



CLOUD-NATIVE DEFENSE-IN-DEPTH SECURITY FOR MISSION-CRITICAL SERVICES IN MANAGED KUBERNETES



AGENDA

INTRODUCTION

MISSION-CRITICAL SERVICES

SECURITY CHALLENGES IN MANAGED
KUBERNETES

DEFENSE-IN-DEPTH ARCHITECTURE

SECURING CLOUD BOUNDARY

SECURING CLUSTER BOUNDARY

SECURING CONTAINER BOUNDARY

SECURING CODE BOUNDARY

KEY TAKEAWAYS





INTRODUCTION

- Kubernetes is the foundation for modern cloud-native infrastructure.
- Managed services: AKS, EKS, GKE simplify operations but add security risks.
- Mission-critical services require strong, layered security.
- This work proposes a defense-in-depth model using the 4Cs.



MISSION-CRITICAL SERVICES

Definition: Software critical to business operations.

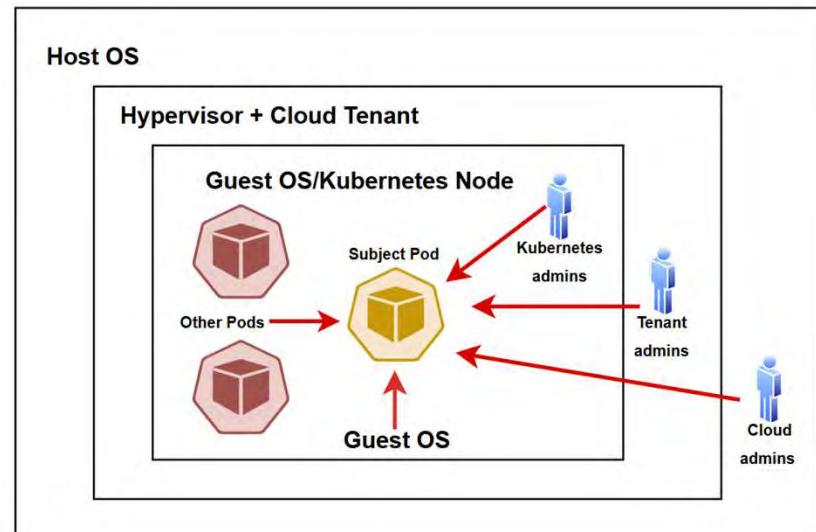
Examples:

- Finance: Payment processing, fraud detection
- Healthcare: EHR, medical monitoring
- Defense: Secure communication, analytics
- Supply Chain: Code signing, scanning, packaging



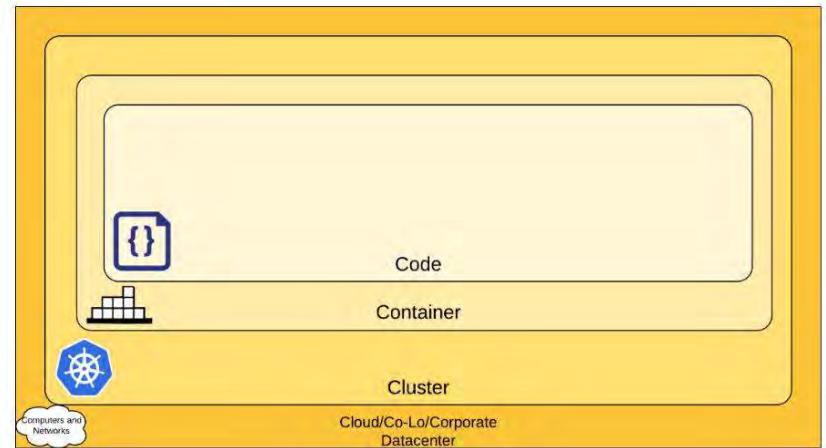
SECURITY CHALLENGES IN MANAGED KUBERNETES

- Insider threats from admins (Cloud, Cluster, Tenant).
- Misconfigurations and weak authentication.
- Malware injected into containers.
- Runtime exploits & privilege escalation.
- Supply chain compromises.



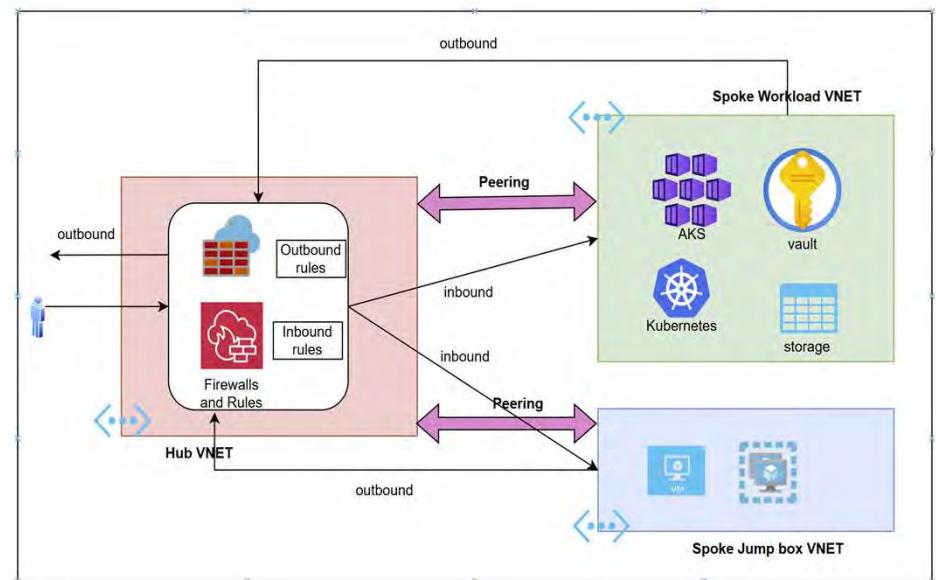
DEFENSE-IN-DEPTH ARCHITECTURE

- Layered controls across Cloud, Cluster, Container, Code (4Cs)
- Zero trust enforcement at all levels.
- Mitigates insider threats, supply chain risks, runtime exploits.
- Built-in integrations with AKS, EKS, GKE.



SECURING CLOUD BOUNDARY

- **Hub-Spoke Network Topology** – Centralized hub VNet manages traffic and security policies, while spoke VNs isolate workloads for stronger segmentation and visibility.
- **Private Clusters & Endpoints** – Use private IP addresses for control plane and registries, ensuring sensitive traffic stays within the internal network and is not exposed to the internet.
- **Restrict Inbound and Outbound Connectivity Using Firewall Rules** – Define strict rules for ingress and egress to validate only authorized communication paths, blocking unauthorized traffic.
- **Protects Workloads from External Threats** – Combined measures minimize exposure, shrink the attack surface, and provide strong boundaries against internet-based attacks.



WHAT SECURITY GAP STILL EXISTS AFTER THE FIRST C - CLOUD?

- The Cloud layer focuses primarily on securing the external network boundary. It provides protection from internet-based threats through firewalls, network segmentation, and controlled connectivity.
- At this stage, all inbound and outbound communication is monitored and validated to ensure that only trusted traffic flows into or out of the Kubernetes environment.
- However, this protection is limited to the network perimeter. While it successfully reduces exposure from external actors, it does not address risks that exist inside the environment.
- The next layers — Cluster, Container, and Code contain components that remain vulnerable to internal threats, privilege misuse, and runtime exploitation.

SECURING CLUSTER BOUNDARY

- **Private API Server & RBAC Integration** – Use private endpoints to restrict exposure and enforce strong authentication and role-based access controls for least-privilege access.
- **Secure etcd with KMS Encryption** – Encrypt sensitive cluster state and secrets at rest using cloud Key Management Services to prevent unauthorized data access.
- **Protect Kubelet and Node Access** – Disable anonymous kubelet access, restrict kubelet ports, and enforce network policies to harden node-level communication.
- **Network Segmentation** – Apply Kubernetes Network Policies or cloud-native firewalls to control pod-to-pod, namespace, and external traffic, limiting lateral movement.
- **OPA/Gatekeeper Admission Control** – Enforce compliance and security rules (e.g., disallow privileged pods, enforce image signing) before workloads are admitted into the cluster.

WHAT SECURITY GAP STILL EXISTS AFTER THE SECOND C - CLUSTER?

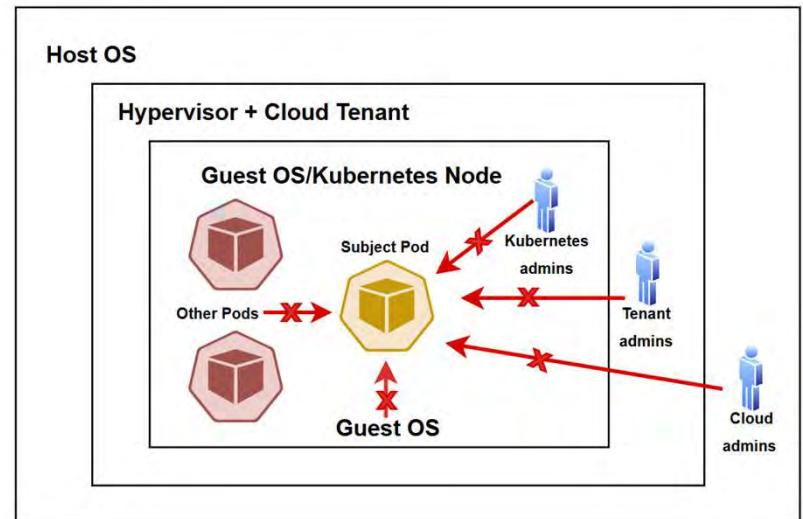
- Covered key approaches to secure the managed Kubernetes cluster and its internal components.
- Network (Cloud layer) and Cluster layer security controls are in place.
- Despite this, containers remain vulnerable to exploitation.
- Attackers may still target container breakouts, kernel flaws, or malicious code injection.

SECURING CONTAINER BOUNDARY

- **Pods at Risk of Breakout or Kernel Exploitation** – Compromised containers can escape to the host or exploit kernel vulnerabilities, threatening the entire cluster.
- **Confidential VMs Protect Node-Level Workloads** – Hardware-backed Trusted Execution Environments (TEEs) encrypt memory and isolate worker nodes, shielding workloads from host-level attacks.
- **Confidential Containers Secure Sensitive Apps in TEEs** – Containers run in hardware enclaves, ensuring application code and in-memory data stay protected even from compromised kernels or runtimes.
- **Prevents Host or Admin Tampering** – Confidential computing prevents even privileged administrators or cloud operators from accessing or modifying sensitive workloads.

WHAT SECURITY GAP STILL EXISTS AFTER THE THIRD C - CONTAINER?

- Vulnerabilities from hypervisor, host/guest agents, and peer pods/containers are mitigated.
- The remaining risk lies in the code running inside the container itself.
- Even with a Trusted Computing Base (TCB), runtime code can introduce threats (e.g., downloaded code, spawning new processes).
- This gap is addressed by securing the fourth “C” – Code.

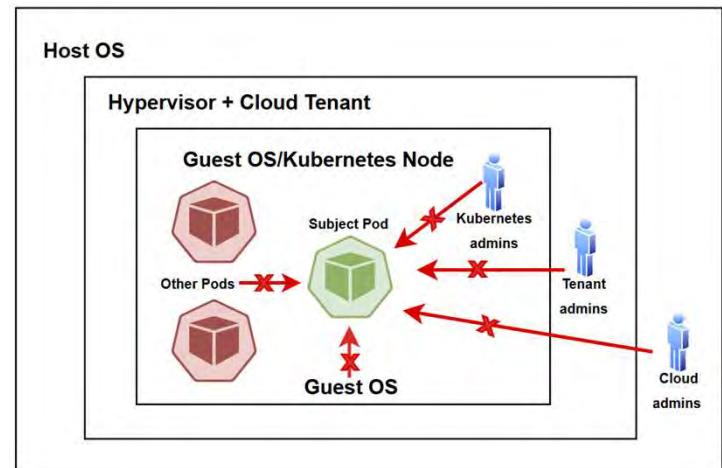


SECURING CODE BOUNDARY

- **Trusted Image Registries and Signing** – Ensure only verified, cryptographically signed container images from trusted registries are deployed, reducing supply chain risks.
- **Use distroless images for Linux workloads** - Distroless images are secured as it does not have any bash/shell and package manager installed and it contains only required software for application to run
- **Immutable Containers** – Containers cannot be modified after deployment; updates require rebuilding and redeploying, preventing runtime tampering.
- **Seccomp Limits Runtime Syscalls** – System call filtering restricts what processes inside containers can execute, blocking actions like creating new processes or mounting filesystems.
- **Falco Detects Abnormal Runtime Behavior** – Provides real-time monitoring of container activity, alerting on suspicious actions such as privilege escalation or unexpected file access.

WHAT SECURITY GAP STILL EXISTS AFTER THE FOURTH C - CODE?

- **Immutable Containers + Seccomp strengthen the Code layer** - Prevent unauthorized modifications, block risky system calls, and restrict container process behavior.
- These measures fully secure the fourth “C” (Code), closing the final gap in the defense-in-depth model.



KEY TAKEAWAYS

- **Protect Mission-Critical Services:** Defense-in-depth secures essential workloads in managed Kubernetes.
- **Beyond Cloud-Provider Security:** Regular cloud security (IAM, networking, perimeter controls) is not enough against modern, multi-vector threats.
- **Defense-in-Depth Advantage:** Multi-layered protection (4Cs, zero trust, runtime monitoring, confidential containers) addresses insider risks, supply chain threats, and runtime attacks.
- **Resilient and Developer-Friendly:** Strong security posture without reducing developer productivity.



THANK YOU