Cloud-Native Defense-in-Depth Security for Mission-Critical Services in Managed Kubernetes

Karthikeyan Thirumalaisamy

Independent Researcher, Washington, USA
Corresponding Author Email: *kathiru11[at]gmail.com*

Abstract: ***Kubernetes serves as the standard deployment tool for cloud-native applications that include mission-critical services across industries such as finance, healthcare, defense, and more. When many organizations start using Kubernetes especially through managed services like AKS, EKS, and GKE which brings additional security challenges that need to be carefully addressed. This paper presents a secure architecture for managed Kubernetes systems which implements defense-in-depth methodology based on the cloud-native security 4C's: Cloud, Cluster, Container, and Code. The paper analyzes each component of managed Kubernetes systems to demonstrate how cloud-native security mechanisms combine to protect against insider threats and software supply chain attacks and escape runtime exploits through cloud-level policy enforcement and cluster hardening and secure containerization and secure development practices. The paper proposes a security architecture delivers operational security controls through admission controllers and role-based access controls (RBAC) and seccomp and runtime detection tools and secure image registries by utilizing AKS, EKS and GKE cloud-native features and integrations. The paper includes concrete execution examples demonstrating how a defense-in-depth security architecture built on Kubernetes technology implements comprehensive security measures at each layer, ensuring robust protection without slowing down development speed or reducing operational capabilities.***

Keywords: Kubernetes, Supply chain, Cloud-Native architecture, Defense in depth, zero trust architecture

1. Introduction

Kubernetes has been foundational to modern cloud-native infrastructure that enables organizations to deploy and scale containerized workloads with a high degree of availability and automation. Managed Kubernetes services such as Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (EKS), and Google Kubernetes Engine (GKE) provide operational simplicity by abstracting control plane management, allowing teams to focus on workload orchestration. Nevertheless, this convenience introduces additional challenges in ensuring the security of mission-critical services.

Mission-critical services have rapidly emerged as some of the most common distributed microservices deployed within managed Kubernetes clusters across industries. These services often consume sensitive metadata, cryptographic materials, or business-critical software which in turn provides an attractive target footprint for adversaries. Providing security to such services in managed Kubernetes environments requires thinking holistically about how to support workload isolation, runtime protection, and policy enforcement across cloud infrastructure and containerized applications.

This paper proposes a cloud-native security architecture to safely secure mission-critical services in managed Kubernetes environments by employing a defense-in-depth model. Our architecture is built around the 4C's of cloud-native security systems such as Cloud, Cluster, Container, and Code which together illustrate the layered control surfaces in the Kubernetes stack. By looking at all the relevant threats and controls for each layer, this paper presents a systematic approach for addressing common risk scenarios, such as lateral movement, privilege escalation, insecure configurations, and secret/workload access by unauthorized entities.

At the heart of the proposal is the use of confidential containers, which leverage hardware-backed Trusted Execution Environments (TEEs) to ensure sensitive workloads are protected not just at rest or in transit, but also while executing. Confidential containers allow a critical set of supply chain functions to be executed in an isolated space with encrypted memory even from a compromised kernel or a compromised administrator running on an underlying host. Confidential containers are intended to bring strong assurances against insider threats and advanced persistent threats.

The proposed architecture comes with the built-in capabilities of managed Kubernetes platforms but also leverages a wide set of open-source tools and standards. Some examples of open-source security tools that we will consider are seccomp profiles, RBAC and Network Policies, admission control mechanisms (e. g., OPA/Gatekeeper), and runtime threat detection (e. g., Falco). By using practical deployment patterns and real-world implications of secure supply chain services in managed Kubernetes, it will demonstrate how the defense-in-depth model can be applied to secure supply chain services without negatively impacting operational efficiency or scalability.

2. What is Mission-Critical services?

Mission-critical services are software programs or systems that are critical to an organization’s core business operations, where failure or downtime could equate to financial loss or compromised safety, regulatory compliance or reputational risks or damage. Mission-critical services usually deal with sensitive or regulated data and require high security, reliability and resiliency in terms of being unscathed from attacks, failures, and outages. Mission-critical services are foundational across multiple industries, especially finance, healthcare, defense and utilities.

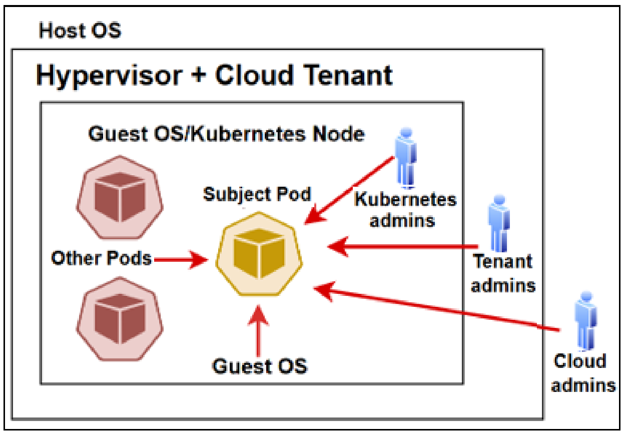
Some examples of mission-critical services globally are:
- Finance: Payment processing, fraud detection, and trading software.
- Healthcare: Electronic health record systems, medical device monitoring, and tele-health software.
- Defense: Secure communication, command & control systems, intelligence information and analytics.
- Utilities: Electricity grid management and water treatment system.
- Transportation: Air traffic control and logistics software.
- Supply Chain Services: Code Signing, Scanning, Packaging

In today's cloud-native environments, many mission-critical services are used as containerized microservices on flexible managed Kubernetes services offered by such providers as Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (EKS), and Google Kubernetes Engine (GKE). Managed Kubernetes services provide scalable, reliable and secure orchestration to support organizations in safely maintaining continuous operations, implementing workload isolation capabilities, and maximizing security posture requirements for mission-critical workloads. Yet, managed Kubernetes services raise several important security concerns that organizations should consider, including risks of hypervisor or host compromise, virtual machine vulnerabilities, potential for API server breaches, threats within cloud environments, and misuse of tenant administrator access.

3. Security Concerns of Building Services on Managed Kubernetes

Security concerns grow in complexity when it comes to Kubernetes and container technologies. Two major potential threats affecting Kubernetes are:
- **Malicious actors –** Once a threat actor breaks into your Kubernetes ecosystem, it becomes easy for them to spread the malicious actors across the cluster. This is possible because containers and pods interact with each other, and a corrupted container could cause the collapse of the entire application. Attackers are constantly looking for exposed containers or portals with either no authentication system or a poor one. Such containers often fall into blind spots; way too often, the organizations will not even realize the exploitation or the breach by compromised or rogue users.
- **Malicious code running inside containers –** Attackers can exploit the misconfiguration to place malware or unknown code inside a container. In 2018, **Tesla's cloud infrastructure was breached**, and crypto-mining malware was placed deep within the environment. The investigation revealed that a particular Kubernetes administrative portal wasn't password protected. Another violation method is attacker leveraging vulnerabilities in container images and image registries.

Below picture illustrates the major security concerns of running services in Managed Kubernetes,



As shown, actors whose identity might be compromised or who might act maliciously include Kubernetes cluster admins, Tenant admins and Cloud admins. These are represented in red arrows in the picture above. The subject pod that is running application code and other pods such as system pods might be compromised by running injected malicious code by varies means, including contaminated container images or code downloaded by containers at run time. These threats are represented as red cubes in the above picture, indicating containers themselves might be infected with malicious code.

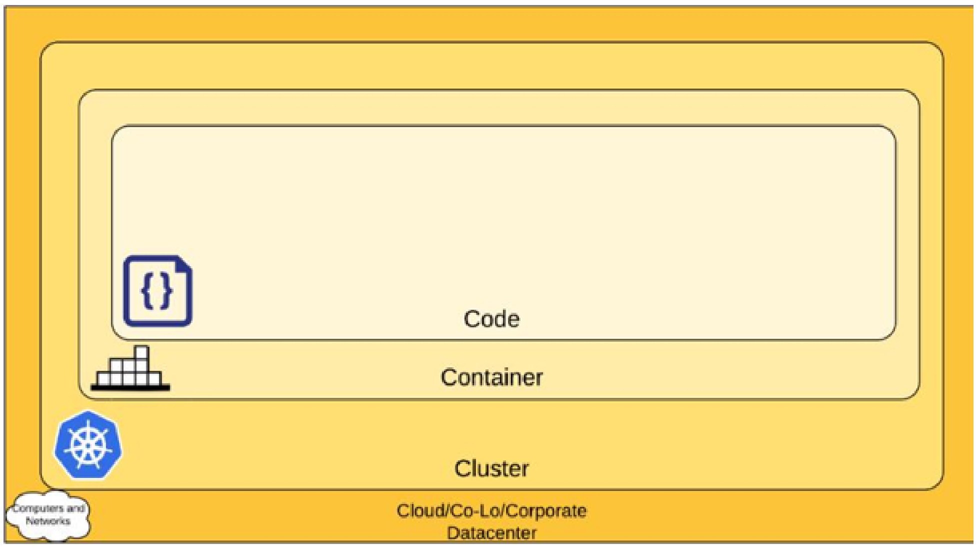
To mitigate these threats, it is important to establish zero-trust security boundaries around containers running in Kubernetes and lock down access to fend off all actors including cloud admins, tenant admins and Kubernetes cluster admins. In addition, it is important to establish security mechanisms to ensure code running inside containers can't be tempered at any time. These include container image integrity and policies enforcing running only allow code both at the startup of containers and throughout their life cycle at the runtime.

4. Cloud Native Security Principles (4 Cs)

Cloud Native Security Principles think of security in layers. The 4Cs of Cloud Native Security Principles are **Cloud, Clusters, Containers** and **Code**. This layered approach augments the defense in depth computing approach to security, which is widely regarded as a best practice for securing software systems. Each inner layer of the Cloud Native security model builds upon next outer layer. The Code layer benefits from strong base security layers of Container, Cluster and Cloud.

By applying consistent security controls across all four layers, organizations can create a resilient and scalable defense strategy that aligns with modern cloud-native practices. This layered model is particularly effective for securing mission-critical services, which span across multiple levels of the stack and require end-to-end protection.

The consideration of security controls at each of the four Cs is described below:



- **Cloud:** Securing cloud involves protecting Cloud infrastructure from unauthorized access from actors and code. Basic measures include identity and access management (IAM), JIT access, and RBAC for cloud resources. Additionally, network segmentation and firewalls are needed to enforce layers of security boundaries the ensure least access privilege to most inner and core assets.
- **Cluster:** Securing cluster involves securing both cluster control plane components like API calls, node access, etcd access, and applications running inside the cluster. The security measures include security practices like encrypted communication and TLS certificate authorization, securing Kubernetes API server by enforcing API authentication mechanism. Once again, it is important to enforce layered security boundaries to protect Kubernetes control plane, nodes, pod networks, pod level security policies, namespaces, and network segregation.
- **Container:** Securing container involves isolating containers from outer layers. Any malicious actor that breaches the container layer can move within the environment, easily communicating with other containers and pods. In its default mode, Kubernetes offers minimal security guardrails to ensure faster software development. One can improve Container level security by hardening the security with robust security governance policies and controls, but this is not enough to address the inherited vulnerability of hypervisor architecture, that is, the guest OS can be penetrated via hypervisor from host OS. This "open" passage from host to guest can't be sealed with access controls. A complete isolation of containers from the outside world is needed to ensure complete security at Container level.
- **Code:** Securing Code involves preventing malicious code running inside containers. This is one of the most targeted attack surfaces for any computing environment. Security measures to secure Code include ensuring container images are created and stored in trusted image registries, container images are malware free, and container images are verified before being deployed. In addition, it is critical that containers cannot be tempered at runtime to run any malicious code.

5. Designing Defense-in Depth Security architecture for Managed Kubernetes

Managed Kubernetes refers to cloud-native Kubernetes solutions like Azure Kubernetes Service (AKS), Amazon Elastic Kubernetes Service (EKS), or Google Kubernetes Engine (GKE). Essentially, the cloud provider takes responsibility for provisioning, operations, and maintenance of the Kubernetes control plane. The hybrid cloud managed services also abstract many of the operational burden of running Kubernetes at scale in terms of managing the control plane upgrades, availability, and even scalability of the control plane.

To put it another way, managed Kubernetes simplifies the operation of running Kubernetes but with a trade-off of a shared responsibility model. The managed service provides security on their infrastructure and control plane layer while the consumer is responsible for securing their workloads running in Kubernetes, configurations, network policies, and runtime behavior. This makes it simple for teams to adopt cloud native architectures quickly but a good security architecture and process at the cluster and application layer is paramount to keep pace with the evolving threat landscape as threat actors continue to evolve their behavior and motivations.

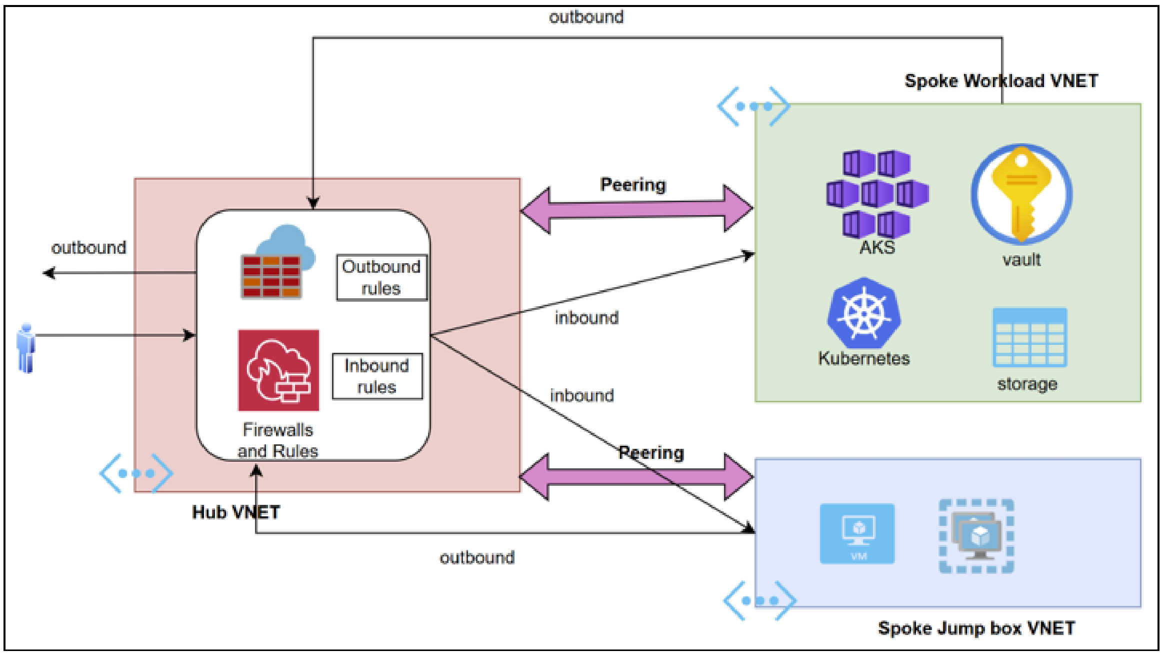
Security is defined as the actions, processes and principles that should be followed to ensure security in your Kubernetes deployments. This includes securing code, configurations, containers, Kubernetes Cluster, Kubernetes network, and Cloud infrastructure.

5.1 Securing Cloud Boundary for Managed Kubernetes

The process of securing the cloud boundary for managed Kubernetes starts with establishing strong network-level protections. Although the Kubernetes control plane is managed by a cloud provider, it is ultimately the users responsibility to provide workload isolation and secure communication paths. This section discusses network security approaches private cluster configurations, virtual network integration, restricted API server access, and firewall rules that can help to create a secure perimeter around managed Kubernetes clusters. Proper network segmentation ensures that only authorized and trusted entities can interact with the cluster, thereby minimizing lateral movement and reducing exposure to external traffic.

5.1.1 Securing Network using Hub Spoke Network Architecture

Network security is the protection of the underlying networking infrastructure from unauthorized access, misuse, or theft. It involves creating a secure infrastructure for devices, users, and applications to work securely. Network security combines multiple layers of defenses at the edge and in the network. The following diagram illustrates the sample deployment of managed Kubernetes cluster in Hub spoke architecture pattern behind firewall.



Each network security layer implements policies and controls. Authorized users gain access to network resources, but malicious actors are blocked from carrying out exploits and threats. Hub-spoke network pattern where the hub virtual network (VNet) acts as a central point of connectivity to many spoke virtual networks. The spoke virtual networks (VNet) connect with the hub and can be used to isolate workloads.

A hub is a centralized network location that controls, inspects, and routes the pathways on traffic traversing a number of different connected environments, such as the internet, on-premises data centers, and spoke virtual networks. In the hub-and-spoke topology, the hub serves as the single control point to manage inbound and outbound traffic for all associated spoke networks, commonly referred to as virtual networks or VNets. Inbound rules manage and control ingress traffic arriving from external sources, either from the internet or on-premises networks, before it reaches a spoke VNet running sensitive workloads, such as managed Kubernetes clusters. Outbound rules manage and control egress traffic leaving from spoke VNets, permitting egress only to approved destinations, while preventing unauthorized communication to an untrusted or external network. This architecture promotes enforcing centralized security policies, unified logging, and deep packet monitoring while minimizing the number of points of external access at the edge of the network, consequently shrinking the attack surface. Collectively managing network controls, such as Azure Firewall, AWS Network Firewall, Network Security Groups (NSGs), and intrusion detection within the hub, allows organizations to maintain a consistent security posture across different spokes, lower the risk of misconfiguration and keep visibility into all communications between VNets or networks. The role of each spoke is to host different types of workloads. The spokes also provide a modular approach for repeatable deployments of the same workloads. It is needed to provision a Kubernetes cluster in the Spoke workload virtual network (VNet) and also provision a second spoke VNT for Jump Box Virtual Machines (VMs). These Jump Boxes will provide developers and administrators with secure access to the Kubernetes cluster, which adds another layer of protection from external threats.

5.1.2 What security gap still exists after the first C?

Securing the network is part of the first ‘C’ (Cloud) in Cloud Native security model that only protects from internet threats, and it ensures that all the inbound and outbound connectivity is monitored and validated using Firewall rules. Additionally, the actors of admins are enforced with access controls (RBAC). But we still need to secure inner components (the remaining 3C’s Cluster, Container and Code) using different approaches which will be discussed further in detail.

5.2 Securing Cluster Boundary for Managed Kubernetes

In managed Kubernetes (AKS, EKS, GKE, etc.) establishing defenses to protect the cluster boundaries is vital to ensuring workloads do not pose undesired access, risk of data exfiltration, and malicious changes to configuration. For this purpose, the boundaries of the cluster can be defined as the control plane and any resources that need to be isolated from anything beyond untrusted (unverified) networks while still allowing for securely logged, auditable, and policy compliant operations within the cluster.

5.2.1 Private Cluster

By default, managed cluster uses a public IP address for accessing the Control Plane. However, using a public IP address will expose the control plane traffic to Internet threats. Hence, by creating a private Kubernetes cluster with a private control plane IP address, we can ensure network traffic between your API server and your node pools remains on the private network so that the cluster is protected from internet security threats. However, the absence of a public IP address prevents you from directly connecting to a private AKS cluster from your computer. In this case, developers and administrators use jump box from Spoke VNet to connect to managed Kubernetes cluster using a private endpoint.

5.2.2 Private Container Registry

In a Managed Kubernetes environment, a container registry is the primary repository for all container images. The registry should be private to reduce the risk of exposing over internet and unauthorized access, as well as supply chain risks. Private registries are incredibly important, as authenticating the registry will allow the only namespaced access for trusted entities to pull or push images. In addition, the registry should only accept approved and cryptographically signed images, to eliminate the possibility of Images that are vulnerable or malicious entering the environment. The Kubernetes cluster and nodes should also connect to the private registry using private, secure endpoints, as the traffic of image-pulling and pushing requests would be private in transit from the external internet.

5.2.3 Other Cloud Resources

Application runs in managed Kubernetes cluster need to talk to other Cloud resources such as Vault, Storage Cosmosdb, SQS, Service Bus, etc. By default, cloud resources are accessible over the internet so we should use private endpoint to connect to these resources and disable the public access. A private endpoint is a network interface that uses a private IP address from your virtual network. This network interface connects you privately and securely to a resource that's powered by Private Link. By enabling a private endpoint, you're bringing the resource into your virtual network (VNet) and protecting it from external malicious users directly accessing the cloud resources over the internet.

5.2.4 Securing API Server

The Kubernetes API is the front end of the Kubernetes control plane and is how users interact with their Kubernetes cluster. The API server determines if a request is valid before processing it. In essence, the API is the interface used to manage, create, and configure Kubernetes clusters. It's how the users, external components, and parts of your cluster all communicate with each other. So, If an API server is compromised then the entire cluster is compromised. The following sections discuss certain steps that we need to perform to ensure the API server is secured.

5.2.4.1 Authentication and Role Based Access Control (RBAC)

Weak authentication and authorization controls can enable an adversary to break into the Kubernetes API server, modify or delete resources, or use other commandeering actions to disrupt the entire cluster. To mitigate these risks, the API server should connect to a strong centralized identity provider for user authentication. When using an identity provider, users authenticate to the cluster using short-lived authentication tokens that are issued by the provider, so the risk of credential theft or misuse is eliminated. Upon authentication, role-based access control (RBAC) must be enforced to maintain the principle of least privilege; most managed Kubernetes services provide native integrations e. g. Azure Kubernetes Service (AKS) has Azure RBAC that can be connected to Microsoft Entra ID, and Amazon Elastic Kubernetes Service (EKS) provides AWS IAM integration for RBAC mappings. Kubernetes-native RBAC, if needed, can then provide more granular control at the namespace or cluster level based on individual or group memberships. Even where an external identity provider is used, local Kubernetes administrative accounts should be disabled unless required, because leaving them enabled allows access while bypassing centralized authentication controls.

5.2.4.2 Adhere to least privilege principle

If we need to create custom roles in Kubernetes RBAC for access control and management then use the following recommendations for permissions and role assignments:

- Avoid wildcard permissions, especially to all resources.
- Use RoleBinding instead of ClusterAdminBinding to give access within a namespace.
- Avoid adding users to the system: master group as it bypasses RBAC.
- Use impersonation rights for admins instead of adding to the cluster admin role. Audit and monitor when impersonation is being done.
- Avoid granting the escalate or bind permissions to roles when not needed, audit and monitor when escalation is being made.
- Avoid adding users to the system: unauthenticated group.
- Limit permissions to issue CSR and certificate.
- Avoid granting users with create rights on service accounts/token, which could be exploited to create TokenRequests and issue tokens for existing service accounts.
- Users with control over validating web hook configurations or mutating webhook configurations can control webhooks that can read any object admitted to the cluster, and in the case of mutating webhooks, also mutate admitted objects.

5.2.4.3 Disable API Server access from Application Pods

By default, all pods running in a cluster can access API server using auth token that is mounted automatically, due to that, if an application pod is compromised then malicious users can take full control of the cluster by compromising the API server. Additionally, every Kubernetes namespace contains at least one default service account. An application running inside a pod can access the Kubernetes API using automatically mounted service account credentials to access Kubernetes API, its level of access depends on the authorization plugin and policy in use. Therefore, we should remove Auth token mounting on the application pod and disable this capability in Kubernetes to automatically mount Service Account's API credentials for namespaces. In addition, we should configure network policies to restrict access to the API server from application pods. This topic will be covered in more detail in the Network Segmentation section.

5.2.4.4 Securing etcd

etcd is the key-value store used by Kubernetes to persist the cluster's state, configuration, and secrets. This includes highly sensitive data such as authentication tokens, TLS certificates, and other credentials stored as Kubernetes Secrets. If etcd is compromised, an attacker could gain full control over the cluster, making its protection critical.
Security for etcd should be implemented at two levels:

1) **Access Control**-Only the Kubernetes API server should have direct access to etcd. All other access should be blocked at the network and firewall level. This prevents unauthorized components or users from querying or modifying etcd data.
2) **Encryption at Rest**-All secrets in etcd should be encrypted using strong keys managed by a secure Key Management Service (KMS). This ensures that even infrastructure administrators with storage access cannot read sensitive data without proper decryption keys.

Managed Kubernetes platforms AKS and EKS provides built-in support for this using KMS plugin.

5.2.4.5 Securing Kubelet

The kubelet is an integral component of Kubernetes that runs on each node and communicates with the API server to manage the lifecycle of pods and containers. The kubelet exposes a kubelet API that if misconfigured, or compromised, can allow an adversary to execute arbitrary commands, pull down logs, or gain access to sensitive pod information. Managed Kubernetes solutions such as Azure Kubernetes Service (AKS) and Amazon Elastic Kubernetes Service (EKS) have default kubelet security configurations. Both AKS and EKS disable anonymous authentication to the kubelet, and use authorization through the Webhook mode, they also pass authorization info to Kubernetes Role-Based Access Control (RBAC). However, additional controls are always necessary to secure the kubelet in production. Kubernetes Network Policies and equivalent clouds network controls should be used to restrict kubelet (API) ports 10250 and 10255 to workloads that need it. With Pod Security Standards (PSS) or Open Policy Agent (OPA) policies, if kubelet were compromised, the actor would be limited in the impact that could be taken. For example, in AKS, administrators can deploy Azure Network Policies to allow for network segmentation. In EKS, AWS Security Groups can be configured to restrict node-level API permissions such as the kubelet API. When using managed services that have built-in protection to mitigate kubelet risk, we can layer and add additional controls to limit risk as part of the overall risk management strategy in multi-cloud Kubernetes.

5.2.5 Securing Network Segmentation

By default, all pods in any Kubernetes cluster (whether AKS, EKS, GKE, or others) can communicate with each other freely. In an open communication model, if one pod becomes compromised, it has the ability to try to communicate, and potentially compromise, other pods. Therefore, organizations should attempt to put rules in place to restrict traffic flows. For example, backend services should only be accessible by selected front-end components, and database tiers should only be accessible by the application layer that require them.

Network Policies in Kubernetes allow you to define access control for how communication takes place between pods (within a namespace), between namespaces, and between pods and external endpoints. Network Policies use label selectors to select specific pods to enforce ordered sets of ingress and egress rules. Network policies are defined using YAML manifests, and they can be deployed separately, or as a single package with other Kubernetes resources like Deployments or Services. Most managed Kubernetes platforms enable multiple network policy providers (e. g. Calico, Cilium, even native cloud network policies) that enforce the rules defined in network policies using the iptables mechanism (Linux), eBPF (Linux), or Windows Host Network Service (HNS) ACLs (Windows). The implementation of network policies will help you block any unexpected pod-to-pod traffic so that only authorized client applications can communicate with server applications.

5.2.6 Open Policy Agent and Kubernetes

OPA "Open Policy Agent" is General Purpose Policy Engine. OPA gives us a higher level and declarative language to author our policies and to enforce them within our environment, always leaving our cluster compliant with the company policies. OPA acts as an admission controller that intercepts requests to the Kubernetes API server before an object is persisted. For example, if you send a request to the API server to create a Deployment resource, the admission controller may intercept this request, mutate, or validate it. When we use OPA as an admission controller we can enforce OPA policies on these requests sent to the API server before they are processed by the API server. This gives us many benefits like:

- Disable shell access to across Pods.
- Make sure our containers do not run in 'privileged' mode.
- Make sure developers can only create internal load balancers.
- Make sure applications cannot use expensive SSDs.
- Make sure configurations have the proper labels attached.
- Used Container Registry must be in allowed list, etc.
- Verify container image signature before allowing it to be deployed.

The OPA admission controller serves as a gatekeeper within managed Kubernetes services. Gatekeeper, or the "General Purpose Policy Engine", is a first-class Kubernetes citizen. It evaluates incoming requests based on Rego policies administrator define and answers the API server what to do with these requests (deny/allow). In practice it can deny requests made to the API server if these requests are not compliant with the policies we define. Gatekeeper is a native Kubernetes initiative and can be easily installed into any managed Kubernetes.

5.2.7 What security gap still exists after the second C?

This section discussed some of the approaches and techniques to secure the managed Kubernetes cluster and its internal components. Although both the network (cloud) and cluster have been secured, there remain vulnerabilities within containers that can still be exploited by attackers. So, let's discuss some of the approaches to secure the container and container images in upcoming sections.

5.3 Securing Containers Boundary for Managed Kubernetes

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled, and each Pod gets dedicated IP address.

Following are the threats if Pod or Container compromised:
- Get Access to the Host/Nodes by break out Container or Kernal vulnerabilities.
- Attack Api Server and Kubelet
- Attack other containers running on the same node.
- Run malicious code inside container.

The security measures applied to the first two C's (Cloud and Cluster) provide protection for containers against unauthorized access within the Kubernetes environment. However, these measures do not address the risk of malicious code affecting pods or containers. Additionally, the inherited security vulnerability of hypervisor opens a door to Nodes and Pods from host OS. This underlying vulnerability occurs in both Windows and Linux platforms, which grants code from host to access guest VMs through hypervisor. A common mitigation to this fundamental security risk is through Confidential Virtual Machines (CVMs) and Confidential Containers.

5.3.1 Confidential Virtual Machines (CVMs)

Confidential VMs leverage hardware-based Trusted Execution Environments (TEEs), such as AMD SEV-SNP, Intel TDX, or Intel SGX, to encrypt memory and isolate the virtual machine at the hypervisor level. This ensures that data in use is protected not just from external attackers but also from cloud administrators and other workloads running on the same physical host. In a Kubernetes context, CVMs can be used as secure worker nodes where the kubelet and container runtime operate entirely inside a TEE. This protects both the workload and the node-level processes from being tampered with by anyone without the encryption keys that are generated and stored in hardware.

Most major cloud providers support CVMs:
- AKS – Azure Confidential VMs (AMD SEV-SNP, Intel TDX)
- EKS – AWS Nitro Enclaves / EC2 Confidential Computing instances
- GKE – Confidential VMs powered by AMD SEV-SNP

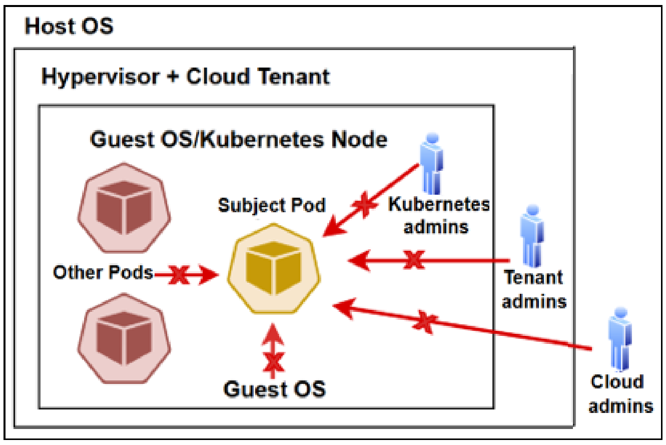
5.3.2 Confidential Containers

Confidential containers take the notion of confidential computing from the node level down to the workload level on an individual container basis, running containers in hardware-backed enclaves so that both the application code and in-memory data are encrypted and verifiable at runtime. This means that threats from a host kernel, Confidential VM, container runtime, or even cluster administrator have been protected.

Confidential Containers (CoCo) based on Kata Containers integrated well with Kubernetes, and similar projects, provide a consistent way to execute Open Container Initiative (OCI) containers inside Trusted Execution Environments (TEEs). Confidential containers add a level of security not only because of the TEE but also because of underlying hardware that reduces the attack surface and provides protection from compromised kernel, compromised Confidential Virtual Machine (CVM) and container runtimes. This means that sensitive workloads, such as those which process financial transactions, supply chain services, conduct training on proprietary AI models, or involve regulated health data, may be operationally run in untrusted cloud environments. Therefore, it is essential to operate mission-critical services as Confidential Containers. Azure Kubernetes Service (AKS) also supports the integration of Azure Container Instances (ACI) based confidential containers through Virtual Node.

5.3.3 What security gap still exists after the third C?

With Confidential virtual machines and containers as a security measure for the third C (Container), our security posture has improved as described in the diagram below:



As illustrated, the security vulnerability exposed to actors as well as code from hypervisor, host/guest agents, peer pods/containers are all mitigated. The remaining security vulnerability is within the code running inside the subject container. Despite TCB to protect the subject pod, the code running inside can still present security risks (such as running downloaded code or starting new processes). The final vulnerability is addressed by the subject of securing the last C, the Code.

5.4 Securing Code Boundary for Managed Kubernetes

As discussed, Confidential Container secures the boundary of container by providing a TCB for application code to run inside a Trusted Execution Environment, but it does not control the runtime behavior of the code running inside the container. Windows implements code integrity to control runtime code behavior, Linux has a different solution that provides a runtime code security as secure as Code Integrity in Windows. The below sections discuss security controls at various levels to secure the last C, the Code.

5.4.1 Securing Container Images

Container images are the foundation of all workloads that run in Kubernetes. If a container image is compromised, it is a potential gateway for injecting vulnerabilities, malware, or backdoors into the cluster. Container images should be secured through integrity and authenticity validation and compliance validations before deploying to a Kubernetes cluster. This will include trusted base images, scanning for known vulnerabilities, enforcing signing & verification requirements, and securely storing images in registries. Security controls need to be in place throughout the image build and delivery pipeline, so unverified or malicious code cannot be deployed to production in a Kubernetes cluster.

5.4.1.1 Container Image scanning at Build time

The first security control is to ensure the container image is free of malware. Containers with outdated base images or unpatched application runtimes introduce security risks and possible attack vectors. We can minimize these risks by scanning containers at build time. The following are the best practices to ensure container images are not vulnerable:

a) 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.
b) Scan your container images for vulnerabilities.
c) Regularly update the base images and application runtime.
d) Regularly deploy updated containers.

5.4.1.2 Container Image signing and verification

Container images are then signed in CI build pipeline using Signing task. It will sign the container image artifacts and attach it to the Image and upload it to a private registry that is secured as described in securing the Cloud section. Verifying container image integrity takes place at various points in container's life cycle: deployment time, startup time and run time. At the deployment time, Custom Admission Controller verifies the Container image signature (validate the entire chain) before it allows the container to be deployed in managed Kubernetes cluster. This will protect running unsigned or wrongly signed container images into Kubernetes cluster.

5.4.1.3 Container Image Re-scanning

Regularly and consistently scanning container images is critical to discover new vulnerabilities that can exist after an image is built and then deployed. This applies to images that are stored in a container registry (e. g., Azure Container Registry, Amazon Elastic Container Registry, Google Artifact Registry) and those images running in the workloads of Kubernetes clusters. Continuous re-scanning keeps images aligned against the most up-to-date vulnerability databases so that security drift does not occur over time. Cloud providers also have built-in security solutions to achieve this (for example, Microsoft Defender for Containers for AKS, Amazon Inspector for EKS, Google Cloud Security Command Center for GKE) to observe and protect containerized assets (Kubernetes clusters, nodes, workloads, registries, etc.).

5.4.2 Securing Container at runtime

Securing the runtime means making the container execution environment minimal, immutable, and constrained so that even if an attacker gains foothold, they cannot escape, persist, or move laterally.

5.4.2.1 Immutable Containers

Immutable containers maintain their original state after deployment because they cannot be altered which protects against unauthorized changes and maintains a stable runtime environment. The process of updating requires developers to build and redeploy a fresh image which strengthens security measures for all three platforms including AKS, EKS and GKE. Containers can be configured as immutable using Security Context of security policy for Confidential Containers. Immutability is important as it denies any alternation of container images after their deployment. Remote access to the container can be blocked by either the security context of the pod configuration or by the security policy of the confidential containers (the later is the preferred approach as it does not reply on the Kubernetes cluster from being malicious). No one can "shell" into containers and make any changes to the container, including running any shell command that might introduce changes to the container after its deployment. Immutability guarantees that container image remains the same throughout its lifetime.

Note, sometimes, your application needs files access, in those scenarios use Kubernetes emptyDir. When we use emptyDir as volume, Kubernetes will attach a local folder from underlying worker node, which lives as long as the pod. Changes are limited to what is on the emptyDir as a local file share. However, the emptydir lives inside the UVM and takes space away from the root file system of the containers. Once emptydir hosted on the node filesystem is enabled, as of now, Confidential Containers will not encrypt the contents, so anything you write there will be visible outside of your TEE.

5.4.2.2 Controlling container runtime behavior using Seccomp

Seccomp is a Linux kernel feature that can be used to restrict the system calls that a process can make. In Kubernetes, Seccomp can be used to secure containers by defining profiles that restrict the system calls that a container can make. Seccomp profiles can be defined for individual containers or for entire namespaces. Using the security profile configured for individual containers or inherited from the namespaces, Seccomp ensures the container can only make the system calls allowed by the profile. The default Docker Seccomp profile blocks 44 syscalls, including reboot, mount, unmount among others. In our implementation, we will add fork and vfork syscalls into this list so the container cannot create any new process. With this security policy, no code inside container can download code from network at the runtime and run downloaded code. Combining immutable containers with Seccomp security profile, the application container runtime behavior is completely controllable.

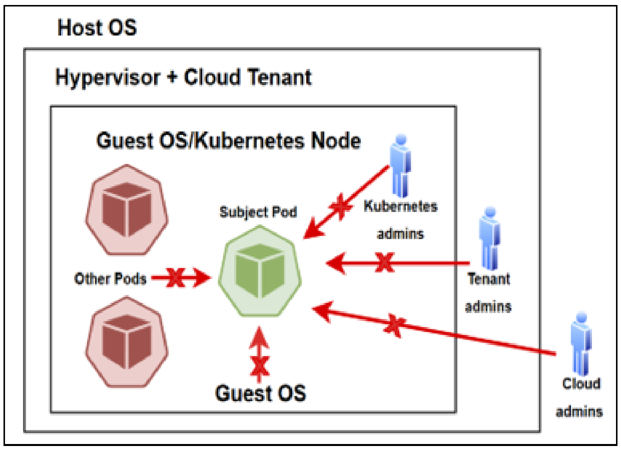
5.4.2.3 Real Time Security Monitoring at runtime using Falco

Falco is a cloud-native security tool designed for Linux systems. It employs custom rules on kernel events, which are enriched with containers and Kubernetes metadata, to provide real-time alerts. Falco helps you gain visibility into abnormal behavior, potential security threats, and compliance violations. With this visibility, Falco allows you to respond to security threats using actions. Sysdig built Falco as an Apache 2.0 Cloud Native Computing Foundation (CNCF) project. It provides a continuous runtime security monitoring system on Linux operating systems that tackle:

- Zero-day vulnerabilities
- Privilege escalation attempts.
- Bugs that cause erratic behavior or resource leaking
- Unexpected behavior in the deployed artifacts

5.4.3 What security gap still exists after the last C?

With a combination of immutable containers and Seccomp security policies as a security measure for the fourth and the last C (Code), our security posture has improved as described in the diagram below:



As illustrated, the vulnerability of subject pod can be completely mitigated in the proposed security architecture.

6. Conclusion

In summary, this paper has advocated for a defense-in-depth security architecture design for managed Kubernetes environments running mission-critical services across industries such as finance, healthcare, defense and supply chain (Signing, Scanning, Packaging, etc.). By securing the Cloud, Cluster, Container, and Code layers, this architecture provides multiple overlapping lines of defense that will minimize insider threats, supply chain compromises and runtime exploits. The features of this architecture rely heavily on security tooling provided by the cloud provider. These cloud security features include Hub-Spoke network architecture for network segmentation and isolation, Confidential Containers to protect sensitive workloads even from cloud infrastructure operations, a comprehensive set of cluster security controls, multiple admission controllers, Role-Based Access Control (RBAC), seccomp profiles, runtime detection, and secure image registries to enforce strict security policy and harden clusters.

More importantly, this security exists as part of the developer workflows and is built in a way that does not affect their productivity or operational flexibility while significantly lifting the organizations overall security posture. The design is also able to protect against more sophisticated attacks by isolating workloads critical to the organization's security and minimizing the attack surface against privileged access from hypervisor or VM or host compromises, cloud administrator breaches, and malicious insider activities from admin accounts.

The examples provided in this work demonstrate that layered security can be applied at all levels and in a scalable and flexible manner to managed Kubernetes deployments. This work offers organizations valuable lessons on how to implement security for cloud-native supply chain activities with Kubernetes as the foundation while still addressing an increasingly complex and evolving threat landscape.

References

[1] Kubernetes, Cluster Architecture, 2024. [Online]. Available: https: //kubernetes. io/docs/concepts/architecture/
[2] CNCF, Cloud Native Security Whitepaper, Cloud Native Computing Foundation, 2023. [Online]. Available: https: //www.cncf. io/reports/cloud-native-security-whitepaper/
[3] Sandeep Kampa, "NAVIGATING THE LANDSCAPE OF KUBERNETESECURITY THREATS AND CHALLENGES, " Journal of Knowledge Learning and Science Technology, 3 (4), 2024, https: //doi. org/10.60087/jklst. v3. n4. p274
[4] Santosh Pai 1, & Srinivasa. R. Kunte, "Secret Management in Managed Kubernetes Services, "International Journal of Case Studies in Business, IT, and Education 7 (2), 2023, http: //dx. doi. org/10.47992/IJCSBE.2581.6942.0263
[5] Luca Passaretta, Hub & Spoke Architecture and Landing Zones on Azure Cloud, 2023. [Online]. Available: https: //medium. com/[at]lupass93/hub-spoke-architecture-and-landing-zones-on-azure-cloud-ad2e1b11c55
[6] Microsoft, Hub-spoke network topology in Azure. [Online]. Available: https: //learn. microsoft. com/en-us/azure/architecture/networking/architecture/hub-spoke
[7] Microsoft, Baseline architecture for an Azure Kubernetes Service (AKS) cluster. [Online]. Available: https: //learn. microsoft. com/en-us/azure/architecture/reference-architectures/containers/aks/baseline-aks
[8] Microsoft, Security concepts for applications and clusters in Azure Kubernetes Service (AKS). [Online]. Available: https: //docs. azure. cn/en-us/aks/concepts-security
[9] Google Cloud, Hub-and-spoke network architecture, 2025. [Online]. Available: https: //cloud. google. com/architecture/deploy-hub-spoke-vpc-network-topology
[10] AWS, Multi-Cluster centralized hub-spoke topology. [Online]. Available: https: //aws-ia. github. io/terraform-aws-eks-blueprints/patterns/gitops/gitops-multi-cluster-hub-spoke-argocd/
[11] Microsoft, Confidential Containers (preview) with Azure Kubernetes Service (AKS), 2025. [Online]. Available: https: //learn. microsoft. com/en-us/azure/aks/confidential-containers-overview

[12] Microsoft, Create and configure an Azure Kubernetes Services (AKS) cluster to use virtual nodes using Azure CLI, 2025. [Online]. Available: https: //docs. azure. cn/en-us/aks/virtual-nodes-cli
[13] Confidential Containers, Overview. [Online]. Available: https: //confidentialcontainers. org/docs/overview/
[14] Tigera, Cloud-Native Security: 4 C's and 5 Strategies. [Online]. Available: https: //www.tigera. io/learn/guides/cloud-native-security/#The-Four-Cs-of-Cloud-Native-Security
[15] David Mosyan, The 4C's of Cloud Native Kubernetes security, 2023. [Online]. Available: https: //medium. com/[at]dmosyan/the-4cs-of-cloud-native-kubernetes-security-958c720e2391
[16] Crowdstrike, The Fundamentals of Kubernetes Security, 2024. [Online]. Available: https: //www.crowdstrike. com/en-us/cybersecurity-101/cloud-security/kubernetes-security/
[17] Asim Mirza, Kubernetes Security Contexts Series-Part 4: Immutable Filesystem, 2025. [Online]. https: //medium. com/[at]mughal. asim/kubernetes-security-contexts-series-part-4-immutable-filesystem-b3d7e5d0be5c
[18] RX-M, CKS Mod 6: Ensure immutability of containers at runtime. [Online]. Available: https: //rx-m. com/lesson/cks-ensure-immutability-of-containers-at-runtime/
[19] Marco Lenzo, Immutable Kubernetes Pods, 2022. [Online]. Available: https: //marcolenzo. eu/create-immutable-kubernetes-pods-with-the-security-context/
[20] Falco, Detect security threats in real time. [Online]. Available: https: //falco. org/