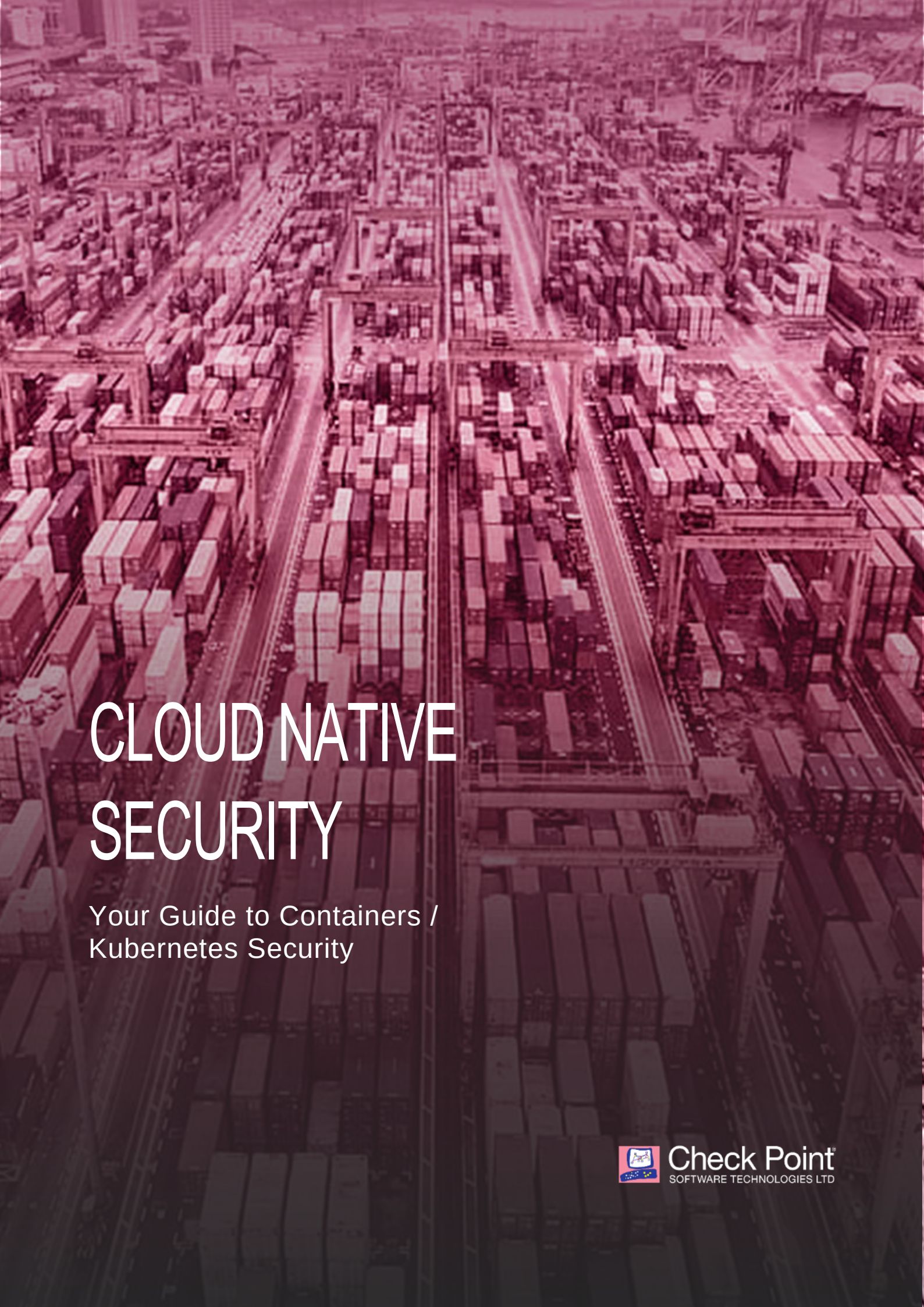


CLOUD NATIVE SECURITY

Your Guide to Containers / Kubernetes Security

Check Point
SOFTWARE TECHNOLOGIES LTD

ABSTRACT

Moving to the cloud is more than a technical transition to new platforms. It's part of an organization's growth strategy, with the transformation digitalizing its operations. It brings new technologies together with new security risks.

This paper helps provide a basic understanding of a modern approach to microservices and container technologies including Kubernetes. It explores the challenges they raise together with possible solutions and introduces the Cloud-Native Security platform.

AUDIENCE

This paper is intended for a technical audience, including security specialists interested in gaining a quick understanding of recent security technological trends, CI/CD pipelines, DevSecOps, containerization, and cloud transformation.

Readers should be familiar with the basic concepts of virtualization, networks, and have a good understanding of security design.

TABLE OF CONTENTS

INTRODUCTION TO CLOUD-NATIVE TECHNOLOGIES

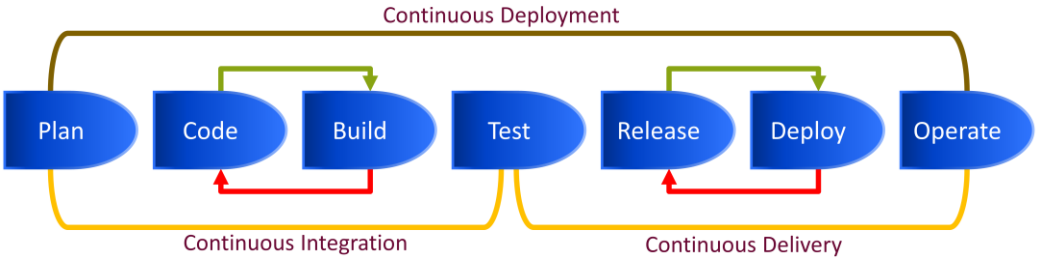
Cloud-native applications use containers, microservices, serverless functions, and code representing infrastructure. Cloud-Native technologies accelerate software development and enable world-class enterprises to create reliable, easily manageable, transparent applications with dynamic scalability.

CI/CD and DevOps

For mobile applications, web development, games, and e-commerce, the consecutive stages of design, assembly, and testing should be as fast as possible. Instead of months, every step now takes hours or even minutes.

For projects requiring a team of programmers, testers, and managers, where code changes need to be made several times a day, a CI/CD approach is recommended. It is based on Continuous Integration, Continuous Delivery, and Continuous Deployment pillars.

DevOps is a software development methodology focused on extremely active interaction and integration between a team of programmers (Development) and testers and admins (Operations) who synchronously serve a common service/product. DevOps is designed to effectively organize the creation and updating of software products and services and is based on the following stages:



Let's look at a few modern technologies and approaches that contribute to the popularity of DevOps.

Microservices

According to Wikipedia, Microservices is a software development technique, a variant of the service-oriented architecture (SOA) structural style, that arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and protocols are lightweight.

This provides multiple benefits: Modularity, scalability, integration of heterogeneous and legacy systems, and distributed development.

Example (https://microservices.io/patterns/apigateway):

Let's imagine you are building an online store that uses the Microservice architecture pattern and that you are implementing the product details page. You need to develop multiple versions of the product details user interface:

- HTML5/JavaScript-based UI for desktop and mobile browsers - HTML is generated by a server-side web application
- Native Android and iPhone clients - these clients interact with the server via REST APIs

In addition, the online store must expose product details via a REST API for use by 3rd party applications.

A product details UI can display a lot of information about a product. For example, the Amazon.com details page for POJOs in Action displays:

- Basic information about the book such as title, author, price, etc.
- Your purchase history for the book
- Availability
- Buying options
- Other items frequently bought with this book
- Other items bought by customers who bought this book
- Customer reviews
- Sellers ranking
- …

Since the online store uses the Microservice architecture pattern the product details data is spread over multiple services. For example,

- Product Info Service - basic information about the product such as title, author
- Pricing Service - product price
- Order service - purchase history for product
- Inventory service - product availability
- Review service - customer reviews …

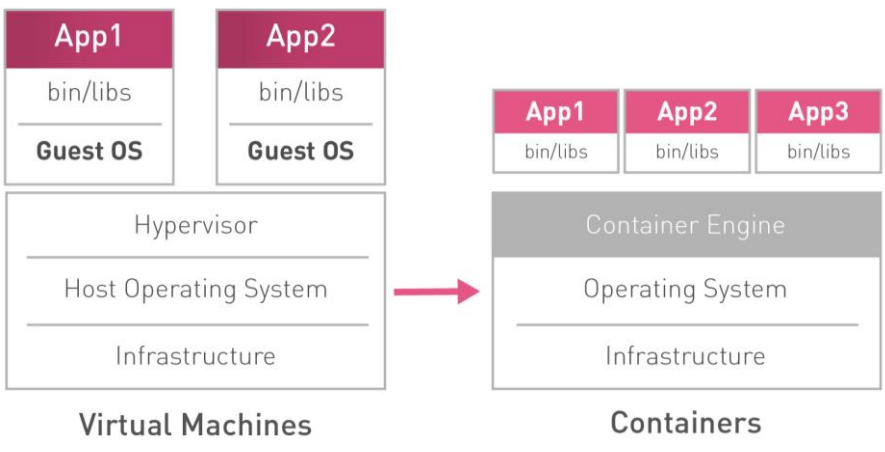
Consequently, the code that displays the product details needs to fetch information from all of these services.

The Microservices approach enables the development of every microservice by a small, dedicated team, using different programming languages, environments, and even deploying new versions into production independently.

Virtual Machines and Containers

Every microservice needs to run somewhere. As they were built in different environments, it is hard to make them work on a single server. Different services may require different libraries versions and can affect other services on the server. At the same time, providing a separate Virtual Machine per microservice is too expensive.

Container technology is a modern, lightweight approach for microservices isolation.



Both virtual machines and containers are used to create isolated virtual environments. However, they have significant differences critical for the cloud-native approach.

A container is a standard unit of software into which an application is packaged with all the dependencies necessary for its operation - the application code, launch environment, system tools, libraries, and settings. Containers use the host operating system's kernel and contain only apps and some lightweight operating system APIs and services that run in user mode. This differentiates them from Virtual Machines running a complete operating system including their own kernel.

Containers provide lightweight isolation from the host and other containers but don't provide as strong a security boundary as a VM. They run the user-mode portion of an operating system and can be tailored to contain just the necessary services for your app, using fewer system resources.

Pets Versus Cattle Meme

There is an interesting metaphor explaining the difference between virtual machines and container infrastructures.

In 2012 Randy Bias gave an impactful talk and established the pets versus cattle meme (Randy attributed the origins to Bill Baker):

- With the **pets approach to infrastructure**, you treat the machines as unique systems, which can never be down. Each pet server might have “pet” names like Poseidon or Vakhmurka. They are “unique, lovingly hand-raised, and cared for, and when they get sick, you nurse them back to health”.
  This approach is generally considered to be the dominant paradigm of a previous (non cloud-native) era, although it is still applicable for mainframes, databases, firewalls, etc.
- With the **cattle approach to infrastructure**, servers are given identification numbers like web01 and app-fin-14, the same way cattle are given identification numbers tagged to their ears. Each server is “almost identical to each other” and “when one gets sick, you replace it with another one”. Scalability is provided by creating more. If one fails it is just replaced with another one, as is done with a faulty hard disk drive.

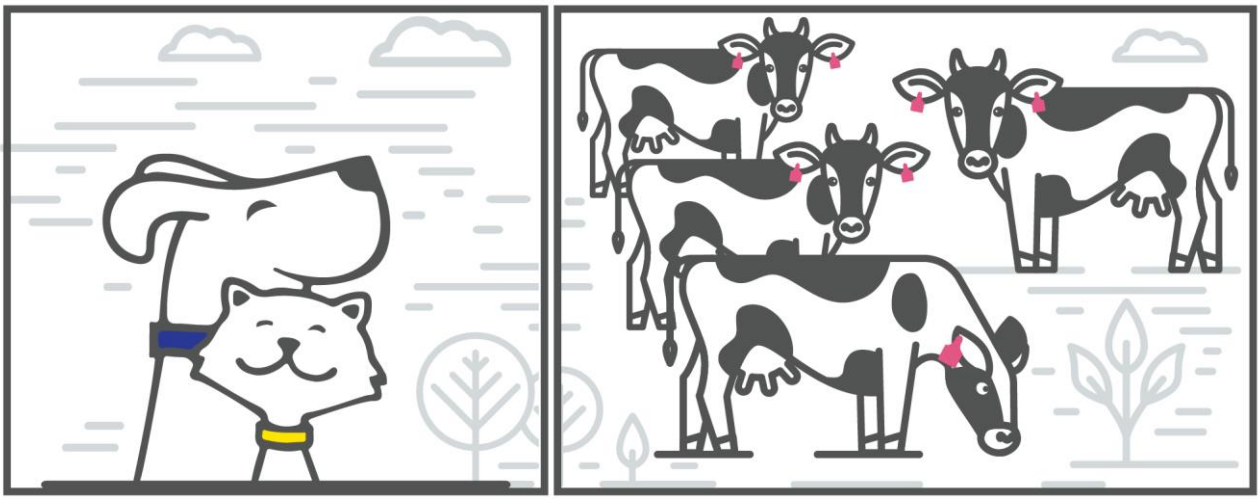
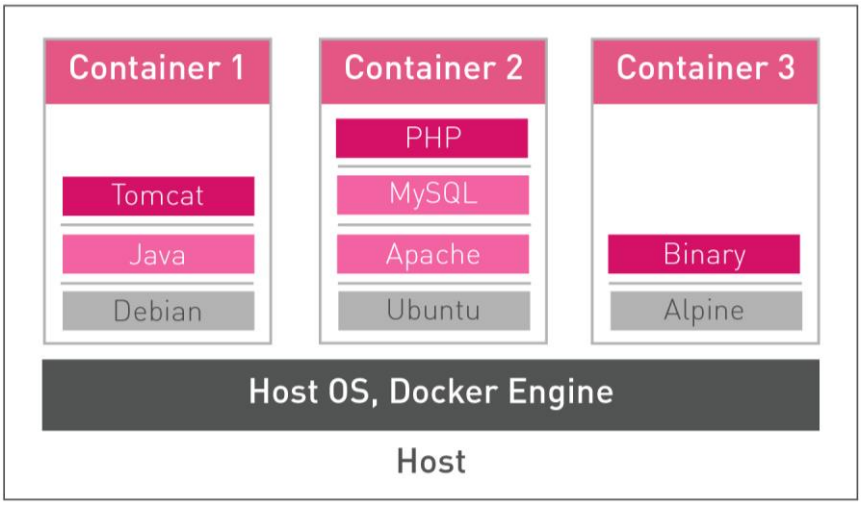


Figure 1 Pets vs Cattle

What is Docker?

Docker is an open-source technology used to deliver software in packages called containers. A typical Docker image consists of several layers. Each layer corresponds to certain instructions in the Dockerfile. In the example below, you can see the container image consisting of 4 layers: Operating system (Ubuntu), Apache, MySQL, and PHP with modules.

It is important for a “host” operating system to be another Linux distributive, which gives very good flexibility.

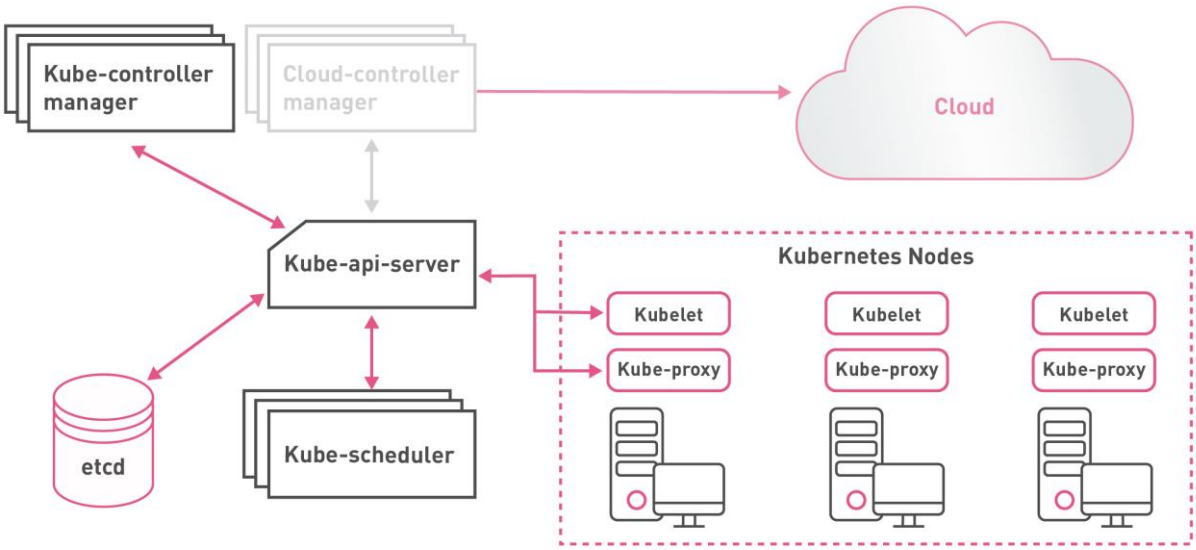


Such images can be stored in private or public registries like Docker Hub, Google Container Registry, Amazon Elastic Container Registry, etc.

What is Kubernetes[1]?

Besides running containers (Docker is one of the run-time platforms for this), it is necessary to automate application deployment, scaling, and management. That's why an orchestrator is also needed.

Kubernetes (commonly stylized as k8s) is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications. Kubernetes is often used to provide a platform or infrastructure as a service (PaaS or IaaS), and many vendors provide their own Kubernetes distributions.



The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

[1] Drawings and definitions in this chapter were taken from the https://kubernetes.io/

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. Multi-master configuration is recommended for reliability.

Kube apiserver

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

Etcd

Consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.

Kube scheduler

Control plane component that watches for newly created Pods with no assigned node and selects a node for them to run on.

Kube controller manager

Control Plane component that runs controller processes.

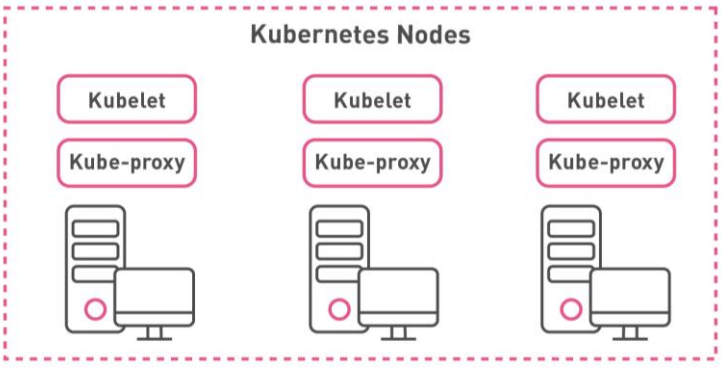
- **Node controller:** Responsible for noticing and responding when nodes go down.
- **Replication controller:** Responsible for maintaining the correct number of pods for every replication controller object in the system.
- **Endpoints controller:** Populates the Endpoints object (joins Services & Pods).
- **Service Account & Token controllers:** Create default accounts and API access tokens for new namespaces.

Cloud controller manager (optional)

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.

If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

Node



The **working nodes** in a Kubernetes cluster are the machines (VMs, physical servers, etc.) that run your applications and cloud workflows. The Kubernetes master controls each node, assigning individual Private IP subnets for every node.

Kubelet

An agent that runs on each node in the cluster, making sure containers are running in a Pod. The kubelet doesn't manage containers not created by Kubernetes.

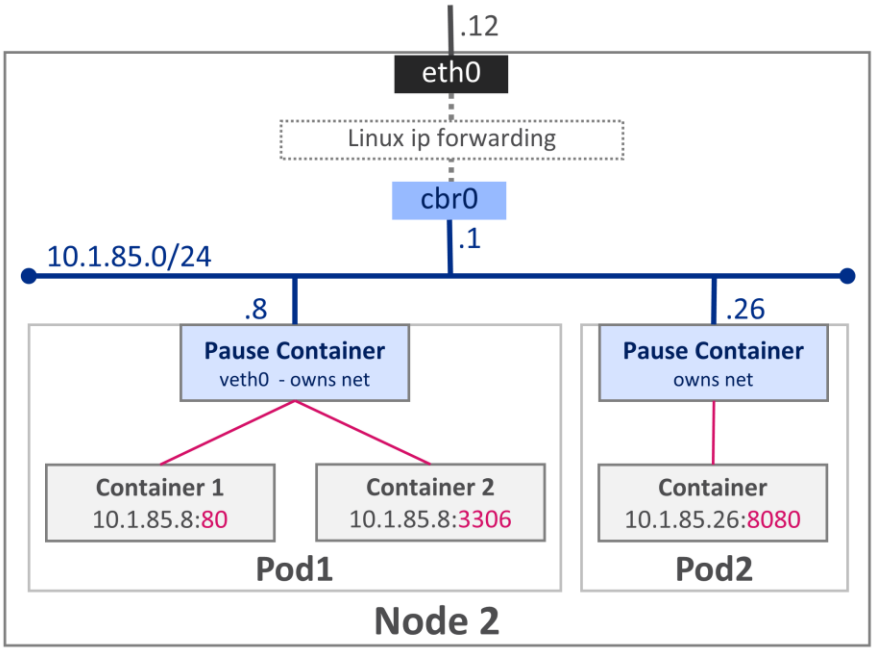
Kube proxy

Is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

The **container runtime** is the software responsible for running containers.

Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

Pod



A **Pod** (as in a pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers.

Containers within a Pod share an IP address and port space and can find each other via localhost. A "pause container" owns the Pod's IP address.

Pods serve as a unit of deployment, horizontal scaling, and replication.

Any container in a Pod can enable **privileged** mode, using the privileged flag on the security context of the container spec. This is useful for containers that want to use Linux capabilities like manipulating the network stack and accessing devices.

Processes within the container get almost the same privileges available to processes outside a container.

Sometimes the Pod might encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources (Pod1). However, the "one-container-per-Pod" model is the most common Kubernetes use case (Pod2).

Kubernetes Networking

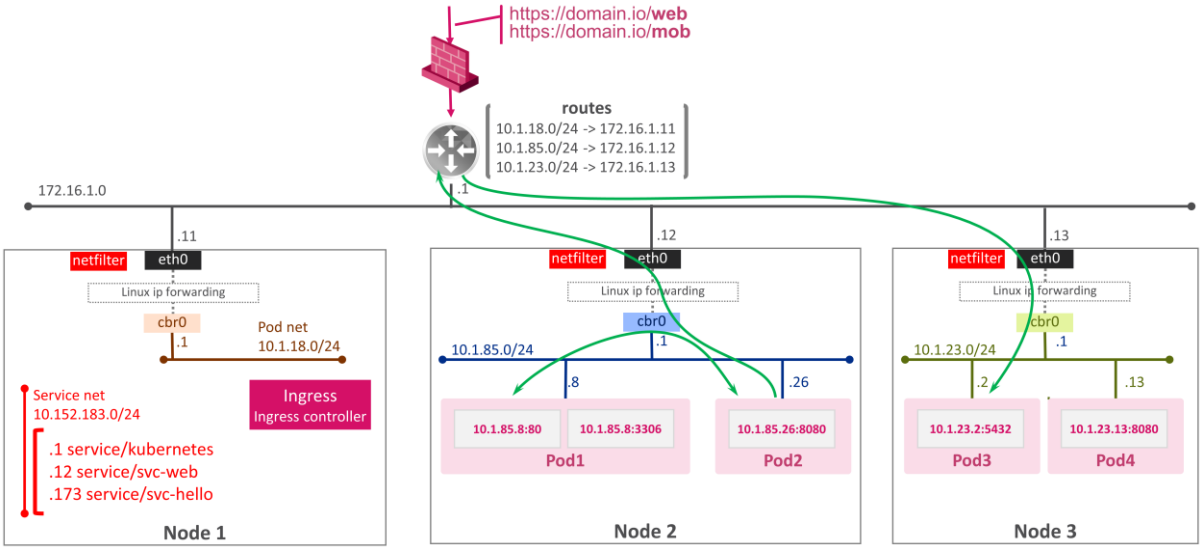


Figure 2 – Kubernetes Networking

The Kubernetes master controls each node and assigns individual Private IP subnets for every node. For example, 10.1.85.0/24 for Node2 and 10.1.23.024 for Node3 on the picture above.

Each **Pod** gets its own IP address. However, Pods are ephemeral. The set of Pods and their IP addresses running in one moment in time could be different from the set of Pods and their IP addresses running that application a moment later. Pod1 has an IP address 10.1.85 but most likely it will have another one after a restart.

There are different ways to establish communication between nodes. It is implemented by the use of different Container Network Interfaces (CNI).

Option 1 - unencapsulated

Pod2 (10.1.85.26) running on Node2 (172.16.1.12) tries to reach Pod3 (10.1.23.2) running on Node3 (172.16.1.13) by sending packets to its default router (172.16.1.1). The router forwards the packet 10.1.23.2 to the 172.16.1.13 (Node3) according to its routing table. And finally, the packet reaches its destination.

This is the Calico CNI plugin approach. Instead of static routes, it often uses BGP (Border Gateway Protocol).

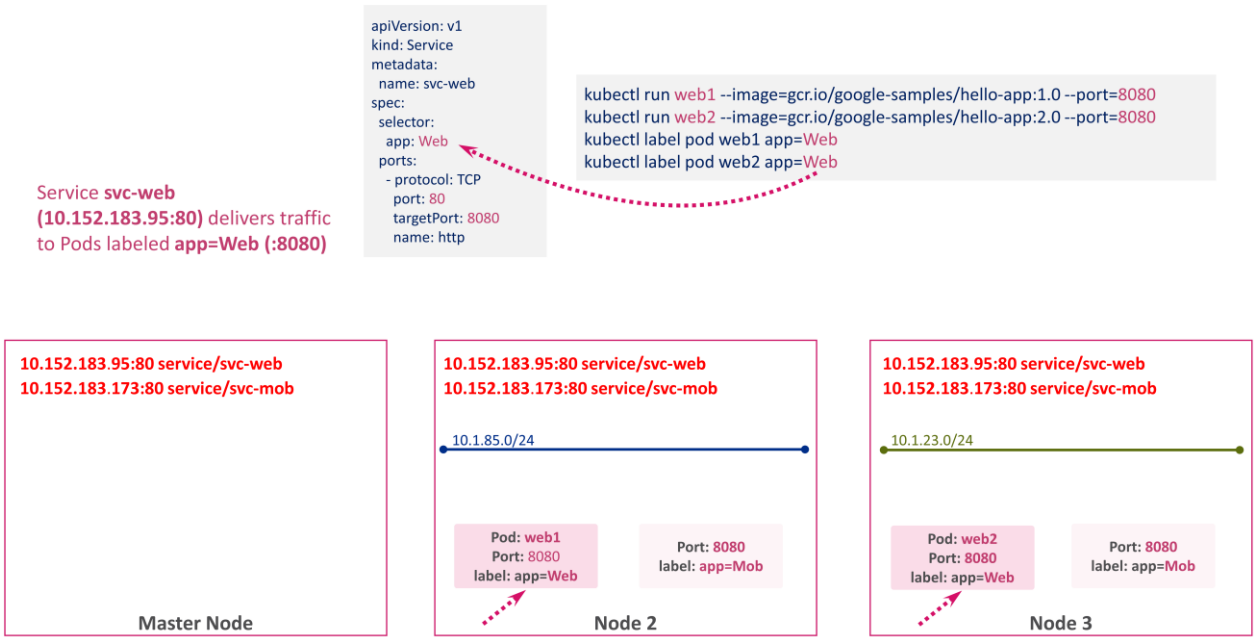
Option 2 – encapsulated

Another CNI plugin (Flannel) uses another approach by default. It configures a layer 3 IPv4 overlay network and creates a large internal network that spans across every node within the cluster. Traffic is routed to pods on different hosts being encapsulated in UDP packets by flanneld.

By default, it uses VXLAN for the encapsulation. Also, it can work in other modes (BGP) and support plugins extending its functionality (for example, it can use Strongswan to encapsulate and encrypt the packets).

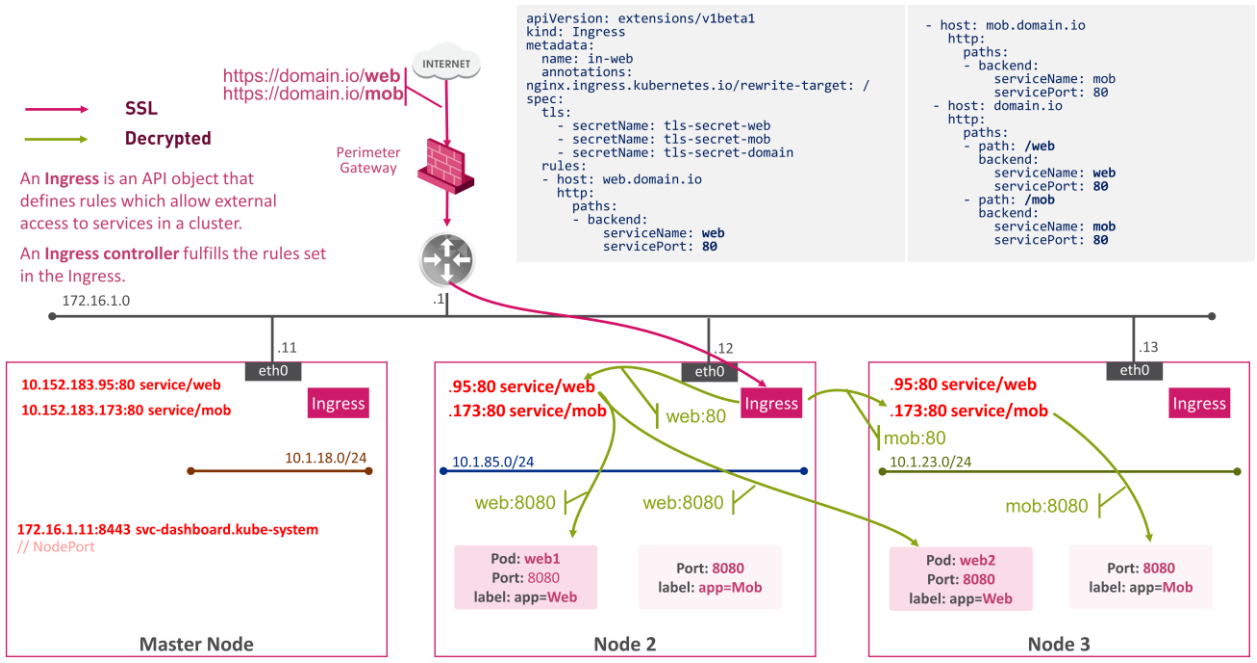
There are many other options, of course.

A **Service** is an abstraction, which defines a logical set of Pods and has a fixed IP address. The set of Pods targeted by a Service is usually determined by a selector (labels). A service svc-web with a fixed IP address 10.152.183.95:80 will distribute traffic to pods with labels app=Web (targetPort=8080).



Service works as netfilter on each node and is effective for communication between pods. In order to make it accessible outside, it needs to be exposed using NodePort or other mechanisms.

Ingress is an API object that manages external access to the services in a cluster. It exposes HTTP and HTTPS services and makes them available from outside the cluster. It may provide load balancing, SSL termination, and name-based virtual hosting.



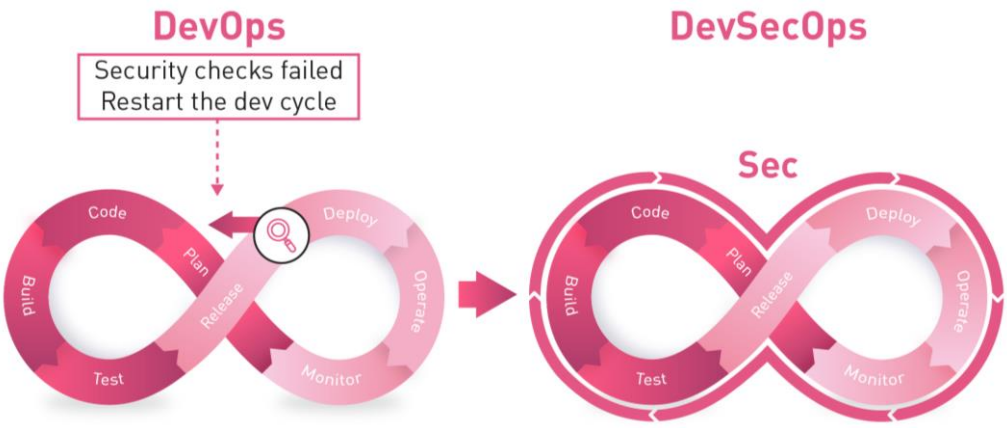
We will explore this in more depth later, together with the security aspects.

SECURITY RISKS AND CHALLENGES

DevSecOps

While the DevOps approach implements security checks at the final stages of the software development life cycle, DevSecOps has a tendency to automate all security checks and use them from the very beginning of software development on each and every stage of the CI/CD pipeline.

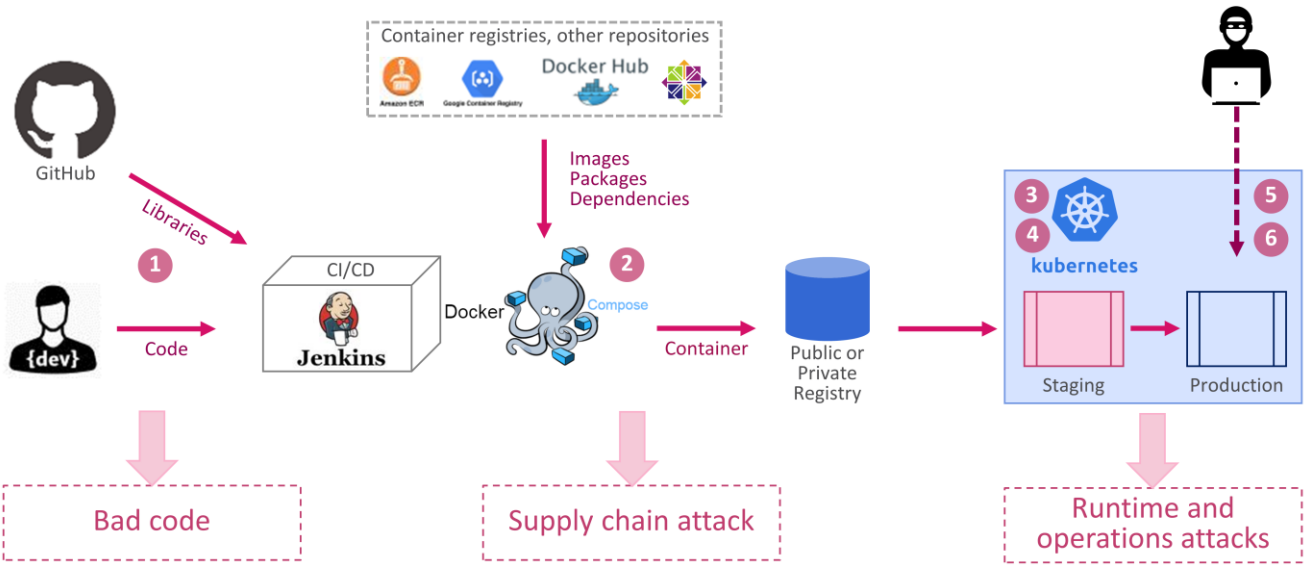
This is especially important because, according to the modern approach, new features can run into production several times a day.



Development workflow security risk (CI/CD)

A typical development workflow includes the following steps: Develop, test locally, commit code to the version control system; CI/CD system takes this code, builds it, pushes it to the Docker compose which builds a container also using images and packages from public repositories and places it into the public or private registry; after successful staging the container goes to production.

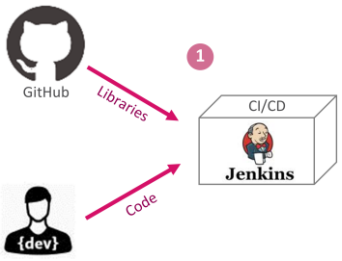
These steps bring additional security risks.



1. Code with vulnerabilities

Company software developers can write a code containing vulnerabilities, credentials, or contain other potential security issues.

They can use external libraries (for example, downloaded from the GitHub) which bring additional risks (poor code quality or even specially crafted backdoors).



Example:

Hardcoded credentials are a very common way to obtain unauthorized access.

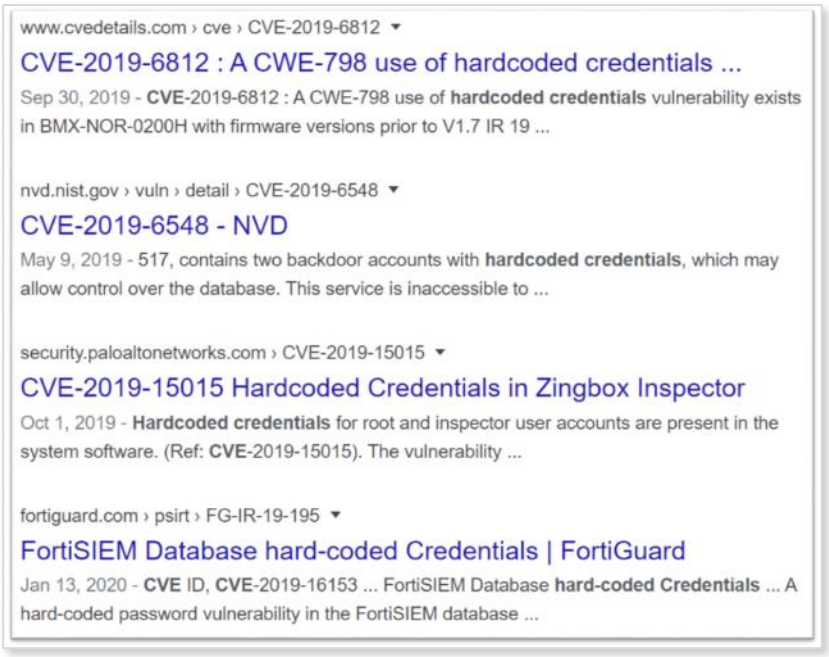
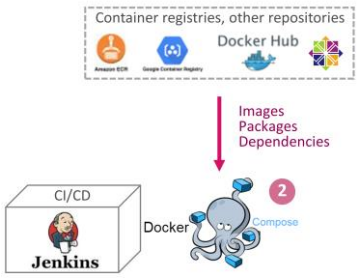


Figure 3 Hardcoded credentials

2. Supply chain attacks

Containers are usually built on top of images from public registries and added packages from public repositories. They can be compromised intentionally, replaced by hackers after hacking the registry. The use of repositories (RPMs and other dependencies) maybe not be safe as well.



Example:

In 2019 the Docker Hub repository was breached, 190K accounts were compromised. Images may have been tampered.

Docker Hub Breach: What to Do Now

May 1, 2019 best practices, container, container images, docker hub, kubernetes, tutorial

by Wei Lien Dang

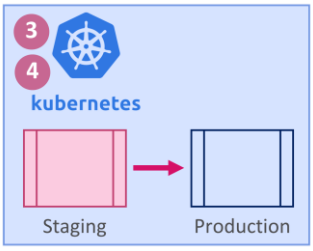
Last week, it was disclosed that a breach had occurred in the Docker Hub repository for container images. In an email to customers, Kent Lamb, director of Docker Support, wrote: "During a brief period of unauthorized access to a Docker Hub database, sensitive data from approximately 190,000 accounts may have been exposed (less than 5% of Hub users). Data includes usernames and hashed passwords for a small percentage of these users, as well as Github and Bitbucket tokens for Docker autobuilds."

As a result of this breach, it's possible that images in your Docker Hub repository may have been tampered with or overwritten. To mitigate this possibility, you should take immediate action. You can follow the recommendations below for any deployment.

Figure 4: Docker Hub breach

3. Insecure posture

Improper environment configuration (file permissions, access rights, etc.) could have a serious negative impact. There are many benchmarks (CIS Kubernetes Benchmark to name one) providing lists of a hundred items to check, which is hard to do manually on a regular basis.



Example:

CIS Kubernetes Benchmark provides prescriptive guidance for establishing a secure configuration posture for Kubernetes to avoid attacks like "Send malicious YAML and JSON payload causing API server to consume excessive CPU or memory or even crashing and becoming unavailable".

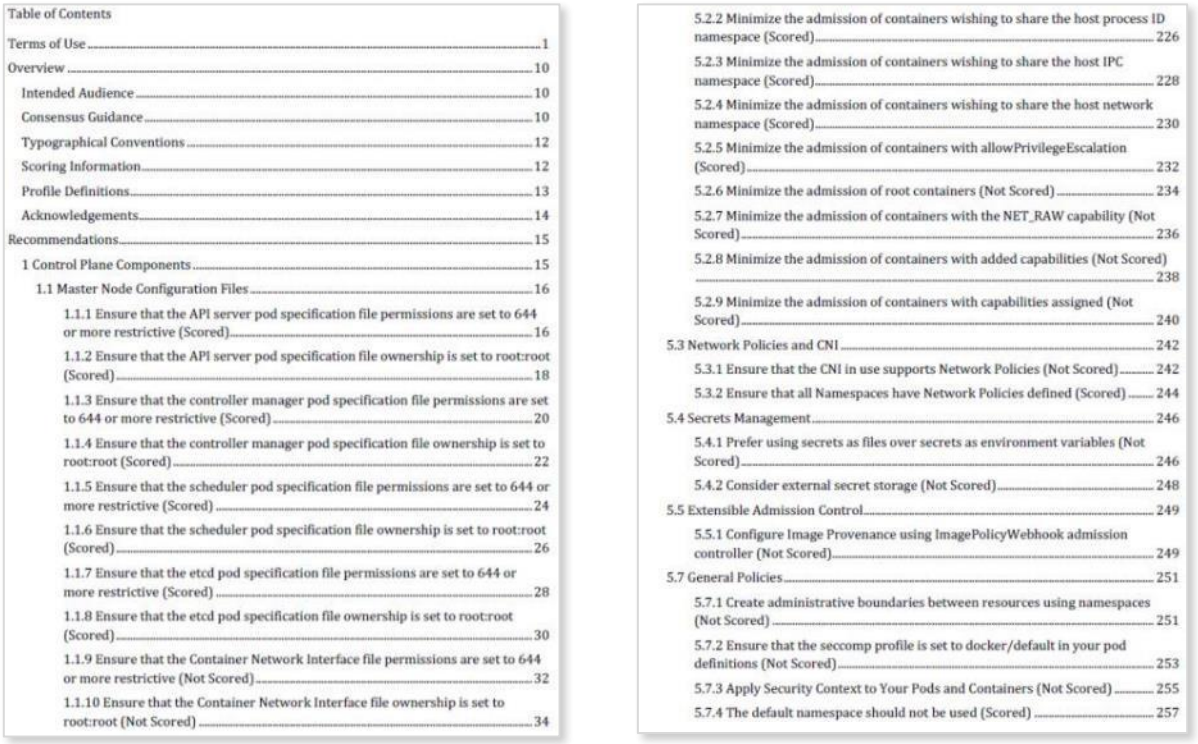


Table of Contents

Figure 5 CIS Kubernetes Benchmark

4. Infrastructure vulnerabilities

The infrastructure itself could have vulnerabilities (multiple CVEs for Docker, Kubernetes, and plugins like “execute code”, “bypass something”, and “gain privilege”), which must be addressed.

Example:

Containers accidentally ran as root gaining permissions, as host processes have.

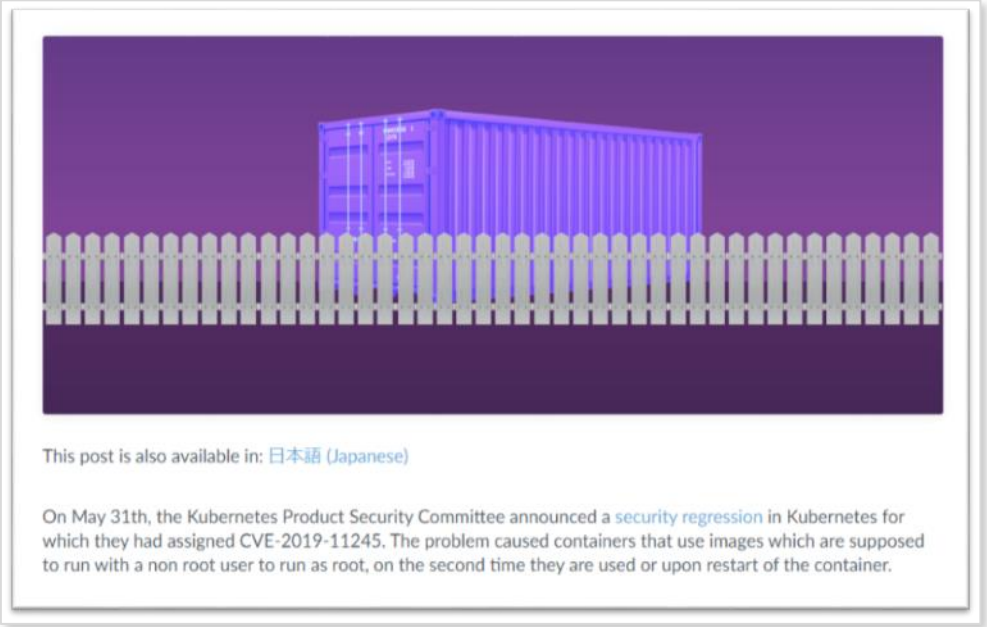


Figure 6 Non-root containers run as root

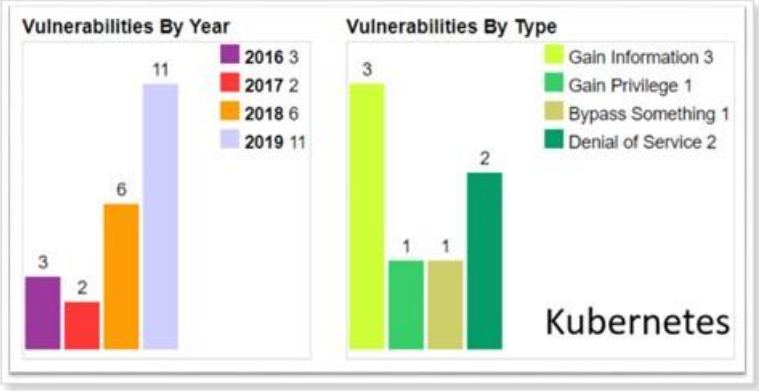
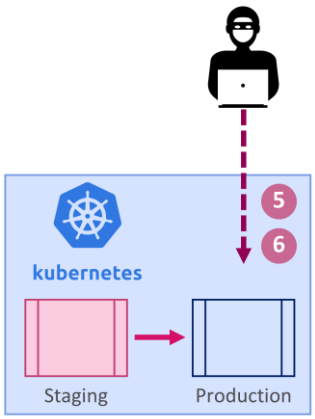


Figure 7 Kubernetes CVEs

5. Kubernetes networking challenges



Workers nodes and Pods have flat network architecture. This means each and every container (pod) can talk to any other one. This brings the legacy security issue of massive infection of the entire environment in a case where one of the containers has been compromised and infected by malware (known as lateral movement).

Communication between Pods can hardly be controlled. Moreover, containers are ephemeral. They usually start on new IP addresses, which makes traditional IP-based firewall policies useless.

Egress traffic from containers is usually NATed behind the node IP address, which makes it impossible to enforce granular policies.

6. Ingress SSL inspection

If SSL decryption is implemented on the Ingress, attacks in the SSL bypass traditional security gateways. There is also a new attack surface: API gateways widely used in the microservices architecture and based on the REST API (also encrypted).

Example:

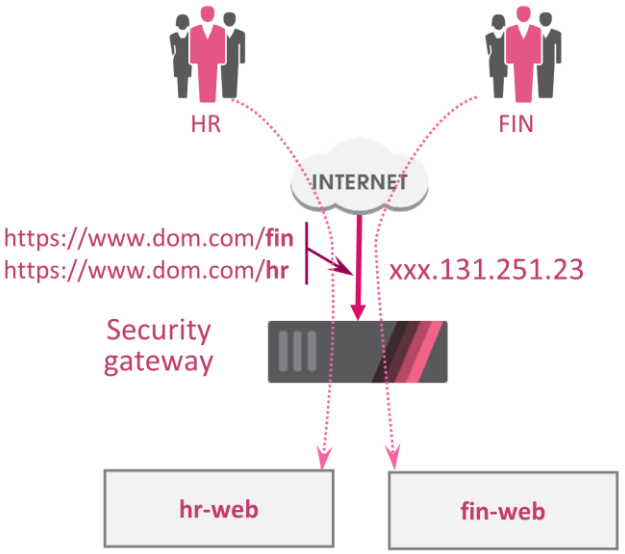


Figure 8 SSL inspection

It is difficult to provide access control (providing access to fin-web only to FIN group but not HR) because both resources (https://dom.com/fin and https://dom.com/hr) have the same domain name and IP address, while the path is encrypted and not visible to the perimeter gateway.

IPS and other security checks are also inapplicable.

Shared Responsibility

At some point, enterprises are likely to use all the available cloud services, including the public IaaS, PaaS, FaaS and SaaS. The issue is all these platforms have different operational benefits, shared responsibilities, and security challenges.

In traditional IT environments, the enterprise owns the whole stack and the dedicated security team makes the necessary infrastructure changes. In the cloud, some responsibilities are transferred to cloud service providers,

and some are transferred to application owners. Shared responsibility models challenge the traditional models of security implementation, management, and administration.

Leading research and advisory company, Gartner, stated that "*through 2020, 95% of cloud security failures will be the customer's fault".* Our own analysis also concludes that customer misconfiguration is the most common reason behind cloud security breaches. We believe this is partly due to customers thinking the cloud provider has secured, monitored, and appropriately configured the environment.

Enterprises must be aware that when they implement cloud-native security controls and integrate with solutions such as FaaS, PaaS, and SaaS, they need to take responsibility for the new cloud security policies such as access control, data protection, application activity visibility, content-awareness, and threat prevention.

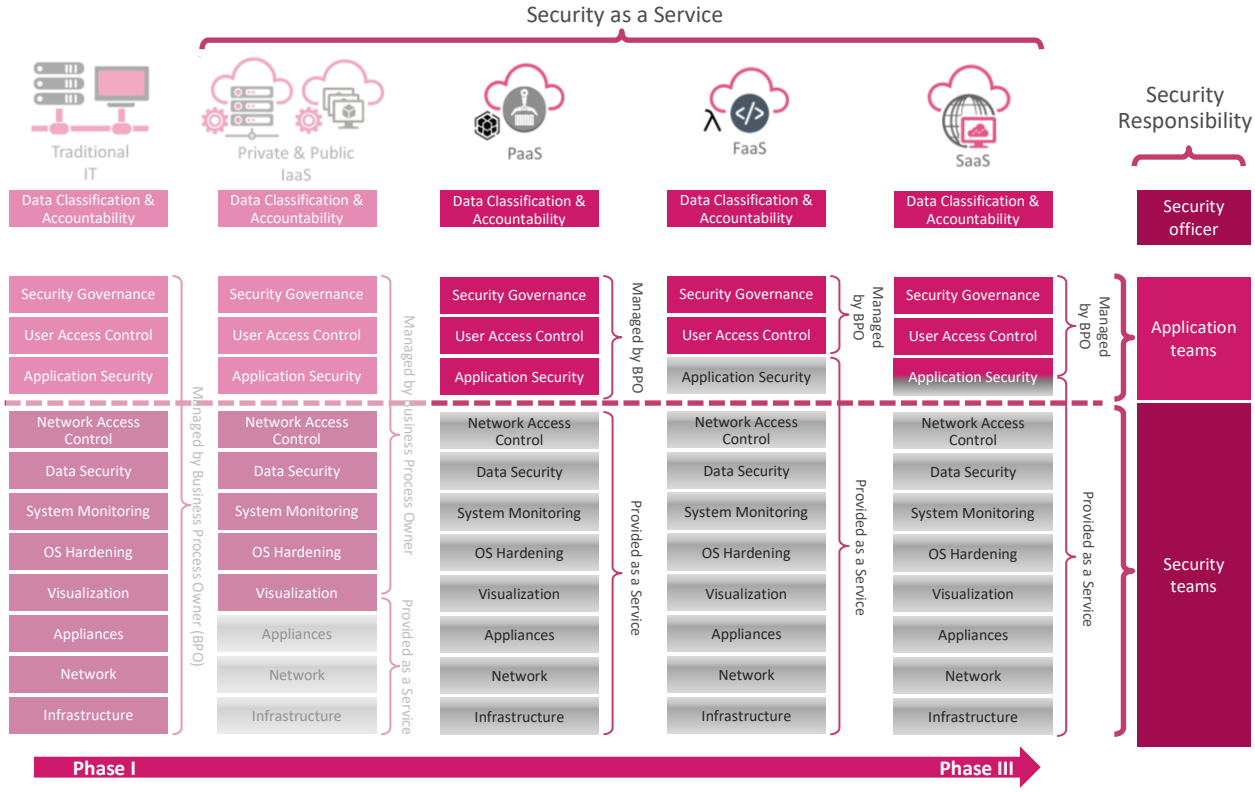


Figure 9 Shared responsibility for the user and business process owner between the cloud computing models

We believe as organizations shift towards SaaS and FaaS services, security responsibility will change and traditional technology teams will have a different role to play in enforcing the organizational security policy. There are a number of different ways we see this happening; either through the assimilation of existing DevOps teams to form DevSecOps teams, or by moving into security assurance and responsibility roles.

This shift in security responsibility means that although the Security team still has overall responsibility for the security posture, the implementation of security is done by Application and DevOps teams. Therefore, it is important for the cloud transformation process to capture who is actually going to own the implementation of security and who is responsible for its management.

For example, an existing DevOps team is responsible for securing their application and moving the app to a PaaS platform. They have to abide by the organizational security policy of installing the Check Point WAAP agent into the ingress node. The responsibility for this install is with the DevOps team. Once the install is complete, the Security team will monitor the traffic.

The graphic below shows an example of the various cloud technology services and how security responsibility is shared between the Security team, Network team, and Application teams as organizations move towards full cloud-native technology and development practices.

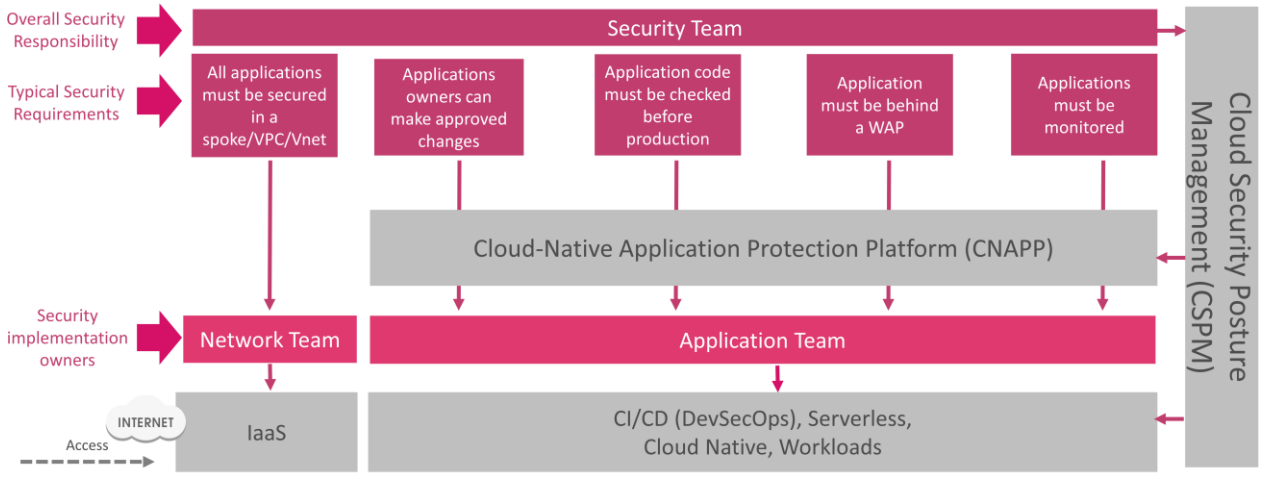


Figure 10 Security oversight and shared security responsibility in cloud-native environments

CLOUD-NATIVE APPLICATION PROTECTION PLATFORM

Gartner's definitions

The Check Point CloudGuard portfolio aims to deliver the industry's most advanced threat protection to keep cloud networks, data, and applications protected from sophisticated Gen V cyber-attacks. The comprehensive portfolio is seamlessly integrated with the largest number of cloud platforms and cloud-based applications to instantly and easily protect any cloud service.

Besides well-known Network Security, the Cloud-Native Security concept brings new terminology, such as Cloud Security Posture Management, Workload Protection, API protection, Cloud Intelligence and Threat Hunting.

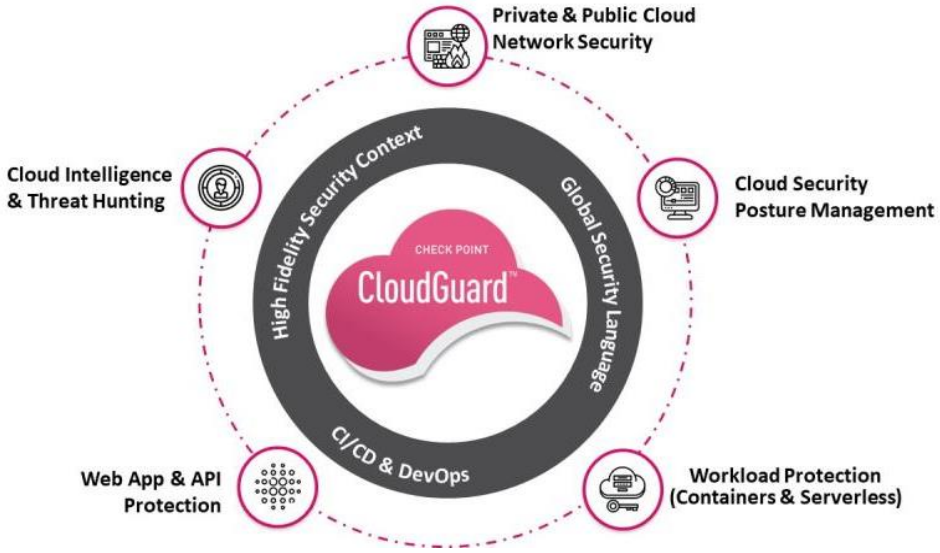


Figure 11 CloudGuard: Any Cloud, Any Asset, Any Application, Unmatched Security

Check Point unifies all these terms and technologies and provides a holistic solution to secure Cloud-Native environments using comprehensive techniques addressing all aspects of cloud security such as containers, Kubernetes, and CI/CD pipeline.

- **CPSM –** Cloud Security Posture Management

  Security, governance, and compliance automation for public clouds and Kubernetes environments. With continuous visibility and control that helps organizations minimize their attack surfaces and protect against vulnerabilities, identity theft, and data loss in the cloud.

- **CNSS** – Cloud-Native Security Services

  Mostly relates to Public and Private IaaS security and leverages virtual Firewall with extended protections, WAF/WAAP to perform macro and micro-segmentation.

- **CWPP** – Cloud Workload Protection Platform

  Automated security for any workload, in particular AWS Lambda functions and Kubernetes. Provides seamless vulnerability assessment and creates deep code flow analysis and behavioral profiles in runtime for unmatched protection - with minimal performance impact.

The concept of a single Cloud-Native Security Platform or, as Gartner describes, a Cloud-Native Application Protection Platform (CNAPP), is the combination of Cloud Workload Protection (CWPP) and Cloud Posture Management (CSPM) into a single platform.

This approach aligns with Check Point's vision of a single overarching security platform in and on which, all Cloud-Native Security functions are built. The infographic below shows this approach whereby Check Point Cloud Guard Dome9 becomes the CNAPP.

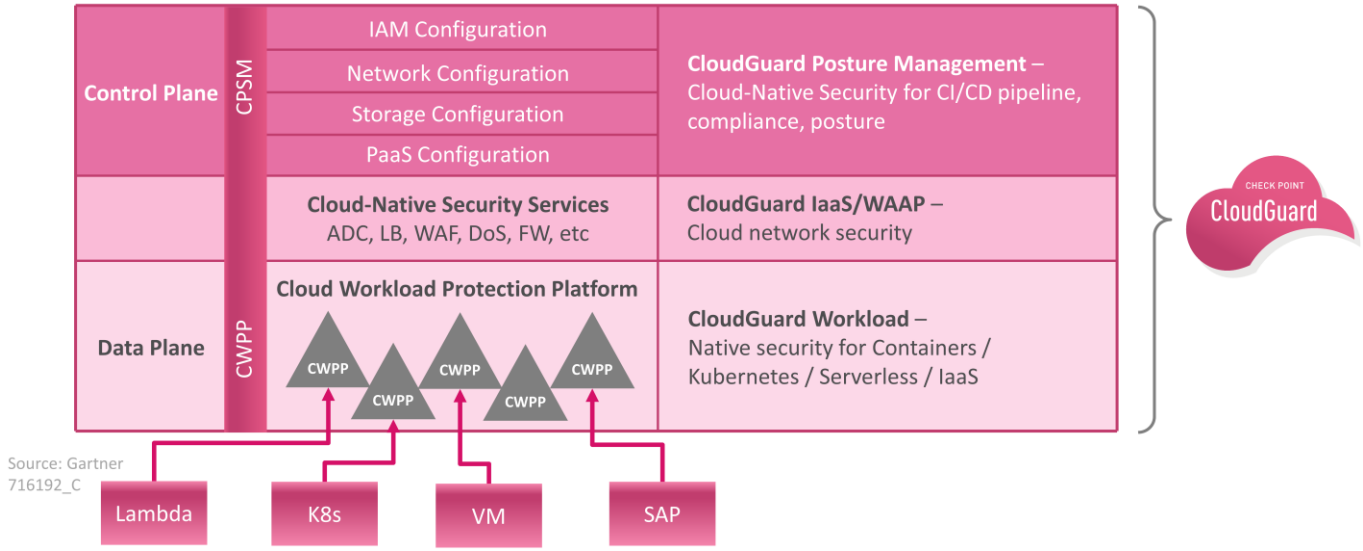


Figure 12 Check Point Infinity CNAPP

The 4Cs Approach

Security, in general, is an immerse topic, which has many interconnected parts. For Cloud-Native Security a 4Cs approach can be applied. It is a layered approach based on **4Cs**: **Cloud, Cluster, Container, and Code**.

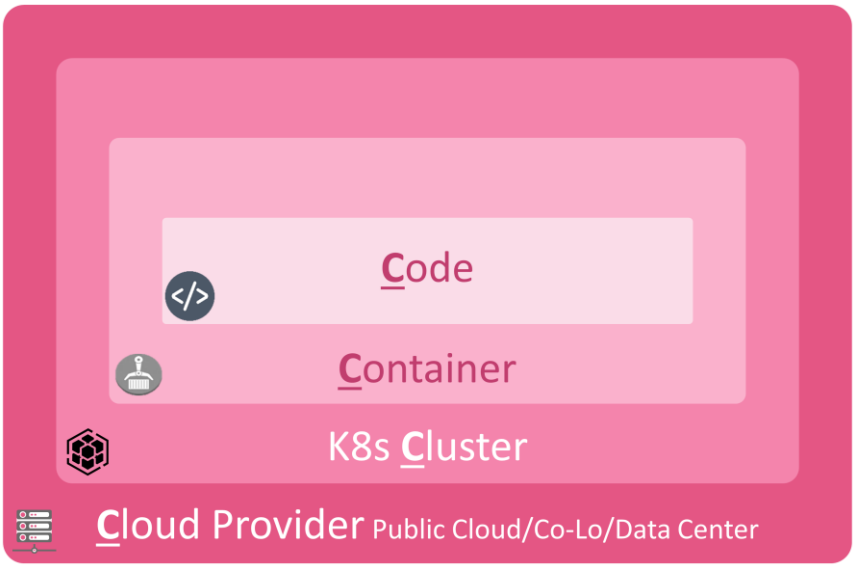


Figure 13 Source: https://kubernetes.io/docs/concepts/security/overview/

This approach summarizes the typical pains in Cloud-Native Security environments and makes obvious the need for layered security solutions, which cover multiple layers in a unified manner, complementing each other and providing seamless protection.

Cloud

In many ways, the Cloud (co-located servers, or the corporate datacenter) is the trusted computing base of a Kubernetes cluster. If these components themselves are vulnerable (or configured in a vulnerable way) then there's no real way to guarantee the security of any components built on top of this base. Each cloud provider has extensive security recommendations they make to their customers on how to run workloads securely in their environment.

Cluster

There are two main areas of concern for securing Kubernetes:

- Securing the non-configurable components which make up the cluster.
- Securing the components which run in the cluster.

Container

In order to run the software in Kubernetes, it must be in a container. Because of this, there are certain security considerations that must be taken into account in order to benefit from the workload security primitives of Kubernetes.

Code

Moving down into the application code level, this is one of the primary attack surfaces over which you have the most control.

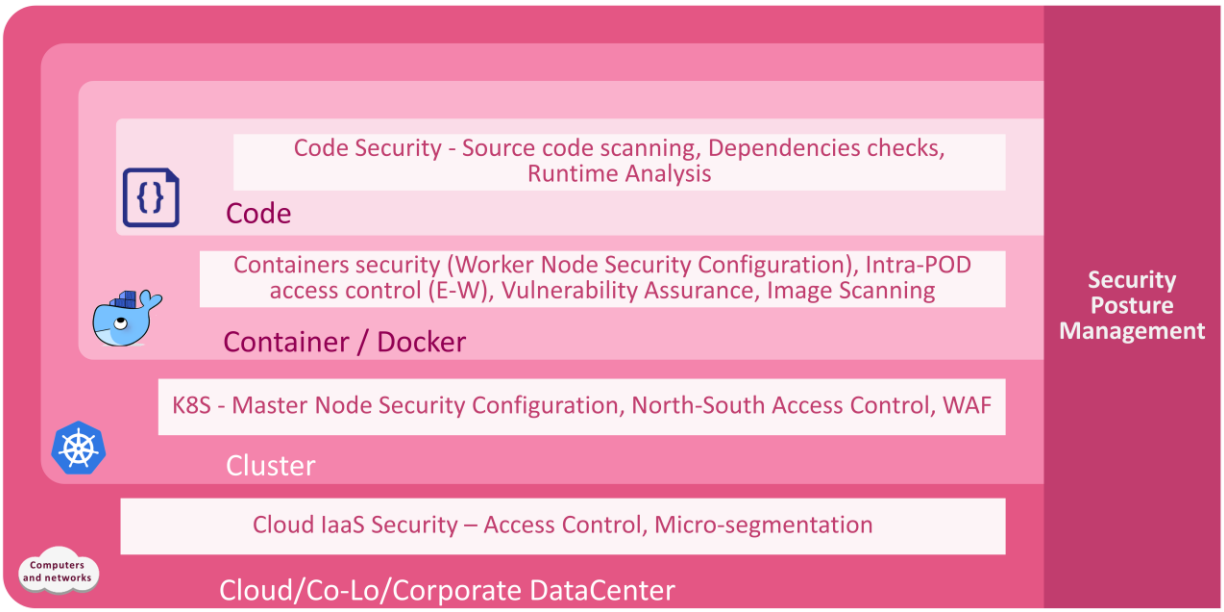


Figure 14 4C approach

K8s/Containers Security Architecture

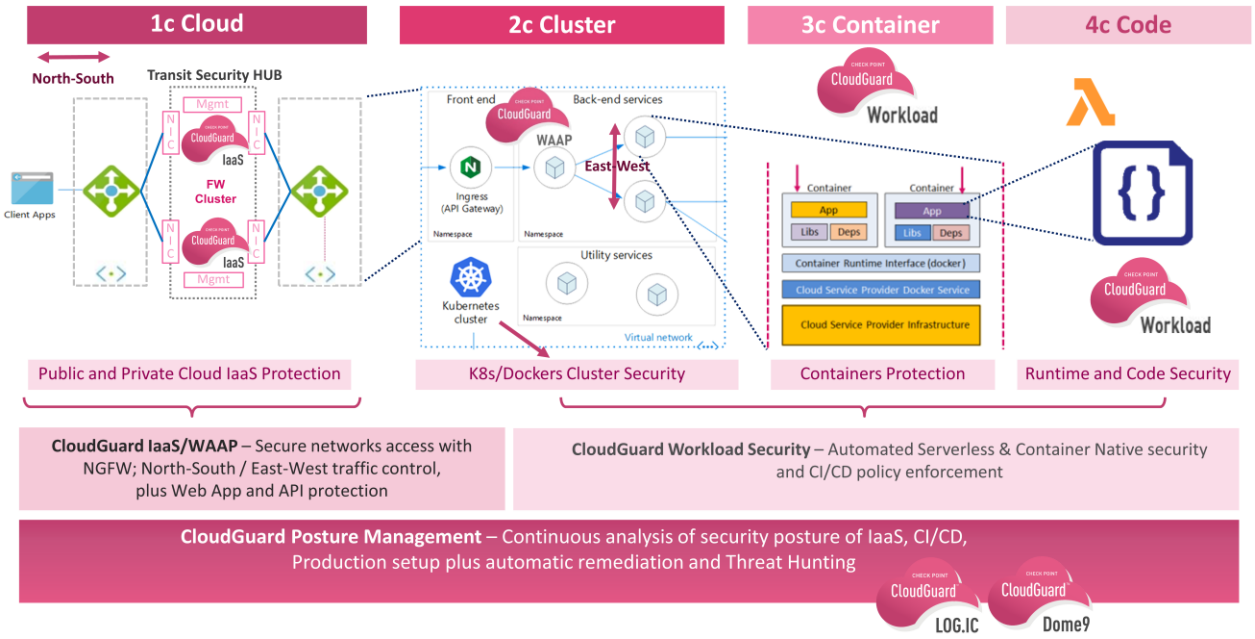


Figure 15 Cloud-Native Security Concept (4C)

Cloud-Native Security is mostly covered by the following product family:

CloudGuard IaaS: Cloud Network Security & Threat Prevention

- Cloud network security gateway including Firewall, IPS, Application Control, IPsec VPN, Antivirus and Anti-Bot.
- DevSecOps automation auto-provisioning and auto-scaling along with automatic policy updates.
- Central management unifies threat visibility and enforcement across cloud and on-premise infrastructures.

DOME9: Cloud Security Posture Management

- Posture management: Continuous analysis of security posture from CI/CD to Production plus automatic remediation.
- High fidelity security: The broadest and most flexible CSPM platform, with over 2000 out of the box rules.
- Unified: Visibility and control across all cloud assets leveraging a single global security language.

Workload Protection

- Vulnerability & posture management: Detect over-permissive permissions, vulnerabilities and embedded threats.
- Runtime protection: Multi-layer security, leveraging machine learning to automatically profile and protect workloads.
- Governance & DevSecOps: Enforce granular security policies during CI/CD and production.

WAAP: Web App & API Protection

- Complete application security: From OWASP top 10 attacks to zero-day API attacks, and malicious bot traffic.
- Automated: Self adapts to application changes, eliminating the need for manual updates and firewall rule tuning.
- Flexible deployment: Provisioned as a reverse proxy, proxy servers add on, or as an ingress controller on K8s.

LOG.IC: Cloud Intelligence & Threat Hunting

- Threat hunting: Detect cloud activity anomalies in a real-time, leveraging machine learning and through dedicated threat research.
- High Fidelity Context: Combining cloud configurations, VPC Flow Logs, cloud Logs, vulnerability data & Check Point's Threat Cloud.
- Actionable Intelligence: Intuitive visualization, querying, intrusion alerts, and notifications.

CHECK POINT CNAPP SOLUTION OVERVIEW

Following the 4Cs approach, Check Point offers solutions for each part (and even more).

These capabilities, shown in the diagram below, are essential building blocks of Check Point Unified Cloud-Native Security.

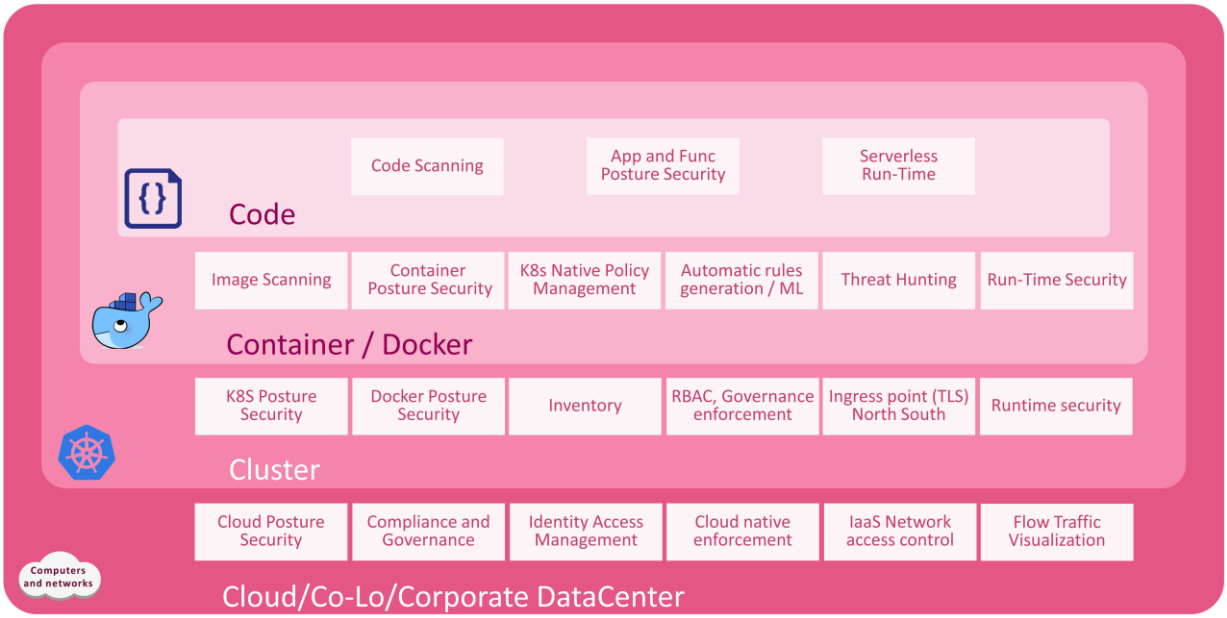


Figure 16 Check Point response to 4C challenges

On the below diagram you can find a reference architecture.

icons represent small (nano) agents, installed on the specific node or as a daemon-set.

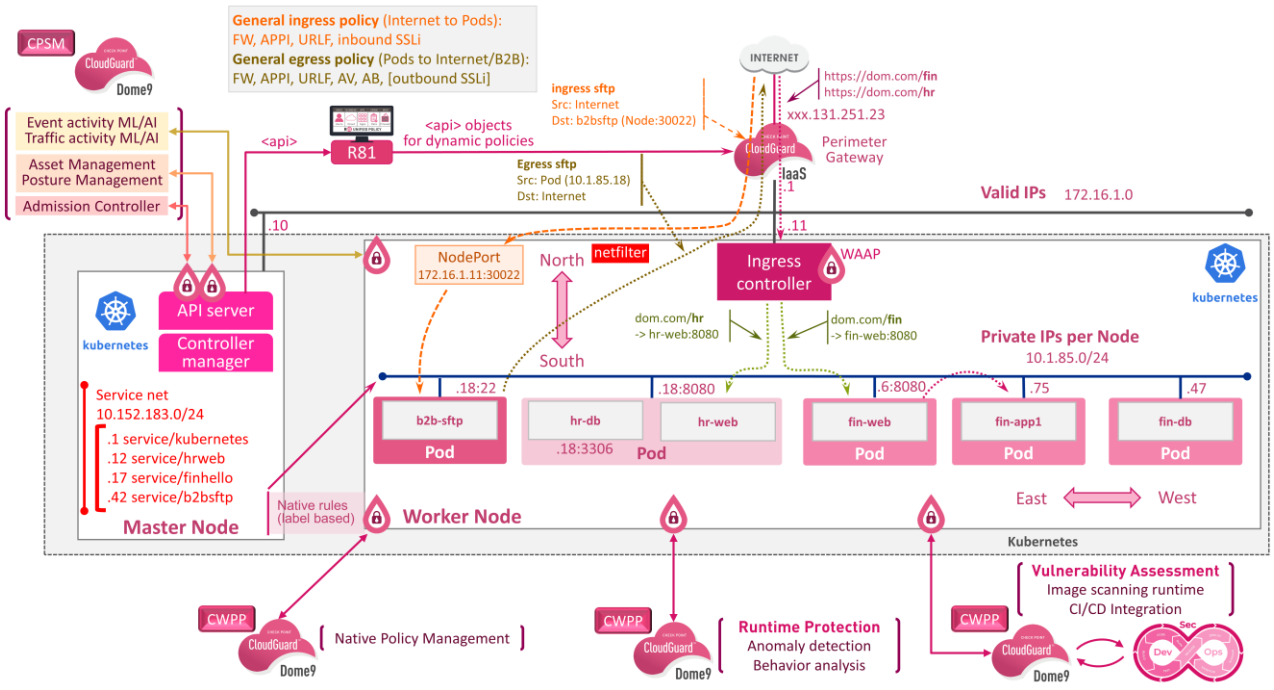


Figure 17 Reference architecture

Each section will be explained below.

Network Security in K8s Environment

Network security in K8s environments has the same definition as in regular networks to control North-South and East-West traffic flows. However, in the K8s platforms, these capabilities are usually achieved by leveraging different technologies with the dependencies of cloud platforms, e.g. Public Cloud AWS/Azure/GCP or Private Cloud VMware NSX-T.

North-South / East-West access control for Public Cloud (IaaS)

The core principle of network access control in the K8s environment is based on tight integration and communication with the Kubernetes management master node system in order to obtain relevant information about PODs and containers, as well as enforcing intra-POD security access policy.

For the North-South traffic inspection, the R81 SmartCenter API is being used to connect to Kubernetes management, while for East-West traffic control CloudGuard Dome9 leverages its native ability to communicate with Kubernetes and enforce security through the NetworkPolicies internal functionality of the Kubernetes cluster.

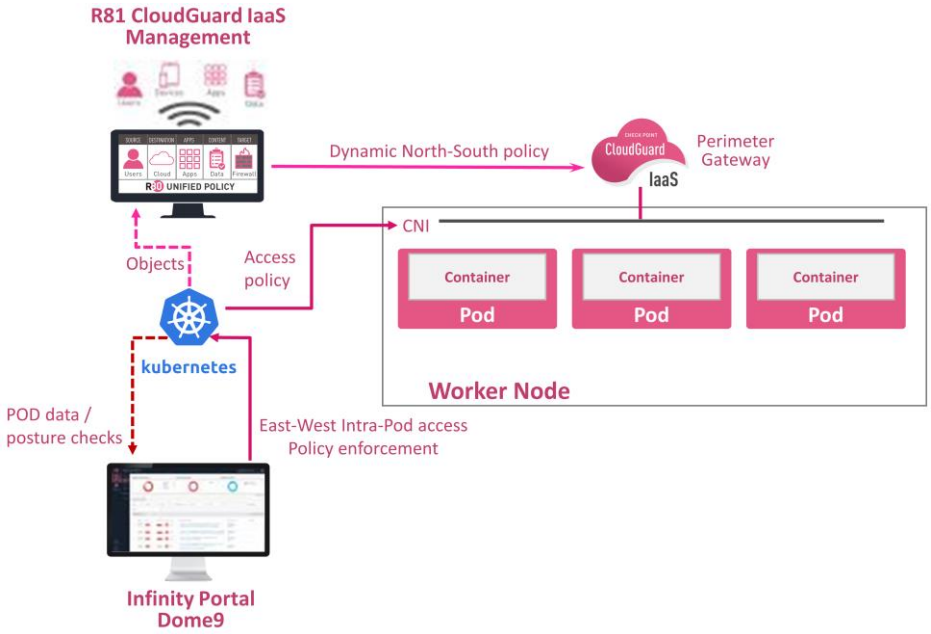


Figure 18 CloudGuard IaaS N-S and E-W protection

North-South

North-South security can be achieved by deploying CloudGuard IaaS, which provides cloud network security and advanced threat prevention for public, private, and hybrid cloud environments. Security features include Firewall, IPS, DLP, Application Control, IPsec VPN, Antivirus, and Anti-Bot. Customers purchasing SandBlast benefit from Threat Extraction and Threat Emulation.

With R81, CloudGuard IaaS provides unified and consistent security management of all public, private, and hybrid environments. For customers with on-premise environments and who are in the process of migrating to the cloud, CloudGuard IaaS enables the easiest, quickest, and most secure cloud migration with the lowest total cost of ownership.

In R81, CloudGuard Controller connects using API to the Kubernetes management master node and fetches POD/Containers object names and IP addresses. Furthermore, this information is used to create and push security access policies to the CloudGuard IaaS firewalls, to protect ingress and egress traffic from the worker nodes[2].

[2] Consider limitations of an Island mode (pods are NATed behind the Node IP address)

Figure 19 Kubernetes objects for dynamic policies

East-West

When East-West traffic protection in the public clouds and container environments is needed, typically native controls are used. However, if a complete traffic inspection is required, traffic from inside the node (between pods) should be redirected to a security gateway. Such an approach highly depends on the environment (used platform).

Network policy in a Kubernetes is a specification of how groups of Pods are allowed to communicate with each other and with other network endpoints. NetworkPolicy resources use labels to select Pods and define rules which specify what traffic is allowed to the selected Pods.

It provides required network access controls. However, this configuration is done by the YAML files, which means:

- It is hard to manage
- Prone to errors
- Lacks easy visibility of the whole picture.

It is much more effective to use a special security console providing a user-friendly interface to manage rules in a familiar way and translate this policy into the native one.

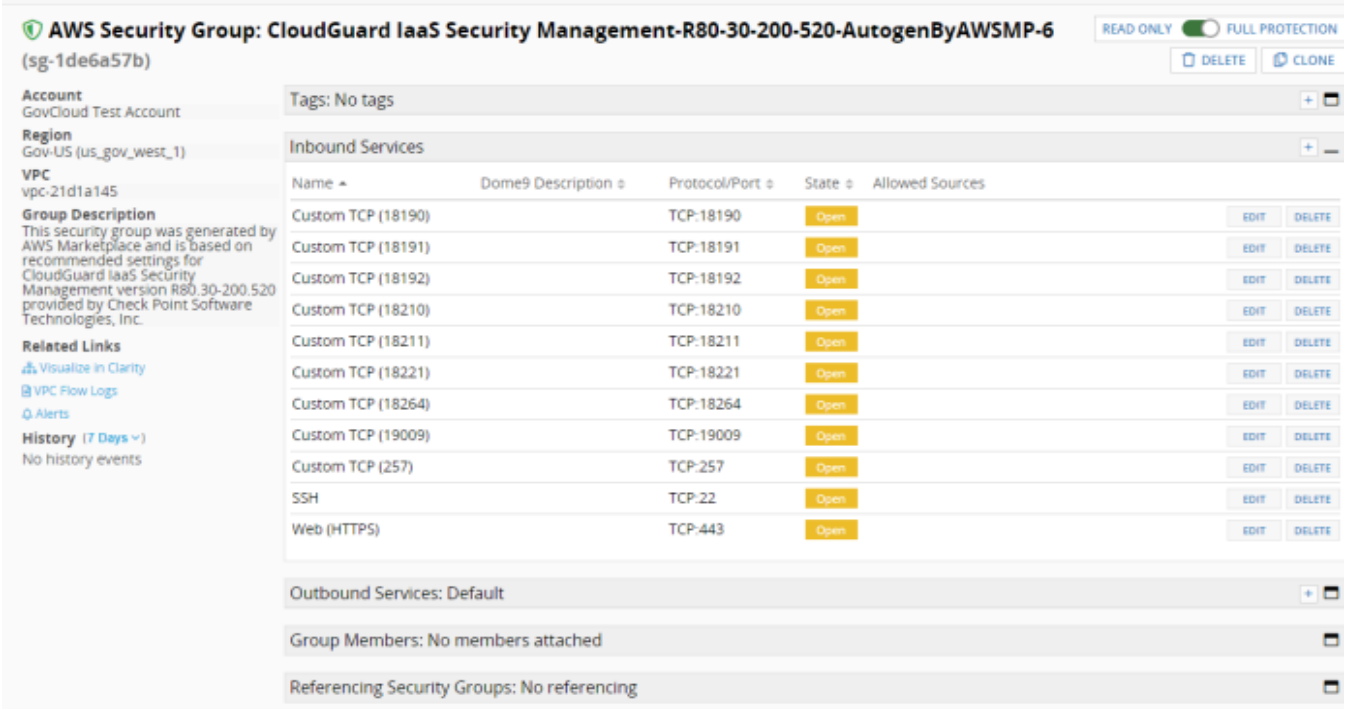


Figure 20 Security rules

The diagram below shows the recommended architecture reference for general secure network access control implementation in the K8s environments:

VMware NSX-T North/South and East/West Security

Kubernetes is a popular open-source container orchestration platform. Many cloud providers offer Kubernetes as a managed service, providing advanced features to businesses (Amazon Fargate on EKS/ECS, Azure Kubernetes Services, Google Kubernetes Engine, and more)

VMware also brings extended features with the integration between NSX-T Data Center and Kubernetes, as well as between NSX-T Data Center and Pivotal Cloud Foundry (PCF).

NSX-T provides a Service Insertion mechanism, which allows applying third-party services to North-South traffic as well as East-West traffic that passes through a router. Configuring the T0 router to redirect traffic to the CloudGuard IaaS enables implementation of thorough traffic inspection including IPS and Application Control with centralized security policy management across the organization.

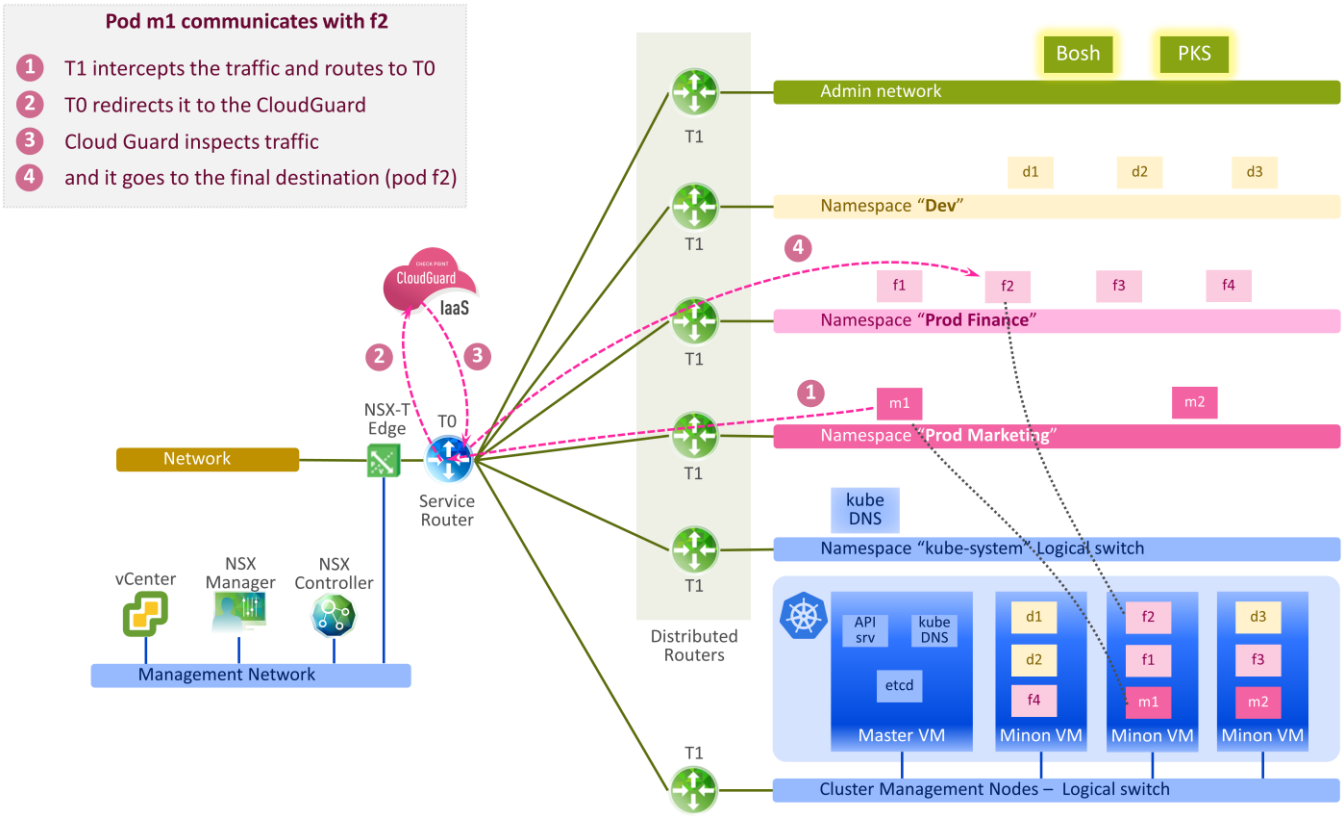


Figure 21 VMware NSX-T redirects intra-pod traffic to the CloudGuard IaaS

This allows North-South traffic inspection as well as East-West. Even traffic between pods f1 and f2 from the same (Finance) namespace is inspected with the Check Point security gateway. And with all this going on, CloudGuard IaaS can enforce k8s aware policies based on security tags/labels, which makes it efficient even in the rapidly changing environments.

Web Application and API Protection (WAAP)

Instead of decrypting traffic on the external load balancer, in Kubernetes environments, the agent can be installed as an Nginx Ingress controller, which will do the encryption and provide additional security protection. The agent protects all incoming traffic similar to an API Gateway (or right after the API Gateway) immediately before the nodes and pods in the cluster.

CloudGuard WAAP (Web Application and API Protection) is used to secure an organization's web applications. It does this by analyzing web transactions using a set of AI engines working in unison. CloudGuard WAAP protects against sophisticated attacks.

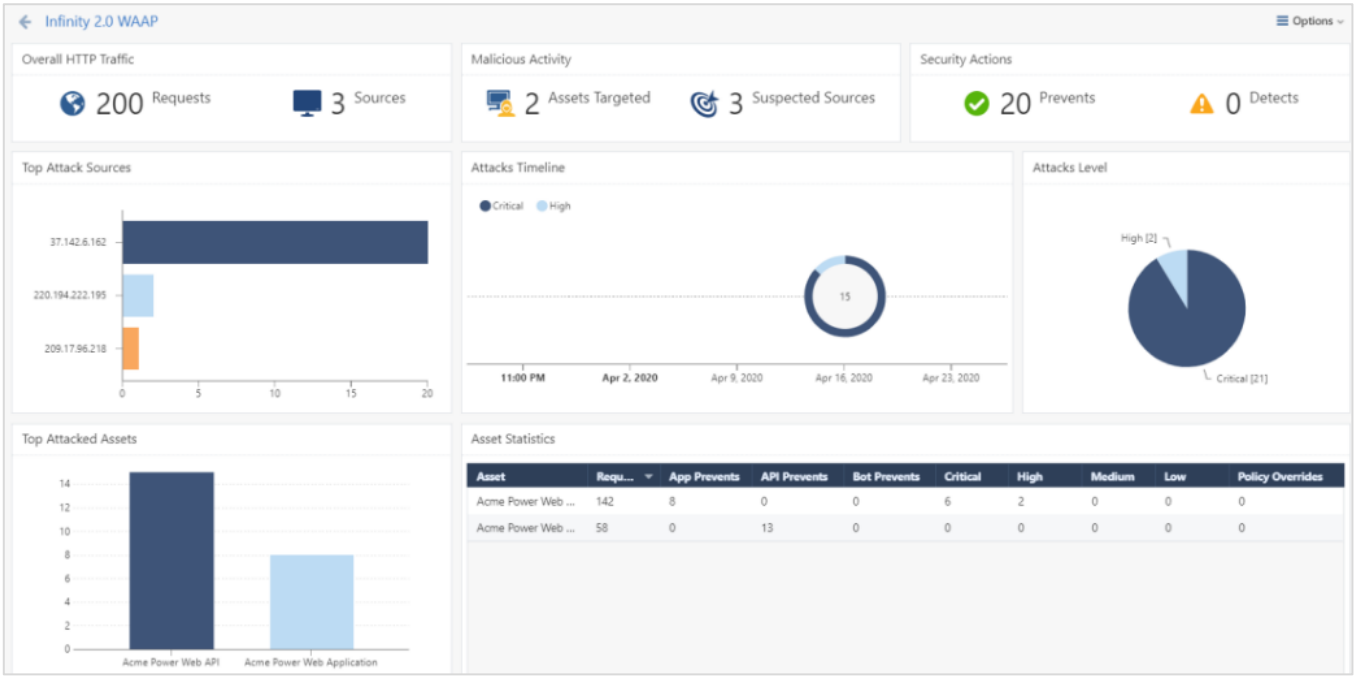


Figure 22 WAAP Dashboard

There are multiple ways of installing the WAAP agent:

- On the existing Nginx Web Server
- As a Container
- As an Nginx Ingress controller on a Kubernetes
- As a dedicated agent (reverse proxy VM) running on different cloud environments/VMware

CloudGuard WAAP has three major security components: Web application protection (WAF), API security, and Bot protection:

- Web Application Protection: OWASP Top 10 and Advanced Attacks
- API Security: Schema Validation
- Bot Protection: Distinguishes Humans from Bots

CloudGuard Posture Management

CloudGuard Dome9 is an innovative service that allows enterprises to efficiently manage the security and compliance of their Kubernetes and public cloud environments at any scale across Amazon Web Services (AWS), Microsoft Azure, and the Google Cloud Platform (GCP). CloudGuard Dome9 offers technologies to visualize and assess security posture, detect misconfigurations, model and actively enforce gold standard policies, protect against attacks and insider threats, provide cloud security intelligence for cloud intrusion detection, and comply with regulatory requirements and best practices.

Dome9 is 100% API driven – connecting to native cloud provider APIs, agentless, with infrastructure-free architecture. It supports all three major clouds – AWS, Microsoft Azure and Google Cloud Platform (GCP), as well as Kubernetes.

- Two modes of operation – read-only and manage.
- Read-only will monitor cloud accounts for changes and provide alerts – useful for Proof of Concept deployments on production environments.
- Manage security groups for all clouds from a single point.
- Provide tamper protection on security groups.

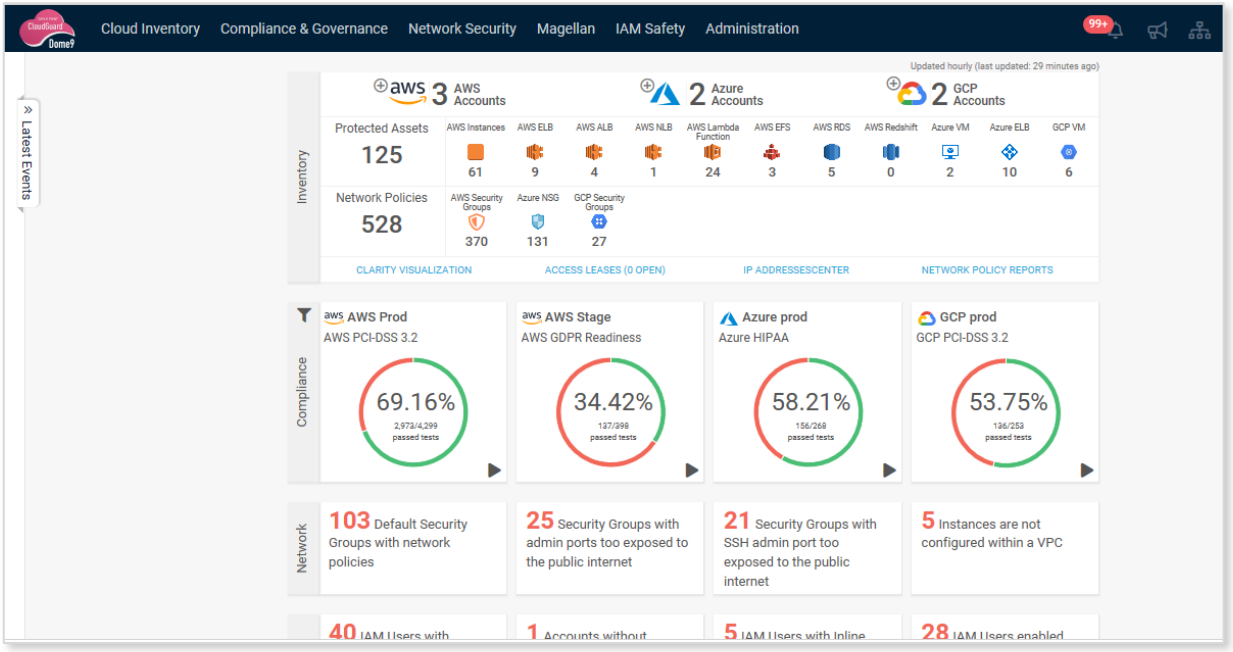


Figure 23 Dome9 dashboard

There are several frameworks to start addressing k8s misconfigurations. In particular:

CIS Benchmarks for Kubernetes – CIS has worked with the community since 2017 to publish a benchmark for Kubernetes. This benchmark is very prescriptive and provides very detailed risk and remediation details regarding the most significant security requirements broken down into the following domains: Master Node Security Configuration (Scheduler, Controller Manager, Configuration Files, etcd, PodSecurityPolicies), Worker Node Security Configuration (Kubelet,Configuration Files).

NIST 800-190 Application Container Security Guide – NIST special publication dedicated to Containers, provides high level details regarding the security risks involved with containerized apps and the effective security measures necessary to mitigate these risks. This is a great framework that is relevant for any company(not only federal government) that uses containerized applications. It Covers the following areas: Image Risks, Registry Risks, Orchestrator Risks, Container RIsks, Host OS Risks.

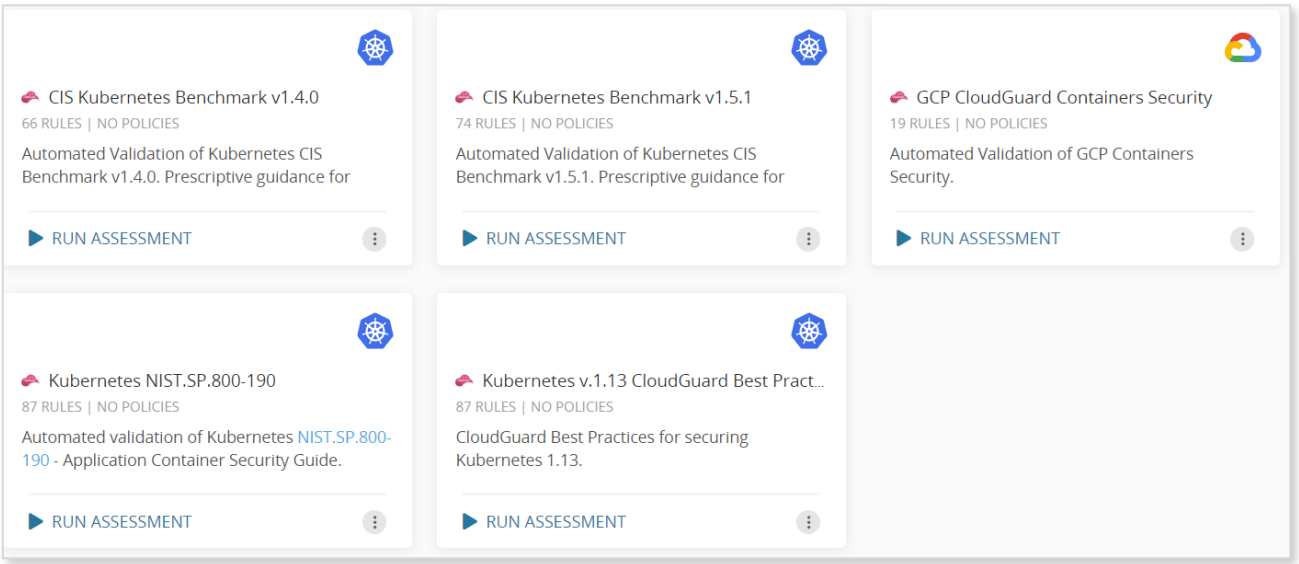


Figure 24 Dome 9 rulesets for Kubernetes

Log.ic

CloudGuard Log.ic delivers advanced cloud intelligence and simplified visualization for faster and more efficient incident response, augmented through AI and ML, across multi-cloud environments and Kubernetes on-prem installations.

- Cloud security intelligence and security analytics, delivering enhanced cloud security processes and decisions with contextualized visualization, intuitive querying, intrusion alerts, and notifications of policy violations.
- Real-time cloud security monitoring and protection, troubleshooting, and security posture awareness for ephemeral assets from Amazon AWS, Google Cloud Platform (GCP), and Azure.
- Simplified threat hunting, data analysis, and forensics, adding real-time contextualized insights into threats and risk levels.
- Automated Detection and remediation of vulnerabilities, anomalies, and security threats in multi-cloud environments.
- Seamless SIEM integration for contextualized and more actionable insights.
- Advanced Threat Prevention: Detect anomalies, alert, and quarantine threats, while utilizing cloud security analytics, threat intelligence feed, and encryption.

CloudGuard Log.ic takes traffic logs and enriches the data with threat cloud IOC information and other types of enrichment. It provides users with the ability to visualize the data flows and run GSL queries for immediate incident response and threat hunting purposes.

- **Consume CloudTrail and VPC Flow Logs** to provide an augmented view of your AWS cloud security.
- **Visualize and analyze** network activity and traffic into and out of your cloud environment.
- **Near real-time view** of events.

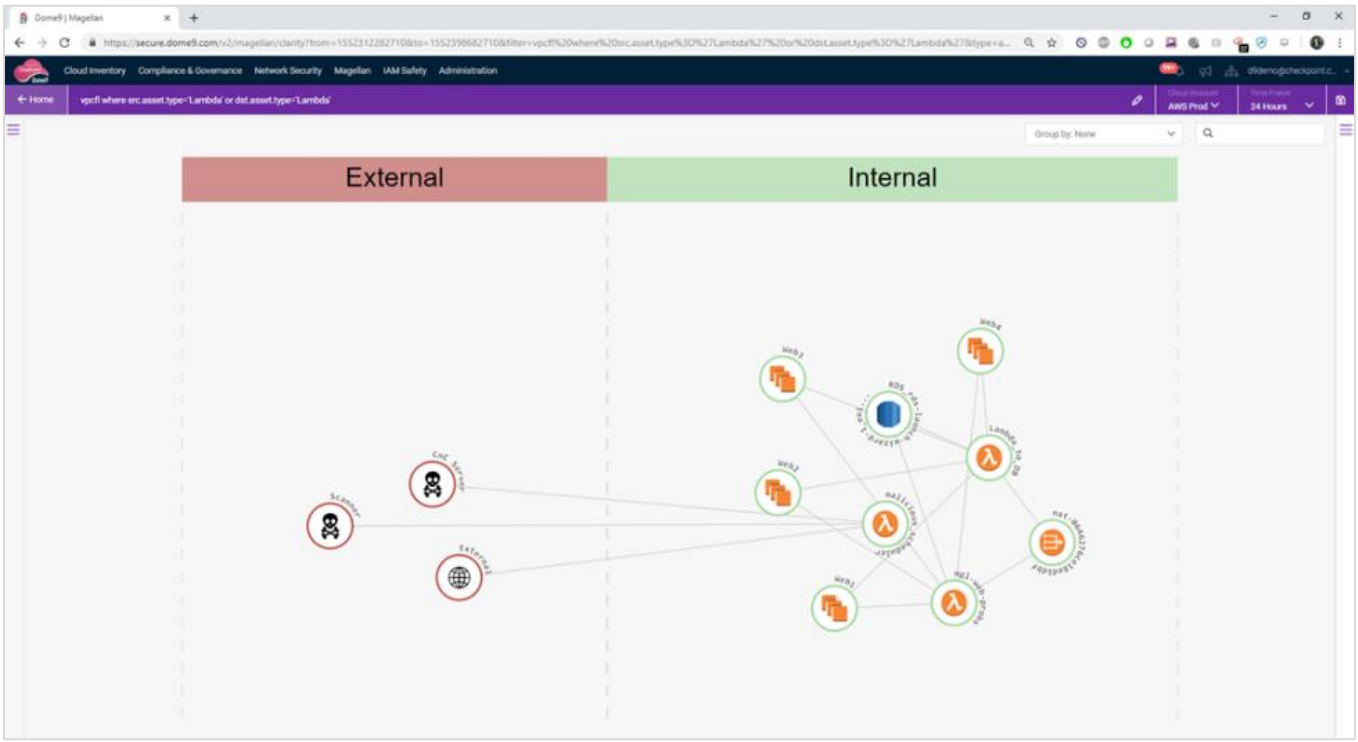


Figure 25 Visualize network activity

You may **visualize** your Cloud Network Policy, **run in detect mode** to automatically generate the appropriate Network Policy based on existing traffic, and also **edit** the policy directly from the dashboard.

- **Clarity** provides a pictorial view of cloud network environments.
- **Trace traffic flow** between points.

- **Effective View** provides aggregated (effective) policy display of instances assigned to any given virtual network.
- Get **contextual** information about cloud objects.
- **Create, edit, and manage** network security groups from a **single location across clouds**.
- **Tamper Protection** prevents unauthorized security group changes.
- **Region Lock** adds new security groups to Dome9 in Full Protection mode and ensures all inbound and outbound rules will be cleared.

Admission Control

Kubernetes offers an "admission control plug-in" which is a piece of code that intercepts requests to the Kubernetes API server prior to the persistence of the object, but after the request is authenticated and authorized. This is an important way to expand Kubernetes security mechanisms.

For instance, many important features in Kubernetes rely on proper labeling. However, by default, any admin with permissions to assign labels may choose any of them. This means that even very strict policy (for example, when using third party solutions with extended network policies like Calico) becomes useless if everybody can assign a label like "AllowInternetAccess", granting such access to their pods. Eventually, access to other services (like finance) can be obtained by such mislabeling.

Check Point Admission Controller intercepts requests to the Kubernetes API server and provides a very convenient way to extend a Dynamic Admission Control to improve security, enforce governance, and validate configuration changes.

Below are some examples:

- Does not allow users to run pods without network policy;
- Limits the list of labels available for admins/roles;
- Specifies namespaces in which an admin can create/manipulate objects;
- Detects and prevents unusual behavior of the admin (for example, if he/she tries to run too many containers per hour);
- Verifies if the starting pod was created from the image verified by the CI/CD process, from the trusted repository, etc.

These rules can be written in a convenient way using GSL (a simple language used for writing compliance rules in the Dome9).

Simple use case:

- Regular admin may use labels to control traffic within his/her namespace only.
- Only a security officer may assign labels granting Internet access to pods.

Vulnerability Assessment

Containers are usually built on top of images from public registries and with packages added from public repositories. They can be intentionally compromised and replaced by hackers after hacking the registry. The use of repositories (RPMs and other dependencies) may also not be safe.

Required features:

- **CI/CD Integration** to prevent deployment of containers with known vulnerabilities, bad code, etc.
- **Image scanning at** runtime to alert regarding containers with newly discovered vulnerabilities (after deployment)

Workload Protection

CloudGuard Workload Protection offers automated security for different workloads, in particular AWS Lambda functions and Kubernetes. It provides seamless vulnerability assessment and creates deep code flow analysis and behavioral profiles in runtime for unmatched protection – with a minimal performance impact.

Main principles:

- **Observability:** Continuously scans functions, to increase security posture - providing observability, and continuous assessment.
- **Least Privilege Protection:** Maximizes workload security through automatic least privilege protection for functions, logs, and databases.
- **Threat Prevention:** Zero-touch serverless application security using pattern matching, whitelisting, blacklisting; mostly applied at the function level for threat prevention.

Serverless Runtime Protection[3]

Runtime protection can **identify misbehaving containers** and either alert on or kill them.

It allows the study of newly created deployments to build a safe baseline for your pods, and alert on or kill pods which stray from the baseline.

Applicable to serverless applications (Java, Node, Python, C Sharp). In particular:

- **Workload Agnostic Application-level Firewall**
  Scan inputs and outputs for functions for malicious data patterns such as SQL and Code injections, and other OWASP Top-10 Risks.
- **Cloud-Application Distributed Security Monitoring**
  Observe and report sensitive behaviors such as cloud resource access, network access, and process and file access.
- **Automated Micro-segmentation Enforcement**
  A backend learns necessary function behavior by combining code-flow analysis and runtime monitoring and enforces these policies.
- **Organizational Zero-Trust Policy Enforcement**
  Beyond the automated micro-segmentation, it also enforces policies that are overlaid by the security organization to prevent application risk.

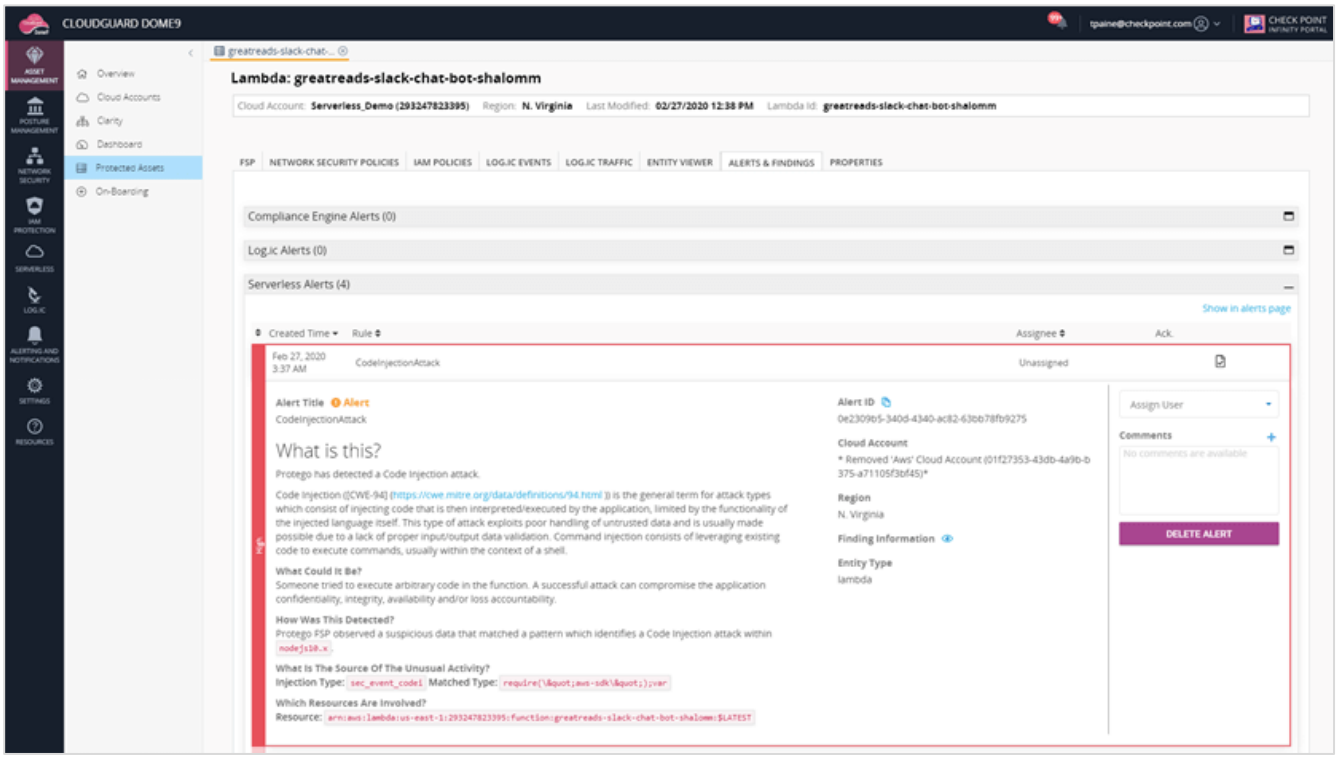


Figure 26 Serverless protection

[3] In Q3 2020 mostly applicable to AWS.

CONCLUSION

We designed paper with a view that it would inform our audience on what a cloud-native environment might look like, the various components and the security implications.

Through our consultancy services,[4] we can attest to the real-world requirement to be “cloud-native” as much as possible. However, we also realize the potential security pitfalls and challenges, clearly, as the technology running our workload changes, so should our approach to security, that is why understanding the concepts of container story, CSPM and CNAPP are so important.

To conclude, as the digital transform expedites the shift to cloud, where the desired sate is cloud-native, the Check Point platforms and tools exist to secure the journey.

CONTACT US

Worldwide Headquarters | 5 Shlomo Kaplan Street, Tel Aviv 67897, Israel | Tel: 972-3-753-4555 | Fax: 972-3-624-1100 | Email: info@checkpoint.com **U.S. Headquarters** | 959 Skyway Road, Suite 300, San Carlos, CA 94070 | Tel: 800-429-4391; 650-628-2117 | Fax: 650-654-4233 | www.checkpoint.com

[4] https://www.checkpoint.com/downloads/products/checkpoint-cloud-transformation-security-consultancy.pdf