable Linux distribution specifically designed for containers; latest vulnerability fixes, self-update to newer versions, as well as support of newer technologies such as extended Berkeley Packet Filter (eBPF), described in the following sections; ensure that the root filesystem is locked and cannot be altered by applications; control of shared libraries from the host operating system and impact on containers; ensure that application developers do not rely on a specific version of the operating system or kernel, for updating the host operating system as needed); *non-essential processes* (remove any nonessential processes

that may be running by default; utilize an immutable Linux distribution optimized for containers so that non- essential processes are eliminated; only run additional processes and applications as containers); *host-based firewalling* (configure the host itself with local firewall rules to restrict which IP address ranges and ports are allowed to interact with the host; use Kubernetes network plug-ins, like Weave Net, Kube-router, and Calico, which include the ability to apply network policies); *latest best practices* (use base best practices that are well-documented online and are available for free, such as the Centre for Internet Security, CIS Benchmarks; and secure your host operating system by using an up-to-date list of CIS Benchmarks on their website).

Cluster Hardening

It relates to *securing the K8s datastore and encrypt secrets at rest* (secure etcd using x509 Public Key Infrastructure (PKI) for mutual Transport Layer Security (mTLS) of data in transit and access control; configure etcd with Certificate Authority (CA) for generating client credentials; configure the K8s API server (client certificate, key, and CA) so it can access etcd; use network-level firewall rules to restrict etcd access; use etcd datastore for K8s only in a separated etcd cluster; rewrite all secrets to trigger their encryption within etcd; maintain etcd backups and secure them); *securing K8s API server and rotate credentials* (create the required keys and certificates and distribute them to the other Kubernetes cluster components; set short life- times on any TLS certificates or other credentials and automate their rotation; include any service tokens used in external integrations or as part of the cluster boot- strap process; rewrite all secrets to trigger their encryption with the latest keys; when using a Key Management Server (KMS) provider, the Key Encryption Key (KEK) can be rotated without requiring re-encryption of all the secrets; restricting cloud metadata API access); *using Role Based Access Control (RBAC) and upgrade K8s frequently* (create separate user accounts for each user and use RBAC to grant users the minimal access they need to per- form their role, following the principle of least privilege access; use groups and roles, rather than assigning RBAC permissions to individual users; periodically review/audit the user privileges across the cluster to verify they are correct and up to date; integrate RBAC with external authentication providers; regularly upgrade your clusters, and have in place the ability to urgently upgrade if a severe vulnerability is discovered, e.g., Kubernetes-announce email group); and *enabling auditing and restrict access to alpha or beta features* (enable Kubernetes audit logging and archiving audit logs on a secure service; actively monitor Kubernetes audit logs and generate alerts based on suspicious activity using own custom tooling or third-party solutions; logs the request metadata (user, timestamp, resource, verb, etc.) but not the request details or response body; make sure that all alpha and beta features you do not intend to use are disabled).

Structured and Unstructured Data Protection

Software should be a unified security control centre that leverages metadata and metrics to perform security assessments of database configurations and users, discover sensitive data within databases, mask data in development and test database instances, and monitor and audit database activities. Actionable intelligence – about database users, database configuration, database content, and database activity – enables database security, data governance, and data protection. Leveraging intelligence available from metadata associated with a database and its configuration, the software can assess security risks, discover & conceal sensitive data, monitor database activity for anomalies, and audit database activity for compliance, such as using the EU's General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA), which share the same basic goal of protecting personal data.

Secure Workload Deployment

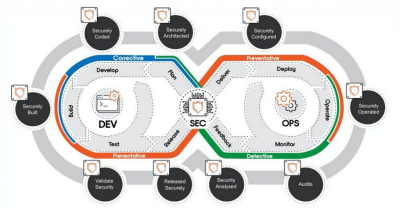
The three stages of **workload deployment** – build, deploy and runtime – and the most important security aspects at each **stage** – security by design, infrastructure controls, deployment controls, runtime controls, observability, and threat detection – are:

- *Build*: image scanning, securing CI/CD pipelines, secrets management, securing the host OS, and securing the workload access to the host.
- *Deploy*: cluster hardening, perimeter firewalls, security groups, admission controllers, and exposing services.
- *Runtime*: network policy, application layer policy, observability, encryption, auditing and compliance, and threat detection.

DevOps Security

DevSecOps stands for development, security, and operations. It is a new approach to secure the DevOps process. It integrates security as a shared responsibility throughout the entire software development lifecycle, as shown in the figure below. Namely:

1. *Securely architected*: security requirements; threat modelling; secure design patterns.
2. *Securely coded*: secure coding guidelines; code reviews; SAST.
3. *Securely built*: vulnerability scanning; secrets management; CIS benchmarks; Security Configuration Assessment (SCA).
4. *Validate security*: run-time scanning; penetration (PEN) testing; malware detection; configuration drift.
5. *Released security*: secure delivery of images; image integrity verification.
6. *Securely configured*: security deployment design; system hardening; security configuration check.
7. *Securely operated*: identity management; access control; run-time monitoring of network; incident detection and response; security orchestration; vulnerability management.
8. *Audits*: security log; performance log; risk assessment; compliance audit.
9. *Security analysed*: threat intelligence; security operation optimization; product security improvement; SIEM/SOC.



www.servian.com

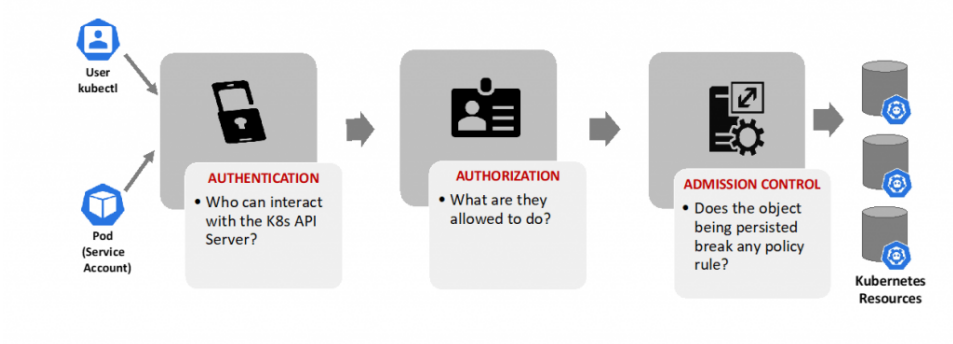
Access Control to Kubernetes Resources

Users access the Kubernetes API via *kubectl*, client libraries, or by making REST requests. Users, Groups and Kubernetes service accounts can be authorized for access to the API server. When a request reaches the API, it passes through several stages, as illustrated in the following diagram:

- *Authentication*: authentication modules consist of client certificates, password, and plain tokens, bootstrap tokens, and JSON Web Tokens (used for service accounts). Multiple authentication modules can be specified. In such scenario, each one is tested sequentially until one of them succeeds.

- *Authorization*: Kubernetes supports multiple authorization modules, such as ABAC mode, RBAC Mode (mostly recommended), and Webhook mode. When an administrator creates a cluster, they configure the authorization modules that will be utilized in the K8s API server.

- *Admission Control*: admission controllers are software modules that act on requests that create, modify, delete, or connect to (proxy) an object, to reject or modify the requests. Multiple controllers can be configured. If any admission controller module rejects the request, then the request is immediately blocked.



Network Policy

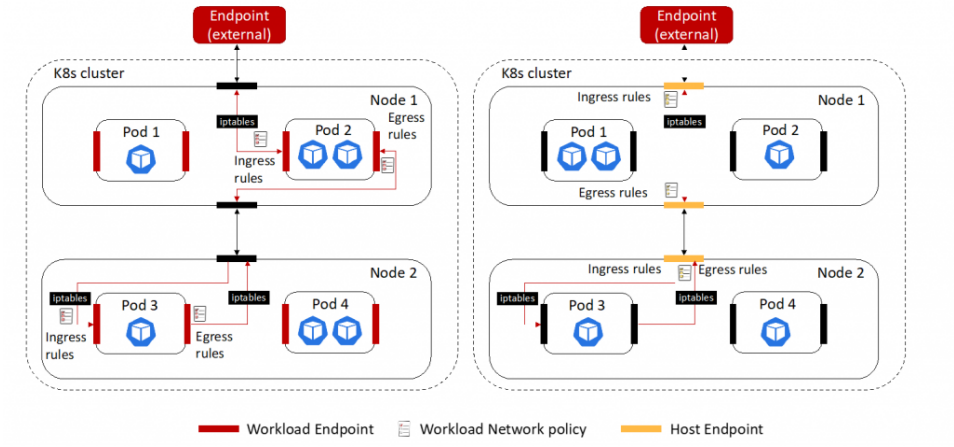
Policies can be applied to any kind of endpoint (pods/containers, VMs, and/or to host interfaces). A policy rule applies to ingress, egress, or both interfaces, and comprises:

- *Actions*: allow, deny, log, pass. (With Calico you cannot deny traffic, only *allow*.)

- *Source and destination match criteria*: ports (numbered, ports in a range, and Kubernetes named ports); protocols (TCP, UDP, ICMP, SCTP, UDPlite, ICMPv6, protocol numbers: 1-255); HTTP attributes (if using Istio service mesh); ICMP attributes; IP version (IPv4, IPv6); IP or CIDR; Endpoint selectors (using label expression to select pods, VMs, host interfaces, and/or network sets); Namespace selectors; Service account selectors.

- *Optional packet handling controls*: disable connection tracking, apply before DNAT, apply to forwarded traffic and/or locally terminated traffic.

The Network Policy capabilities for a *workload endpoint* and a *host endpoint* depicted in the figures below are related to Calico CNI, which extends the native K8s Network Policy functionality.



Calico network policies apply to endpoints. In Kubernetes, each pod is a **Calico endpoint**. However, Calico can support additional types of endpoints, which are classified as:

- *Workload endpoint*: such as a Kubernetes pod or OpenStack VM.
- *Host endpoints*: an interface or group of interfaces on a host.

A **host endpoint** (HostEndpoint) represents one or more real or virtual interfaces attached to a host that is running Calico:

- It enforces Calico policy on the traffic entering or leaving the host's default network namespace through those interfaces.
- A host endpoint with interfaceName: * represents all host's real or virtual interfaces.
- A host endpoint for one specific real interface is configured by interfaceName: <name-of-that-interface>, or example interfaceName: Eth0, or by leaving interfaceName empty and including one of the interface's IP in expectedIPs.
- Each host endpoint may include a set of labels and list of profiles that that will be used by Calico to apply policy to the interface.

Calico supports two types of Network Policies:

- *Namespaced (workload) policy*: it is a namespaced resource that applies to pods/containers/VMs in that namespace
- *Global network policy*: it is a non-namespaced resource and can be applied to any type of endpoint (pods, VMs, host interfaces) independent of namespace.

For workload network policies, the **service account** of the source or destination Pod can be used as a match criterion. The service account can be associated with roles (employing role bindings) and labels can be attached to it.

The Global network policy can be applied to traffic before DNAT, forwarded traffic and disable connection tracking for packets where this policy is applied.

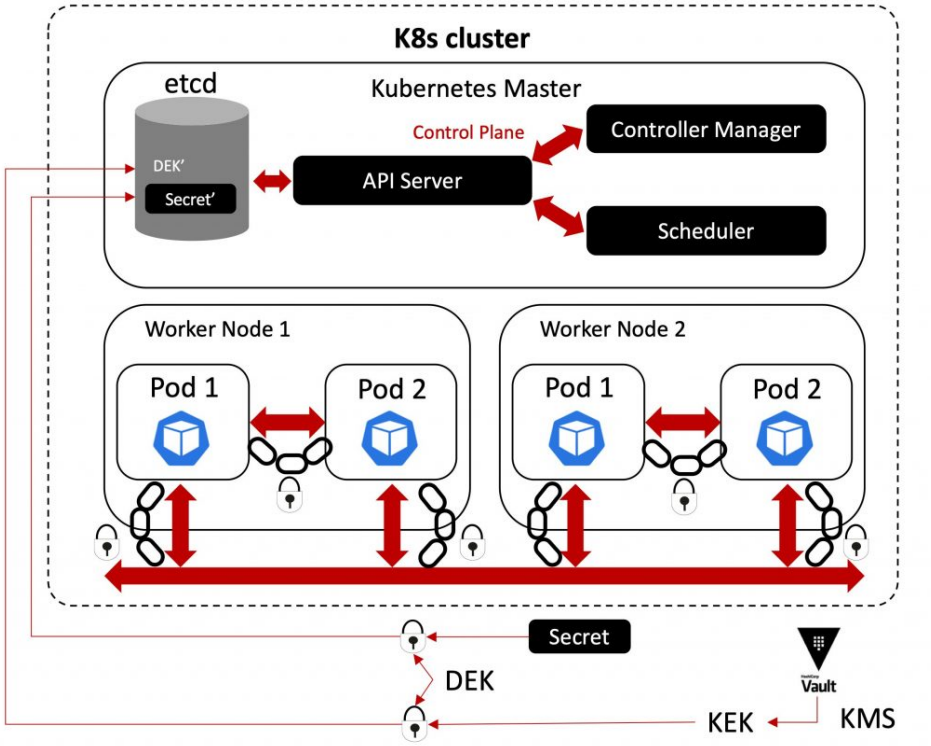
Encryption of Data at Rest and in Motion

For providing Pod to Pod security there are three options:

1. Use encryption at the App level **mTLS**, e.g., using Istio Service Mesh.

2. Use encryption at a lower infrastructure level using protocols as **IPSec**, which is included as a standard option in the Linux Kernel, but it introduces management complexity for setting up and maintaining secure network connections.

3. Use **WireGuard**, e.g., Calico, which is a kernel alternative to IPSec that aims to “be faster, simpler, leaner, and more useful”.



For data at rest, our recommendations are as follows:

- *Use Secrets*: Use Secrets, enable encryption, and avoid including the confidential data in your application code.

- Use Envelop Encryption: Each Secret is encrypted with a dynamically generated Data Encryption Key (DEK). The DEK is then encrypted using a Key Encryption Key (KEK) provided by an external Key Management Server (KMS), such as HashiCorp’s Vault, and the encrypted DEK (DEK’) is stored in etcd alongside the encrypted Secret (Secret’). The KEK is always hosted by the KMS as the central root of trust and is never stored within the cluster.

The figure above represents how envelop encryption could be implemented.

CNF Certification

The CNCF provides a list of essential tests for obtaining a CNF Certification. The List of Essential Tests for Security allow you to mitigate the risks related to:

- Container socket mounts

- External IPs
- Privileged containers
- Privilege escalation
- Symlink file system
- Sysctls
- Application credentials
- Host network
- Service account mapping
- Ingress and Egress blocked
- Insecure capabilities
- Non-Root containers
- Host PID/IPC privileges
- Linux hardening
- Resource policies
- Immutable File Systems
- HostPath Mounts
- SELinux options

Each test lists a general overview of what the test does, a link to the test code for that test, and links to additional information when relevant/available.