KUBERMATIC

Kubermatic Kubernetes Platform PCI DSS 4.0 Compliance Architecture and Design Guide

Published June 2026 (aligned to PCI DSS v4.0.1 and KKP v2.30)

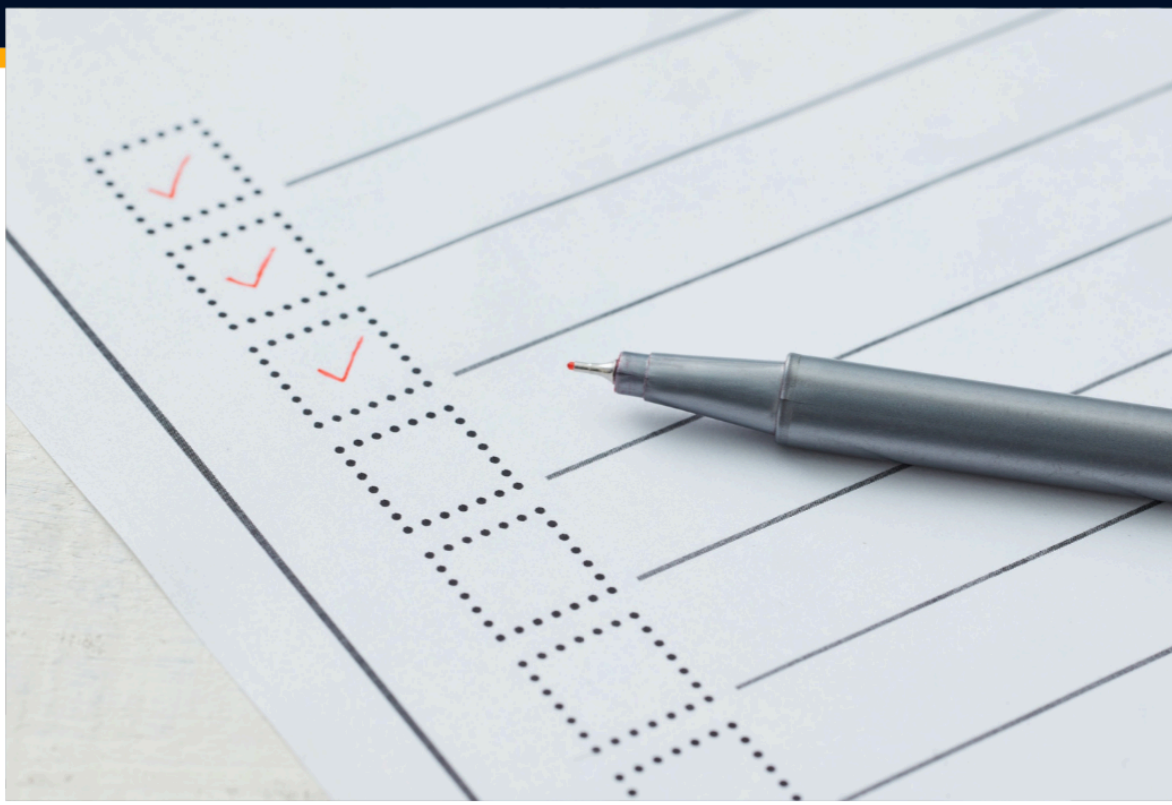

Table of Contents

1. Introduction

1.1. Purpose and Scope of the Guide

This guide provides an architectural and design framework for organizations using Kubermatic Kubernetes Platform (KKP) to achieve and maintain compliance with the Payment Card Industry Data Security Standard (PCI DSS). It is written against **PCI DSS v4.0.1** — the limited revision published in June 2024 that introduced no new or removed requirements relative to v4.0, only clarifications and corrections. As of 2026 the v4.0 transition period has ended: all requirements, including the formerly future-dated ones that became mandatory on 31 March 2025, are in full effect, and v4.0.1 is the baseline for assessments. This document serves as a practical resource for building, configuring, and managing a secure Cardholder Data Environment (CDE) on KKP. It covers the KKP architecture, maps KKP features and Kubernetes-native controls to PCI DSS 4.0 requirements, offers detailed hardening guidelines, outlines security operations best practices, discusses third-party integrations, and provides strategies for continuous compliance.

The scope encompasses KKP deployments across various infrastructures, including on-premises, multi-cloud, and edge environments. It addresses the security of the KKP management plane (Master and Seed Clusters) and the User Clusters where CDE workloads reside. This guide is intended for security architects, Kubernetes administrators, compliance officers, and DevOps teams responsible for PCI DSS compliance within KKP environments.[1]

1.2. Target Audience

This document is designed for technical and compliance professionals involved in designing, implementing, managing, and auditing PCI DSS compliant environments using Kubermatic Kubernetes Platform. This includes:

- Security Architects and Engineers
- Kubernetes Platform Administrators and Operators
- DevOps and DevSecOps Teams
- Compliance Officers and Auditors
- IT Management responsible for payment security

A foundational understanding of Kubernetes concepts, KKP architecture, and PCI DSS principles is assumed.[3]

1.3. Understanding PCI DSS 4.0 and v4.0.1

PCI DSS 4.0, released in March 2022, became fully effective on March 31, 2024, with all future-dated requirements becoming mandatory by March 31, 2025.5 **PCI DSS v4.0.1**, a limited revision released in June 2024, is the current active version; it introduced no new or deleted requirements (only clarifications and error corrections), so the requirement numbers used throughout this guide are identical in v4.0 and v4.0.1. This version introduces significant changes

to address the evolving threat environment and modern technologies like cloud computing and containerization.[5] Key objectives of PCI DSS 4.0 include promoting security as a continuous process, adding flexibility through a new "customized approach," and enhancing validation methods.[5]

Key Changes in PCI DSS 4.0 Relevant to KKP Environments:

- **Customized Approach:** Allows organizations to meet security objectives using innovative controls tailored to their specific environment, provided they conduct a targeted risk analysis and document how their custom controls meet the intent of the PCI DSS requirement.[5] This is particularly relevant for complex, dynamic Kubernetes environments.
- **Expanded Multi-Factor Authentication (MFA):** MFA is now required for *all* access into the CDE — an expansion from the prior remote-access-only scope. This includes access to KKP management interfaces, Kubernetes APIs, and any system component within the CDE.[6]
- **Updated Password and Authentication Requirements:** Minimum password length increased to 12 characters (unless technically infeasible), and stricter controls around password management and shared/group accounts.[5]
- **Automated Technical Solutions for Web Applications:** Public-facing web applications must be protected by an automated technical solution that continually detects and prevents web-based attacks (e.g., a Web Application Firewall - WAF). This is mandatory under Requirement 6.4.2.6
- **Management of Payment Page Scripts:** New requirements (6.4.3, 11.6.1) mandate inventory, authorization, integrity verification, and monitoring of all scripts loaded and executed on payment pages to prevent e-skimming attacks.[11] This applies if applications on KKP serve payment pages.
- **Authenticated Internal Vulnerability Scans:** Requirement 11.3.1.2 mandates authenticated internal vulnerability scans at least every three months, providing deeper insights than unauthenticated scans.[16]
- **Targeted Risk Analyses:** Required for several controls, allowing organizations to define frequencies or methods based on their specific risks.[5]
- **Enhanced Focus on Third-Party Service Provider (TPSP) Security:** Increased responsibility for organizations to monitor and ensure the PCI DSS compliance of their TPSPs, including cloud providers and managed service providers.[4]

Understanding these changes is essential for designing a PCI DSS 4.0 compliant architecture on KKP. This guide will address how KKP features and recommended practices align with these updated requirements.

2. Kubermatic Kubernetes Platform (KKP) Overview for PCI DSS

2.1. KKP Architecture

Kubermatic Kubernetes Platform (KKP) is a Kubernetes management platform designed to automate the deployment and operations of potentially thousands of Kubernetes clusters across hybrid-cloud, multi-cloud, and edge environments.[1] Its architecture is inherently multi-cluster and consists of distinct layers:

- **Master Cluster:** This is a central Kubernetes cluster that hosts the KKP master components. It is responsible for storing global information such as user definitions, projects, and SSH keys. The KKP API and UI Dashboard are served from the Master Cluster.[19] For PCI DSS, the Master Cluster itself must be secured rigorously as it manages access and configurations for downstream clusters that might be part of the CDE.
- **Seed Cluster(s):** A Seed Cluster is a Kubernetes cluster responsible for hosting the control plane components (kube-apiserver, etcd, kube-scheduler, kube-controller-manager) of one or more User Clusters.[1] This architectural choice isolates User Cluster control planes, enhancing resilience and security. For PCI DSS, Seed Clusters managing CDE User Clusters are critical components within the CDE scope and require stringent security controls. KKP allows a Master Cluster to also function as a Seed Cluster in smaller setups, though this is not recommended for large-scale or highly sensitive PCI deployments due to the increased blast radius.[19]
- **User Cluster(s):** These are the Kubernetes clusters created and managed by KKP where end-user applications, including those processing or storing cardholder data, are deployed.[1] User Clusters designated as part of the CDE are subject to all applicable PCI DSS requirements. KKP supports creating User Clusters on a wide array of infrastructure providers.

KKP uses Kubernetes Operators for multi-cluster management, automating the creation and full lifecycle management of clusters.[1] This automation is a key feature for maintaining consistent configurations and applying security updates, which are vital for PCI DSS compliance.

2.2. Deployment Models and Infrastructure Options

KKP offers flexibility in deployment models, supporting:

- **On-Premises:** Deployments on bare metal, VMware vSphere, OpenStack, Nutanix, and other virtualized infrastructures.[2] In this model, the organization is responsible for the physical security and underlying infrastructure security.
- **Cloud Providers:** Support for major public clouds like AWS, Google Cloud Platform (GCP), Microsoft Azure, and over 20 others.[2] KKP can manage native Kubernetes services (EKS, GKE, AKS) or deploy clusters on cloud IaaS.[21] The shared responsibility model with the Cloud Service Provider (CSP) is critical here.

- **Edge Environments:** KKP is designed to manage Kubernetes clusters at the edge, which may have resource constraints.[2]

The choice of infrastructure provider directly impacts the scope of PCI DSS responsibilities. When using cloud providers, organizations inherit some controls (e.g., physical security of data centers) but remain responsible for securing their configurations, data, and applications *in* the cloud.[22]

2.3. Key KKP Features Relevant to PCI DSS Compliance

KKP provides several features that can be used to meet PCI DSS 4.0 requirements:

- **Centralized User Management and OIDC Integration:** KKP allows management of users from a single panel and supports integration with external OpenID Connect (OIDC) providers.[1] This is essential for PCI DSS Req 8 (Identify users and authenticate access), enabling unique user IDs and facilitating MFA integration.
- **Monitoring, Logging, and Alerting (MLA) Stack:** KKP provides a two-tiered MLA stack. The Master/Seed Cluster MLA stack monitors management components, accessible only by KKP Admins. The User Cluster MLA stack offers multi-tenancy for end-users and a consolidated overview for Admins, built using Prometheus, Loki, Cortex, and Grafana (in KKP v2.30 the User Cluster MLA collection agent moved to Grafana Alloy; the backend set is unchanged).[1] This supports PCI DSS Req 10 (Log and monitor all access).
- **Cluster Lifecycle Management and Automation:** KKP automates the creation, scaling, updating, and full lifecycle management of Kubernetes clusters.[1] This aids in maintaining secure configurations (Req 2) and timely patching (Req 6).
- **OPA Gatekeeper Integration:** KKP integrates with OPA Gatekeeper for policy enforcement, allowing administrators to define and apply custom policies to clusters.[1] This is vital for enforcing security configurations (Req 2) and access controls (Req 7).
- **Network Policy Support:** While KKP itself doesn't enforce network policies directly at the User Cluster level (this is typically handled by the CNI), its architecture supports the use of CNIs like Cilium and Canal which enable Kubernetes NetworkPolicies.[25] These are fundamental for network segmentation within the CDE (Req 1).
- **API Server Access Control:** KKP allows restricting access to the User Cluster API server based on source IP ranges and provides API Server Network Policies for egress traffic control from the API server pods in the Seed cluster.[27] This supports Req 1 and Req 7.
- **Automated Backups:** KKP provides etcd backup and restore for User Clusters to S3-compatible storage in the Community Edition, plus an integrated Velero-based cluster backup of workloads, namespaces, and persistent-volume contents in the Enterprise Edition (introduced in v2.25). Backups can be restored to different KKP instances, supporting disaster recovery and data availability (relevant to business continuity aspects of PCI DSS).[28]
- **Secrets Management:** While KKP doesn't provide a dedicated secrets management vault, it manages Kubernetes Secrets which, when combined with etcd encryption, provides a baseline

for secrets protection. Integration with external vaults is recommended for enhanced security (Req 3).

- **Multi-Tenancy and Cluster Separation:** KKP's architecture inherently supports multi-tenancy and separation of clusters, which can be used to isolate CDE environments from non-CDE environments, helping to reduce PCI DSS scope.[21]

These features, when configured and managed correctly, provide a strong foundation for building a PCI DSS compliant environment on KKP.

2.4. Shared Responsibility Model for PCI DSS on KKP

Achieving PCI DSS compliance in a KKP environment is a shared responsibility between Kubermatic (for certain aspects of the KKP software and managed services, if applicable), the organization deploying KKP (the customer), and potentially the underlying infrastructure provider (e.g., CSP for cloud-hosted deployments or the organization itself for on-prem data centers).

- **Kubermatic's Responsibilities (General):**
  - Providing KKP software that is capable of being configured securely.
  - Ensuring the security *of* the KKP software itself (e.g., patching vulnerabilities in KKP components).
  - If using Managed KKP, Kubermatic takes on more operational responsibilities for the KKP platform as defined in the service agreement.[29] Kubermatic is ISO 27001 certified, which indicates a commitment to information security management.[2] Their Data Processing Agreement (DPA) outlines technical and organizational measures for data protection.[30]
- **Customer's Responsibilities:**
  - **Scope Definition:** Clearly defining the CDE within the KKP environment.
  - **Secure Configuration:** Securely configuring KKP Master, Seed, and User Clusters, including all Kubernetes components (API server, etcd, kubelets, etc.), network policies, RBAC, OIDC integration, logging, and monitoring according to PCI DSS requirements.[22]
  - **Infrastructure Security:**
    - **On-Premises:** Securing the physical data center, underlying hardware, and virtualization layers.
    - **Cloud:** Managing "security *in* the cloud, " which includes configuring cloud provider services (VPCs, security groups, IAM, storage encryption) securely, and understanding the CSP's responsibilities for "security *of* the cloud".[22]
  - **Workload Security:** Securing containerized applications deployed within User Clusters, including image security, application code security, and data handling practices.
  - **Data Security:** Implementing encryption for data at rest (CHD in persistent volumes, etcd) and data in transit (TLS/mTLS). Managing cryptographic keys securely.

    - **Vulnerability Management:** Scanning for and remediating vulnerabilities in KKP components (if self-hosted), operating systems, container images, and applications.
    - **Access Control:** Implementing and managing strong authentication (including MFA) and authorization (RBAC, OIDC) for all access to the CDE and KKP components.
    - **Monitoring and Logging:** Configuring and managing thorough logging and monitoring for all CDE components, reviewing logs, and responding to alerts.
    - **Incident Response:** Developing and maintaining an incident response plan for CDE breaches.
    - **Compliance Validation:** Regularly assessing and validating PCI DSS compliance, including engaging with a Qualified Security Assessor (QSA) if required.
- **Infrastructure Provider Responsibilities (e.g., AWS, Azure, GCP):**
    - Security *of* the cloud: Physical security of data centers, security of the underlying network and virtualization infrastructure.[22]
    - Providing PCI DSS compliant infrastructure services that customers can build upon. Customers should review the CSP's Attestation of Compliance (AoC).[3]

A clear understanding and documentation of these responsibilities, often formalized in a Responsibility Matrix (especially with TPSPs), are essential for successful PCI DSS compliance.[11] The complexity of KKP's architecture, with its Master, Seed, and User Cluster layers, means that these responsibilities must be carefully mapped to each component and operational aspect. For instance, securing the etcd instances for User Clusters (which run on Seed Clusters) involves both KKP's operational model and the customer's configuration of security features like encryption and access control for those Seed Clusters.

3. Understanding the Cardholder Data Environment (CDE) in KKP

3.1. Defining the CDE Scope in a KKP Environment

The Cardholder Data Environment (CDE) encompasses all people, processes, and technology that store, process, or transmit cardholder data (CHD) or sensitive authentication data (SAD), as well as system components that may not store, process, or transmit CHD/SAD but have unrestricted connectivity to or could impact the security of CDE components.[4] Accurately defining the CDE scope is a critical first step in any PCI DSS compliance effort, as it determines which systems and networks are subject to the full set of PCI DSS requirements. Reducing the scope can significantly decrease the complexity and cost of compliance.[33]

In a KKP environment, the CDE could include:

- **Specific User Clusters:** User Clusters designated to host applications that handle CHD.

- **Namespaces within User Clusters:** If a User Cluster hosts both CDE and non-CDE applications, specific namespaces might be designated as in-scope.
- **Worker Nodes:** The worker nodes (VMs or bare metal servers) hosting the pods within the in-scope User Clusters or namespaces.
- **Seed Cluster(s):** Any Seed Cluster that hosts the control plane components (API server, etcd, etc.) for in-scope User Clusters. These are critical as they manage the CDE workloads.
- **Master Cluster Components (Potentially):** If the Master Cluster directly manages or has privileged access to Seed or User Clusters within the CDE, certain components or access pathways of the Master Cluster could be considered in scope or connected-to systems.
- **Persistent Storage:** Any persistent volumes (PVs) and underlying storage systems used by CDE applications to store CHD.
- **Networking Components:**
  - CNI plugins (e.g., Cilium, Canal) and their configurations within CDE User Clusters.
  - Ingress controllers, load balancers, and WAFs that route traffic to CDE applications.
  - Service mesh components (e.g., Istio) if used for traffic management, security, or observability within the CDE.
  - Underlying physical or virtual network infrastructure connecting CDE components.
- **Logging, Monitoring, and Alerting (MLA) Systems:** KKP's User Cluster MLA stack components (Prometheus, Loki, Cortex, Grafana) if they collect, process, or store logs/metrics containing CHD or from CDE systems. Also, any centralized SIEM or log management system receiving data from the CDE.
- **Backup Systems:** Systems and storage locations used for backing up CDE User Clusters or CHD (e.g., KKP's Velero integration and its storage destinations).
- **CI/CD Pipelines:** Components of CI/CD pipelines that build, test, and deploy applications or configurations into the CDE.
- **Management Tools and Interfaces:** KKP UI/API, kubectl access points, and any other tools used to manage or access CDE components.

3.2. Strategies for CDE Segmentation and Isolation in KKP

Network segmentation is a key strategy to isolate the CDE from the rest of the organization's network, thereby reducing the scope of the PCI DSS assessment.[33] If segmentation is performed effectively and validated, systems outside the CDE may not be subject to the full PCI DSS audit.

KKP Architectural Advantages for Segmentation:

- **Dedicated User Clusters for CDE:** The most straightforward approach is to deploy all CDE workloads in one or more dedicated User Clusters. These clusters can then be isolated at the

infrastructure level (e.g., separate VPCs/VNETs in the cloud, dedicated physical hardware on-prem) and through KKP configurations.

- **Seed Cluster Isolation:** Seed Clusters managing CDE User Clusters can also be deployed in isolated network segments.
- **Multi-Tenancy:** KKP's multi-tenancy features allow for logical separation of projects and users, which can align with CDE boundaries.[21]

Technical Segmentation Mechanisms in KKP and Kubernetes:

1. **Infrastructure-Level Segmentation:**
    - **Cloud:** Deploy CDE User Clusters and their associated Seed Clusters in dedicated Virtual Private Clouds (VPCs) or Virtual Networks (VNETs) with strict firewall rules (Security Groups, NSGs) controlling inbound and outbound traffic.[38]
    - **On-Premises:** Use VLANs, physical firewalls, and separate physical hardware to isolate CDE infrastructure.
2. **Kubernetes NetworkPolicies:**
    - Implement default-deny ingress and egress policies for all namespaces within CDE User Clusters.[3]
    - Explicitly allow only necessary traffic flows between pods, to and from specific services, and to external systems (e.g., payment gateways) based on labels, IP CIDRs, and ports.[3]
    - CNIs like Cilium (supported by KKP) offer advanced NetworkPolicy capabilities, including identity-based and DNS-aware policies, and can enforce policies using eBPF for greater efficiency and granularity.[25]
3. **Service Mesh (e.g., Istio):**
    - Provides microsegmentation at the service level, enforcing which services can communicate with each other using AuthorizationPolicies.[33]
    - Enforces mTLS for all service-to-service communication within the CDE, ensuring encrypted and authenticated traffic flows.[33]
4. **KKP API Server Access Controls:**
    - Use KKP's feature to restrict User Cluster API server access to specific IP ranges.[27]
    - Use KKP API Server Network Policies to control egress traffic from API server pods in the Seed cluster.[27]
5. **Role-Based Access Control (RBAC):**
    - Implement strict RBAC policies within Kubernetes to limit access to CDE resources (namespaces, pods, secrets) based on the principle of least privilege.[3]
    - KKP's OIDC integration facilitates mapping enterprise identities to Kubernetes roles.[1]
6. **Namespace Isolation:**

- Use Kubernetes namespaces to logically separate CDE workloads from non-CDE workloads within the same User Cluster, if dedicated clusters are not feasible.[47] This must be coupled with strict NetworkPolicies and RBAC.

Validation of Segmentation:

PCI DSS requires that segmentation controls are tested and validated regularly (at least annually and after significant changes) to ensure they are effective.[34] This typically involves penetration testing and specific segmentation checks.

The ability of KKP to manage numerous, potentially smaller, User Clusters facilitates the creation of dedicated CDE clusters. This architectural pattern is generally preferable for PCI DSS, as it provides stronger isolation boundaries compared to relying solely on namespace-level segmentation within a shared cluster. Furthermore, KKP's management of User Cluster control planes within Seed Clusters means that the security and isolation of these Seed Clusters are paramount when they manage CDE User Clusters. If a Seed Cluster manages both CDE and non-CDE User Clusters, careful consideration must be given to the potential for compromise and the scope implications.

4. Mapping KKP Features and Kubernetes Controls to PCI DSS 4.0 Requirements

This section provides a mapping of Kubermatic Kubernetes Platform (KKP) features, native Kubernetes controls, and recommended third-party tool integrations to the twelve primary requirements of PCI DSS 4.0. This mapping aims to guide organizations in using KKP and its ecosystem to build and maintain a compliant CDE. The "Customized Approach Applicability" column indicates where PCI DSS 4.0's customized approach might be particularly relevant for Kubernetes environments.

Table 1: Mapping PCI DSS 4.0 Requirements to KKP Features and Controls

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
Goal 1: Build and Maintain a Secure Network and Systems						
Req 1: Install and Maintain Network Security Controls	Processes and mechanisms for installing and maintaining network security controls (NSCs) are defined and understood.	- KKP supports CNIs enabling NetworkPolicies. - API Server Allowed IP Ranges.[27] - API Server Egress	- Network Policies (default deny, allow specific flows).[3] - Ingress/ Egress controllers. - Service definitions.	- **CNI Plugins (Cilium, Canal/Calico):** Advanced policy enforcement, eBPF capabilities (Cilium).[25] - **Service Mesh (Istio, Linkerd):** Microsegmentation, mTLS.[33]	- Define CDE boundaries within KKP (dedicated User Clusters/Seed Clusters). - Implement strict NetworkPolicies for CDE namespaces. - Configure KKP API server access restrictions.	Yes, for complex software-defined networking and microsegmentation strategies.

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
	NSCs are configured and maintained. Network access to and from the CDE is restricted. 4	Network Policies.[27] - Cluster templating for consistent network configuration s.[1]		- **WAFs:** For ingress protection (mandatory for public web apps per 6.4.2).[6] - **External Firewalls/Cloud NSGs/Security Groups.**	- Document all NSC configurations and justifications. - Regularly review and test NSC rules.	
Req 2: Apply Secure Configurations to All System Components	System components are configured and maintained securely. Vendor-supplied defaults are changed. Wireless environments are configured securely. 51	- KKP automates cluster deployment and lifecycle management, facilitating consistent configurations.[1] - OPA Gatekeeper integration for policy enforcement.[1] - Cluster templating.[1]	- Pod Security Admission (PSA).[53] - RBAC for least privilege.[3] - Hardening configurations for API server, kubelet, etcd. - Secure container runtime configurations.	- **Policy Engines (OPA/Gatekeeper, Kyverno):** Enforce custom configuration policies.[55] - **Configuration Management Tools (Ansible, Chef, Puppet):** For OS and node hardening. - **Vulnerability Scanners (Qualys, Tenable, Aqua Security, Sysdig):** To verify secure configurations against benchmarks (CIS).[42]	- Harden KKP Master, Seed, and User Cluster components. - Harden worker node OS (CIS Benchmarks). - Use PSA (Restricted profile for CDE) and/or OPA/Gatekeeper to enforce secure pod configurations. - Change all default credentials. - Maintain an inventory of system components in the CDE.	Yes, for demonstrating how dynamic Kubernetes configurations meet security objectives if prescriptive controls are challenging.
Goal 2: Protect Account Data						
Req 3: Protect Stored Account Data	PAN is unreadable wherever it is stored. Storage of sensitive authentication data (SAD) after authorization is prohibited. 52	- KKP provides a first-class Data Encryption at Rest feature for User Clusters (Cluster `spec.encryptionConfiguration`, with key rotation; dashboard-toggle since v2.29).[3] - KKP supports infrastructure providers offering encrypted storage.	- Kubernetes Secrets encryption at rest (via API server config).[3] - Encryption of persistent volumes (PVs) using CSI drivers and underlying storage encryption.	- **External Secrets Management (HashiCorp Vault, Cloud KMS-integrated solutions):** For strong secrets and key management.[66] - **Database Encryption Solutions.** - **Tokenization Solutions.** - **Data Discovery & Classification Tools.**	- Enable etcd encryption for all Kubernetes Secrets and sensitive ConfigMaps in CDE clusters. - Encrypt all persistent volumes storing CHD using strong cryptography (e.g., AES-256). - Implement strong key management procedures for encryption keys (Req 3.6, 3.7). - Prohibit storage of SAD post-authorization. - Define and enforce data retention and disposal policies.	Yes, for use of advanced encryption methods (e.g., homomorphic encryption) or tokenization solutions not explicitly listed.

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
Req 4: Protect Cardholder Data with Strong Cryptography During Transmission Over Open, Public Networks	Strong cryptography and security protocols are used to safeguard CHD during transmission over open, public networks. 52	- KKP UI/API access via HTTPS. - KKP facilitates secure communication between its components (Master, Seed, User Clusters) typically over TLS.	- TLS for API server communication.[3] - Ingress controllers with TLS termination. - NetworkPolicies can restrict unencrypted protocols.	- **Service Mesh (Istio, Linkerd):** For automated mTLS between services within the CDE.[33] - **WAFs/Load Balancers:** For TLS termination at the edge with strong cipher suites. - **VPNs/IPsec:** For secure remote access or site-to-site connections.	- Enforce TLS 1.2 or higher for all CHD transmissions. - Use strong, industry-accepted cryptographic algorithms and key lengths. - Implement mTLS for all inter-pod/service communication within the CDE. - Securely manage SSL/TLS certificates and keys. - Protect wireless transmissions if applicable.	No, this requirement is generally prescriptive regarding TLS versions and strong cryptography.
Goal 3: Maintain a Vulnerability Management Program						
Req 5: Protect All Systems and Networks from Malicious Software	Malicious software (malware) is prevented or detected and addressed. Anti-malware solutions are deployed and maintained. 70	- KKP itself does not directly provide anti-malware. Customer deploys on nodes/containers.	- Pod Security Admission/OPA can restrict execution of untrusted code. - Using minimal, hardened base images reduces attack surface.	- **Anti-malware/EDR solutions:** For worker nodes and potentially within containers (if supported by vendor). - **Container Runtime Security Tools (Falco, Aqua Security, Sysdig, Cilium Tetragon):** Can detect malware-like behavior.[3] - **Image Scanners:** To detect malware in images before deployment.[42]	- Deploy anti-malware on all systems commonly affected by malware within the CDE (especially worker nodes). - Keep anti-malware signatures and engines up-to-date. - Perform periodic scans. - For containers, focus on image scanning and runtime behavior monitoring.	Yes, if using non-traditional anti-malware approaches like behavioral detection as primary control.
Req 6: Develop and Maintain Secure Systems and Software	Security vulnerabilities are identified and addressed. Secure software development practices are followed. Changes to system components are managed securely. 14	- KKP automates cluster upgrades and patching for KKP-managed components.[1] - OPA Gatekeeper for enforcing secure deployment policies.[1]	- Secure software development lifecycle (SSDLC) for applications deployed on KKP. - Regular patching of worker node OS and container base images. - Change management processes for CDE configurations.	- **Vulnerability Scanners (Authenticated & Unauthenticated):** For OS, Kubernetes, images, applications.[16] - **Patch Management Tools.** - **SAST/DAST tools:** For application code. - **WAF (Req 6.4.2):** Mandatory for public web apps.[6] - **Client-Side Script Management tools (Req 6.4.3, 11.6.1)**.6	- Establish a process to identify and risk-rank vulnerabilities (CVSS). - Apply security patches promptly. - Implement secure coding practices for CDE applications. - Deploy WAF for public-facing web apps in CDE. - Manage and secure payment page scripts. - Maintain inventory of bespoke/custom software and third-party components.	Yes, for defining alternative controls for managing payment page scripts or for justifying risk-based patch management timelines.

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
Goal 4: Implement Strong Access Control Measures						
Req 7: Restrict Access to System Components and Cardholder Data by Business Need to Know	Access to system components and CHD is limited based on an individual's job responsibilities and need to know. 48	- KKP User Management with project-based access.[1] - OIDC integration for centralized identity.[1]	- Kubernetes RBAC (Roles, ClusterRoles, RoleBindings, ClusterRoleBindings).[3] - Namespaces for resource isolation.	- **Identity and Access Management (IAM) solutions.** - **Policy Engines (OPA/Gatekeeper, Kyverno):** For fine-grained access control policies. - **Service Mesh (Istio):** For service-level authorization policies.[33]	- Implement principle of least privilege for all KKP users, Kubernetes users, and service accounts. - Define access needs for each role. - Default "deny-all" access. - Document and regularly review access controls. - KKP project structure can help align access to business need.	Yes, for complex, attribute-based access control (ABAC) models if they meet the objective.
Req 8: Identify Users and Authenticate Access to System Components	All individuals with access to system components or CHD are uniquely identified and authenticated. MFA is implemented for all access into the CDE. 9	- KKP User Management provides unique user accounts.[1] - OIDC integration supports federated identities and can enforce MFA via the IdP.[1]	- Kubernetes user accounts (via OIDC) and service accounts. - Strong password policies for local accounts (if any).	- **External Identity Providers (IdPs) with MFA capabilities (Okta, Azure AD, Keycloak, etc.).** Thales and F5 offer MFA solutions.[6] - **Hardware tokens, authenticator apps.**	- Assign unique IDs to all users. - Enforce MFA for ALL access to KKP UI/API, kubectl, SSH to nodes, and any CDE component (Req 8.4.2).[6] - Implement strong password/passphrase policies (12 characters min. per Req 8.3.6 by Mar 2025). - Secure all application and system accounts. - KKP OIDC integration is key for MFA.	No, MFA for all CDE access is prescriptive. Customized approach may apply to *how* MFA is implemented for specific systems if standard methods are infeasible.
Req 9: Restrict Physical Access to Cardholder Data	Physical access to CHD is restricted. 52	- Not directly applicable to KKP software, but to the infrastructure it runs on.	- Not directly applicable to Kubernetes software.	- **Physical security controls for data centers (biometrics, CCTV, guards).**	- For on-prem KKP deployments, ensure data centers housing Master/Seed/User Clusters meet PCI DSS physical security requirements. - For cloud deployments, rely on CSP's physical security (review their AoC).[3] - Secure any media containing CHD. - Restrict physical access to KKP console terminals or devices used to access CDE.	No, physical security controls are generally prescriptive.
Goal 5: Regularly Monitor and Test Networks						
Req 10: Log and Monitor All Access to System Components	Logs are generated, protected, and reviewed to detect and respond to	- KKP MLA Stack (Prometheus, Loki, Cortex, Grafana) for Master/Seed	- Kubernetes Audit Logs (API server activity).[3] - Container logs	- **Centralized SIEM (Splunk, Elasticsearch, IBM QRadar, Microsoft Sentinel, Google Security Operations, Datadog,**	- Enable thorough logging for all KKP components, Kubernetes API server (audit logs), OS, applications, and network devices in the CDE.	Yes, for defining risk-based log review frequencies or alternative

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
and Cardholder Data	security incidents. 52	and User Clusters.[1] - Configurable log retention within KKP's MLA.	(stdout/stderr). - Node OS logs (syslog, journald).	**LogRhythm):** For log aggregation, correlation, alerting, and retention.[3] - **FIM tools (Req 10.3.4, 11.5.2):** (e.g., Wazuh, Tripwire, Aqua Security, Cilium Tetragon).[25]	- Ensure logs capture user, event, timestamp, success/failure, etc. - Centralize logs to a secure SIEM. - Retain logs for at least 1 year, with 90 days immediately available.[91] - Implement daily (or more frequent for critical systems) log reviews (automated where possible). - Protect logs from tampering. - Synchronize time across all systems (NTP).	log monitoring mechanisms if they meet the objective (Req 10.4.2.1).
Req 11: Test Security of Systems and Networks Regularly	Security systems and processes are tested regularly to identify and address vulnerabilities. 52	- KKP facilitates deployment of scanning tools as applications.	- N/A directly, but Kubernetes configurations are subject to testing.	- **Vulnerability Scanners (Qualys, Tenable, Nessus, OpenVAS, Aqua, Sysdig):** For internal/external network scans and authenticated scans (Req 11.3.1.2).[16] - **Penetration Testing Tools & Services.** - **IDS/IPS solutions (Req 11.5.1):** (e.g., Snort, Suricata, commercial solutions).[35] - **FIM solutions (Req 11.5.2):**.15 - **Client-Side Script Monitoring tools (Req 11.6.1):**.6	- Conduct quarterly internal and external vulnerability scans (ASV for external). - Perform authenticated internal scans quarterly (Req 11.3.1.2). - Conduct annual (or more frequent) penetration testing on CDE scope and segmentation controls. - Deploy IDS/IPS to monitor traffic at CDE perimeter and critical points. - Deploy FIM for critical system files. - Implement change-detection for payment page scripts/headers.	Yes, for justifying alternative testing frequencies or methods based on targeted risk analysis (e.g., for Req 11.6.1).
Goal 6: Maintain an Information Security Policy						
Req 12: Support Information Security with Organizational Policies and Programs	A thorough information security policy is maintained and supported by programs that address people, processes, and technology. 18	- KKP's features support policy enforcement (e.g., OPA, RBAC via OIDC). - KKP's automated backups support DR/BCP.[28]	- Kubernetes configurations should align with documented security policies.	- **GRC (Governance, Risk, Compliance) platforms.** - **Security Awareness Training platforms.** - **Incident Response Management tools.**	- Develop, publish, and maintain a thorough information security policy. - Implement a formal security awareness program, including training on CHD protection. - Perform targeted risk analyses for applicable requirements.[5] - Manage TPSP risks, including due diligence and documented agreements.	Yes, extensively. The customized approach itself requires targeted risk analysis (part of Req 12.3.1) and thorough documentation of how custom

PCI DSS 4.0 Requirement	Requirement Description (Abbreviated)	KKP Native Feature/ Control	Kubernetes Native Control/ Feature	Recommended Third-Party Tool/Integration	Configuration Guidance & PCI DSS Considerations for KKP	Customized Approach Applicability
					- Maintain an incident response plan (IRP) and test it annually.[3] - Define roles and responsibilities for PCI DSS compliance.	controls meet PCI DSS objectives.

This table highlights the multi-faceted nature of achieving PCI DSS compliance with KKP. While KKP provides foundational elements like OIDC integration for strong authentication (Req 8) and a logging/monitoring stack (Req 10), these are building blocks. The organization must then configure these features appropriately—for example, ensuring the OIDC provider enforces MFA for all CDE access, and that the KKP MLA stack is configured to collect all necessary logs, retain them as per PCI DSS, and forward them to a SIEM for proper analysis and alerting.

Furthermore, many PCI DSS requirements, such as thorough vulnerability management (Req 5, 6, 11), advanced network security controls like WAFs (Req 6.4.2), FIM (Req 11.5.2), and strong secrets management (Req 3), necessitate the integration of specialized third-party tools. KKP's open architecture allows for such integrations, but the responsibility for selecting, implementing, and managing these tools securely falls on the customer. The "Customized Approach" offered by PCI DSS 4.0 5 provides an avenue for organizations to use Kubernetes-native or innovative solutions, but this requires rigorous targeted risk analysis and detailed documentation to prove that the chosen controls meet the intent of the standard. This is particularly relevant where traditional, prescriptive controls are difficult to apply directly in a dynamic, containerized KKP environment.

5. Hardening Guidelines for KKP in a PCI DSS Environment

Securing a Kubermatic Kubernetes Platform (KKP) environment for PCI DSS compliance demands a multi-layered hardening strategy. This strategy must encompass all architectural components, from the KKP control plane (Master and Seed Clusters) down to the User Clusters and their worker nodes where the Cardholder Data Environment (CDE) resides. A "defense-in-depth" approach is essential, ensuring that multiple security controls protect sensitive data and systems.

5.1. KKP Control Plane Hardening (Master and Seed Clusters)

The KKP control plane, consisting of Master and Seed Clusters, is critical to the overall security of the managed User Clusters. Compromise of these components could lead to widespread impact on CDE workloads.

Securing Master Cluster Components:

The Master Cluster hosts the KKP API, UI Dashboard, and potentially identity management components like Dex if KKP's built-in OIDC provider is used.[19]

- **Access Control:** Network access to the Master Cluster's API and UI should be strictly limited, ideally through VPNs or private networks. Role-Based Access Control (RBAC) within KKP should be configured based on the principle of least privilege for all administrative users.
- **Authentication:** Enforce Multi-Factor Authentication (MFA) for all administrative access to the KKP Master Cluster UI and API, as mandated by PCI DSS 4.0 Req 8.4.2.6 KKP's OIDC integration can facilitate this by using an IdP that supports MFA.[1]
- **Component Hardening:** Secure the underlying Kubernetes cluster hosting the Master components according to general Kubernetes hardening best practices (e.g., CIS Benchmarks for Kubernetes).

Securing Seed Cluster Components:

Seed Clusters host the control planes (kube-apiserver, etcd, controller-manager, scheduler) for the User Clusters.[1] If these User Clusters are part of the CDE, the Seed Cluster components are inherently in-scope for PCI DSS.

- **Network Isolation:** Seed Clusters should be deployed in isolated network segments (e.g., dedicated VPCs/VLANs) with strict firewall rules controlling traffic to and from the Master Cluster, and to the worker nodes of the User Clusters they manage.[39]
- **Secure Communication:** All communication between the Master Cluster and Seed Clusters, and between Seed Clusters and User Cluster worker nodes (e.g., kubelet to API server), must be encrypted using strong TLS protocols.[3] KKP generally handles this for its managed components.
- **etcd Security:** The etcd instances for each User Cluster control plane, residing on the Seed Cluster, must be secured. This includes enabling etcd encryption at rest for sensitive data like Secrets (see Section 5.6), restricting network access to etcd, and using TLS for etcd client-server and peer communication.[3]
- **API Server Egress Policies:** KKP provides API Server Network Policies to restrict egress traffic from User Cluster API server pods running in the Seed Cluster to only necessary control plane components like etcd and dns-resolver.[27] This helps contain the blast radius in case of an API server compromise.

Principle of Least Privilege:

Apply the principle of least privilege to all KKP components and their service accounts. KKP's internal service accounts and permissions should be reviewed to ensure they only have the necessary access to perform their functions.

Regular Patching and Updates:

A disciplined patch management process is essential for all KKP platform components (Master and Seed). Kubermatic provides updates for KKP; organizations are responsible for applying these updates in a timely manner, especially for self-hosted KKP deployments.[1] This aligns with PCI DSS Req 6.

The distributed nature of KKP, where User Cluster control planes are managed on Seed Clusters, means that the security posture of a Seed Cluster directly impacts all User Clusters it manages. If a Seed Cluster manages CDE User Clusters, that Seed Cluster is a highly critical asset requiring the most stringent security measures.

5.2. User Cluster and Worker Node Hardening

User Clusters host the CDE applications and are a primary focus for PCI DSS compliance. Worker nodes within these clusters require thorough hardening.

- **Operating System Hardening:** The underlying operating system (OS) of worker nodes must be hardened. This involves:
  - Using minimal OS distributions to reduce the attack surface.[50]
  - Applying security configurations based on industry-standard benchmarks like the Center for Internet Security (CIS) Benchmarks for the specific OS (e.g., Linux variants).[51]
  - Disabling unnecessary services and ports.
  - Implementing host-based firewalls.
- **Kubelet Hardening:** The kubelet is a critical agent running on each worker node.
  - Secure its configuration file with appropriate permissions.
  - Restrict kubelet permissions using RBAC.[3]
  - Enable TLS for all kubelet communication (to API server, for metrics).
  - Disable the read-only port if not strictly necessary, or ensure it's properly secured.
  - Ensure --anonymous-auth=false and --authorization-mode=Webhook are set for the kubelet.
- **Container Runtime Hardening:** Secure the configuration of the container runtime (e.g., containerd, CRI-O) according to best practices and CIS Benchmarks for the runtime.[46] This includes ensuring that the runtime is up-to-date and configured with security-enhancing options.
- **Pod Security Admission (PSA) / OPA Gatekeeper:**
  - PodSecurityPolicy (PSP) was deprecated in Kubernetes v1.21 and removed in v1.25. KKP does not support PSP on Kubernetes 1.25 or higher and recommends Open Policy Agent (OPA) as a replacement.[100]
  - **Pod Security Admission (PSA)** is the built-in replacement. PSA allows defining security standards (Privileged, Baseline, Restricted) at the namespace level using labels (pod-security.kubernetes.io/enforce, pod-security.kubernetes.io/audit,

pod-security.kubernetes.io/warn).[53] For CDE namespaces, the Restricted profile should be enforced where feasible, or at least Baseline.

- **OPA/Gatekeeper or Kyverno:** For more fine-grained policy enforcement beyond PSA's predefined profiles, policy engines like OPA/Gatekeeper (which KKP integrates 1) or Kyverno should be used.[55] These tools can enforce custom policies critical for PCI DSS, such as:
  - Disallowing privileged containers.
  - Restricting hostPath mounts.
  - Limiting Linux capabilities.
  - Requiring runAsNonRoot and specific runAsUser/runAsGroup.
  - Enforcing read-only root filesystems for containers.
  - Mandating specific labels or annotations for CDE resources.

Hardening at the User Cluster and worker node level is fundamental because these components directly run and support the CDE applications. A failure to harden these layers can undermine all other security controls. The transition from PSP to PSA and policy engines means organizations must adapt their pod security strategies, defining policies in Rego (for OPA) or as Kyverno policies to meet PCI DSS requirements for secure configurations (Req 2) and least privilege.

5.3. Securing Container Images and Registries

Container images form the building blocks of applications within the KKP CDE. Securing the image supply chain is vital for PCI DSS compliance, particularly Req 5 (malware protection) and Req 6 (secure systems and software).

- **Trusted Image Sources:**
  - Only use container images from trusted, approved registries. Preferably, use private registries (e.g., Harbor, Artifactory, cloud provider registries like ECR, GCR, ACR) to store vetted and approved base images and application images.[3]
  - Strictly control which images can be pulled into CDE User Clusters, potentially using admission controllers (OPA/Gatekeeper) to enforce policies based on image registry source.[55]
- **Image Vulnerability Scanning:**
  - Integrate automated vulnerability scanning into the CI/CD pipeline at multiple stages:
    - When base images are ingested.
    - After application code is built into an image.
    - Periodically for images stored in registries.

        - Optionally, before deployment using admission control, and continuously for running containers.
    - Tools like Aqua Security (which lists Kubermatic among its supported Kubernetes platforms 73), Sysdig, Trivy, Snyk, Clair, or Qualys can be used.[42]
    - Findings should be risk-ranked (e.g., using CVSS scores), and high-risk vulnerabilities must be remediated according to PCI DSS timelines (Req 6.3.1).
- **Minimal Base Images:**
    - Use minimal base images (e.g., distroless, Alpine Linux, or custom-built minimal images) to reduce the attack surface by excluding unnecessary libraries, shells, and utilities.[50] This limits potential vulnerabilities and tools available to an attacker if a container is compromised.
- **Image Signing and Verification:**
    - Implement image signing to ensure the integrity and authenticity of images deployed into the CDE.
    - Tools like Cosign (part of Sigstore) or Notation (the Notary Project's signing client, formerly "Notary v2") can be used to sign images in the CI/CD pipeline.[3]
    - Configure Kubernetes admission controllers (e.g., OPA/Gatekeeper, Kyverno) to verify image signatures before allowing pods to run, ensuring only trusted and unmodified images are deployed.
- **Regular Updates:** Ensure base images and application dependencies are regularly updated to include the latest security patches.

The dynamic nature of containerized environments means that image security cannot be a one-time check. Continuous scanning and monitoring are essential. For PCI DSS, demonstrating a well-defined process for building, storing, scanning, and deploying secure images is key to meeting requirements related to vulnerability management and malware prevention.

5.4. Network Security Hardening

Network security is a cornerstone of PCI DSS, with Requirement 1 mandating the installation and maintenance of network security controls. In a KKP environment, this involves hardening at multiple layers.

- **CNI Configuration (Cilium/Canal):** KKP supports various CNIs. Specific hardening depends on the chosen CNI:
    - **Cilium:** Uses eBPF for advanced networking and security.[25]
        - Prefer identity-based network policies over IP-based policies where possible.
        - Use DNS-aware policies to control egress traffic based on FQDNs.

        - Explore L7 policy enforcement capabilities for protocols like HTTP and Kafka if applicable to CDE traffic flows.[44]
        - Enable Hubble for network flow visibility and troubleshooting within CDE User Clusters.[105]
        - Cilium (supported by KKP as a CNI) provides transparent encryption (WireGuard or IPsec) and, via its Tetragon component, eBPF runtime security and file-integrity-style monitoring that can directly support PCI DSS requirements. Commercial support and additional enterprise features are available through Isovalent Enterprise for Cilium, now part of Cisco's Isovalent Enterprise Platform following Cisco's April 2024 acquisition of Isovalent.[25]
    - **Canal (Calico for policy, Flannel for networking) / Calico:**
        - Focus on implementing strict Kubernetes NetworkPolicies using Calico's rich policy model.
        - Ensure Calico components (e.g., Typha, Felix) are securely configured and monitored.
- **Kubernetes NetworkPolicies:**
    - Implement a default-deny policy for all ingress and egress traffic in CDE namespaces.[3] This ensures that no traffic is allowed unless explicitly permitted.
    - Create specific NetworkPolicies to allow only necessary communication paths:
        - Between pods within the CDE.
        - From specific Ingress controllers to frontend pods.
        - From application pods to backend databases or services within the CDE.
        - Egress traffic from CDE pods to approved external services (e.g., payment gateways, update servers) on specific ports and protocols.[3]
- **Ingress/Egress Traffic Control:**
    - **Secure Ingress:** Configure Ingress controllers (e.g., Nginx Ingress, Traefik) securely. This includes using TLS, restricting allowed annotations, and regularly updating the controller. KKP can deploy and manage Ingress controllers as add-ons.
    - **Web Application Firewalls (WAFs):** PCI DSS 4.0 Req 6.4.2 mandates an automated technical solution (like a WAF) to protect public-facing web applications from web-based attacks.[6] Integrate a WAF at the edge of the CDE, in front of Ingress controllers. This WAF should be capable of inspecting traffic to CDE applications and blocking known and emerging threats (e.g., OWASP Top 10).
    - **Egress Filtering:** Control outbound traffic from the CDE. This can be achieved using:
        - Kubernetes NetworkPolicies (for pod-to-pod and pod-to-external IP/port).

    - Service mesh egress gateways (e.g., Istio Egress Gateway) for more advanced control and monitoring of outbound traffic.[33]
    - Dedicated egress firewalls or cloud provider NAT gateways with outbound filtering rules.[38]

- **Encryption in Transit:**
    - **TLS for External Communication:** All communication with external entities (e.g., user browsers to web applications, CDE applications to external payment gateways) must use strong TLS (TLS 1.2 or higher, with secure cipher suites and protocols).[3]
    - **Mutual TLS (mTLS) for Internal Communication:** Encrypt and authenticate all service-to-service (pod-to-pod) communication within the CDE using mTLS. This can be implemented using a service mesh like Istio or Linkerd, or potentially with Cilium's service mesh capabilities.[25] This provides defense-in-depth against lateral movement if a pod within the CDE is compromised.
- **DMZ Architecture:** For internet-facing CDE components, implement a Demilitarized Zone (DMZ) architecture to control traffic between untrusted networks (internet) and the CDE.[94] In KKP, this might involve specific User Clusters or namespaces acting as the DMZ, with stricter firewalling and WAFs.

Network security hardening is paramount in a KKP environment intended for PCI DSS. The platform's support for various CNIs and its distributed architecture require careful planning of network segmentation, traffic flow control, and encryption strategies to ensure that cardholder data is protected both at the perimeter and within the CDE. The mandatory nature of WAFs for public web applications under PCI DSS 4.0 necessitates integrating such solutions into the KKP network architecture if CDE applications are web-facing.

5.5. API Server Security

The Kubernetes API server is the central management point for a Kubernetes cluster. In KKP, each User Cluster has its own API server (control plane running on a Seed Cluster), and the Master Cluster has its own API server for KKP management. Securing these API servers is critical for PCI DSS compliance.

- **Authentication:**
    - **Disable Anonymous Access:** The --anonymous-auth=false flag should be set for all API servers (Master, Seed-managed User Cluster API servers) to prevent unauthenticated access.[3] KKP typically configures this by default for User Clusters.
    - **Strong Authentication (OIDC + MFA):** Enforce strong authentication for all users and service accounts accessing any API server.
        - KKP supports OIDC integration, allowing federation with enterprise identity providers (IdPs).[1] The IdP must be configured to enforce Multi-Factor Authentication (MFA) for

all users accessing CDE-related clusters or KKP management interfaces that can impact the CDE. This is a direct requirement of PCI DSS 4.0 Req 8.4.2.6

    - Service accounts should use token-based authentication, with tokens having short lifespans and tight permissions.

- **Authorization (RBAC):**
    - Implement the principle of least privilege for all users, groups, and service accounts using Kubernetes Role-Based Access Control (RBAC).[3]
    - Define granular Roles and ClusterRoles that grant only necessary permissions (verbs like get, list, watch, create, update, patch, delete) on specific resources (pods, secrets, networkpolicies, etc.) within specific namespaces or cluster-wide.
    - Avoid using overly permissive roles like cluster-admin for day-to-day operations or for service accounts unless absolutely necessary and tightly controlled.
    - Regularly audit RBAC policies and bindings to identify and remediate excessive permissions.[50]
- **Network Access:**
    - Restrict network access to the API server endpoint.
        - For KKP User Clusters, KKP allows configuring apiServerAllowedIPRanges in the Cluster spec to limit which IP CIDR blocks can access the API server.[27] This should be configured to allow access only from trusted networks (e.g., corporate network, CI/CD systems, specific bastion hosts).
        - Consider using private Kubernetes clusters where the API server endpoint is not exposed to the public internet, if supported by the underlying infrastructure and KKP deployment model.[38]
    - KKP also implements API Server Network Policies for User Cluster API servers running in Seed Clusters, restricting their egress traffic to essential control plane components.[27]
- **TLS Encryption:**
    - Ensure all communication to and from the API server uses strong TLS encryption (TLS 1.2 or higher) with valid, trusted certificates.[3] KKP manages certificates for its components.
- **Audit Logging:**
    - Enable thorough Kubernetes audit logging for all API server requests and responses. This is essential for PCI DSS Req 10.3
    - Configure an audit policy that captures relevant events, especially for CDE clusters:
        - Metadata level for most read operations.
        - Request or RequestResponse level for write operations (create, update, patch, delete), access to Secrets, modifications to NetworkPolicies, Pod exec, and port-forwards.

- Audit logs should be shipped to a centralized, secure SIEM for analysis and retention (see Section 6.1).

Securing the API server is a foundational element of Kubernetes security. Given KKP's architecture where User Cluster API servers run on Seed Clusters, the security of these Seed Clusters, including network isolation and access controls, is paramount for protecting the API servers of CDE User Clusters.

5.6. etcd Data Store Protection

The etcd data store holds the state of each Kubernetes cluster, including sensitive information like Secrets, ConfigMaps, and resource definitions. In KKP, each User Cluster has its own dedicated etcd instances, which run as part of its control plane on a Seed Cluster.[1] Protecting these etcd instances is vital for PCI DSS Req 3 (Protect stored account data).

- **Encryption at Rest:**
  - PCI DSS Req 3.5.1 requires PAN to be rendered unreadable wherever it is stored. If Kubernetes Secrets are used to store PAN or other CHD (e.g., encryption keys, sensitive configuration), they must be encrypted at rest in etcd.
  - Kubernetes supports enabling encryption at rest for API resources (like Secrets and ConfigMaps) via the API server's --encryption-provider-config flag.[3] This typically involves configuring an encryption provider (e.g., aescbc, aesgcm, or an external KMS provider).
  - KKP v2.30 provides a first-class **Data Encryption at Rest** feature for User Clusters, configured declaratively via the Cluster resource's `spec.encryptionConfiguration` (feature-gated). Administrators choose which resources to encrypt (e.g. Secrets) and the feature supports encryption-key rotation via secondary keys; it can also be enabled or disabled from the KKP dashboard on a running cluster (since v2.29). This should be enabled for all CDE clusters.
  - Cloud provider-managed Kubernetes services often offer built-in etcd encryption, sometimes using envelope encryption with a KMS (e.g., Amazon EKS 61). For KKP deployments on self-managed infrastructure or on IaaS, this needs to be explicitly configured. OpenShift documentation also provides examples of enabling etcd encryption.[62]
- **Access Control:**
  - Direct access to etcd instances should be highly restricted. Communication with etcd should primarily occur through the Kubernetes API server.
  - Use strong authentication (TLS client certificates) for any permitted direct access (e.g., for administrative or backup purposes).
  - Ensure etcd peer communication (between etcd members in a cluster) and client-server communication (API server to etcd) are secured using TLS.

- **Backups:**
  - KKP supports configuring etcd backups for User Clusters, storing them in locations like S3 or MinIO.[1]
  - These backups must be encrypted, both in transit to the backup location and at rest in the backup storage.
  - Access to backup locations and backup encryption keys must be strictly controlled.
  - Regularly test etcd restore procedures to ensure data can be recovered in a disaster scenario, which is important for data availability aspects of PCI DSS.
- **Physical/Logical Isolation:**
  - Since User Cluster etcd instances run on Seed Clusters in KKP, the physical and logical security of the Seed Cluster infrastructure is critical. This includes network isolation of the Seed Cluster and strong access controls to the nodes running etcd pods.

The protection of etcd is fundamental because it stores the entire configuration and state of CDE User Clusters. A compromise of etcd could lead to a full compromise of the cluster and the data it manages. Enabling encryption at rest for Secrets in etcd is a non-negotiable control if those Secrets contain or protect CHD.

5.7. Secrets Management Strategies

Managing sensitive information such as API keys, passwords, TLS certificates, and database credentials (collectively, "secrets") is critical for PCI DSS compliance, particularly Req 3 (Protect stored account data) and Req 8 (Identify users and authenticate access).

- **Kubernetes Secrets:**
  - By default, Kubernetes Secrets are stored in etcd as Base64-encoded strings, which is not encryption.[67] To protect them, etcd encryption at rest must be enabled (as discussed in Section 5.6).[3]
  - Access to Kubernetes Secrets must be tightly controlled using RBAC. Only pods and users with a legitimate need-to-know should have get, list, or watch permissions for specific Secrets.[3]
  - **Best Practices for Kubernetes Secrets:**
    - Scope Secrets permissions to the namespace level where possible, avoiding cluster-wide grants.
    - Avoid mounting all Secrets in a namespace into a pod; only mount the specific Secrets required.
    - Prefer projected service account tokens (time-bound and audience-bound) over non-expiring default service account tokens where possible.
    - Do not log secret data.

- **External Secrets Management Solutions:**
  - For enhanced security, especially in CDE environments, consider using external secrets management solutions like HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, or Google Cloud Secret Manager.[66]
  - **Benefits:**
    - **Centralized Management:** Manage secrets for multiple KKP clusters and other applications from a single, secure store.
    - **Strong Encryption:** These solutions typically offer strong encryption mechanisms, often backed by HSMs or KMSs.
    - **Dynamic Secrets:** Some vaults can generate secrets on-demand with short lifespans (e.g., database credentials).
    - **Fine-grained Access Control:** Detailed policies control who or what can access specific secrets.
    - **Audit Trails:** Thorough audit logs of secret access and management operations.[67]
  - **Integration with KKP/Kubernetes:**
    - **Secrets Store CSI Driver:** This driver allows Kubernetes to mount secrets stored in external vaults as volumes directly into pods, without the secrets being stored in etcd.[66] This is a recommended approach.
    - **Vault Agent Injector:** For HashiCorp Vault, the agent injector can automatically inject secrets into pods as files or environment variables.
  - **PCI DSS Considerations for External Vaults:** The vault itself, if managing secrets for the CDE, becomes a critical component and must be secured according to PCI DSS requirements. This includes secure key management for the vault's master key, strong authentication and authorization for vault access, and detailed auditing.[108]
- **Secrets Rotation:**
  - Implement a process for regularly rotating all secrets, including passwords, API keys, and certificates.[42]
  - Automate rotation where possible, especially when using dynamic secrets from an external vault. Under PCI DSS 4.0, Req 8.3.9 governs user passwords where a password is the only authentication factor (change at least every 90 days, or dynamically analyze account posture to determine real-time access), while Req 8.6.3 governs passwords/passphrases for application and system accounts (changed periodically and upon suspicion of compromise, with complexity defined by a targeted risk analysis).

Choosing the right secrets management strategy involves balancing security requirements, operational complexity, and cost. For PCI DSS CDEs on KKP, using an external, hardened secrets management solution integrated via the Secrets Store CSI Driver is generally the preferred

approach over relying solely on native Kubernetes Secrets, even with etcd encryption enabled. This approach minimizes the attack surface within the Kubernetes cluster itself by keeping sensitive data out of etcd.

5.8. Persistent Volume Encryption and Key Management

Protecting cardholder data at rest stored in persistent volumes (PVs) is a fundamental requirement of PCI DSS Req 3. This involves encrypting the data on the storage media and securely managing the encryption keys.

- **Encryption at Rest for Persistent Volumes (PVs):**
  - **Using Underlying Storage Provider Capabilities:** Most cloud providers offer native encryption for their block storage services (e.g., AWS EBS encryption using KMS, Azure Disk Encryption using Azure Key Vault, GCP Persistent Disk Encryption using Cloud KMS).[106] When KKP provisions User Clusters on these cloud providers, these native encryption features should be enabled for all PVs used by CDE workloads. This is typically configured via StorageClasses in Kubernetes, which specify the encryption parameters.
  - **On-Premises Deployments:** For KKP deployments on-premises, organizations must use storage solutions that provide encryption at rest. This could be:
    - Self-Encrypting Drives (SEDs).
    - Software-based encryption at the storage array level.
    - Filesystem-level encryption (e.g., LUKS on Linux worker nodes, though this can be complex to manage at scale for PVs).
    - Container Storage Interface (CSI) drivers that integrate with external encryption and key management systems.[109]
  - **Encryption Algorithms:** Ensure strong, industry-accepted encryption algorithms like AES-256 are used for volume encryption.[60]
- **Key Management for Volume Encryption:**
  - Securely managing the encryption keys used for volume encryption is as important as the encryption itself. PCI DSS Req 3.6 and 3.7 outline requirements for cryptographic key management.
  - **Key Management Service (KMS) / Hardware Security Module (HSM):** Use a strong KMS (like those offered by cloud providers) or an on-premises HSM to manage Data Encryption Keys (DEKs) and Key Encryption Keys (KEKs).[6]
    - DEKs are used to encrypt the data on the volumes.
    - KEKs are used to encrypt the DEKs. KEKs should be stored in a secure KMS or HSM.
  - **Customer-Managed Keys (CMK) / Bring Your Own Key (BYOK):** For greater control and to meet specific compliance requirements, organizations can use CMKs or BYOK options

offered by cloud providers or integrated with their KMS/HSM solutions.[106] This allows the customer to manage the lifecycle of the KEKs.

- **Key Rotation:** Implement policies and procedures for regular rotation of KEKs and, where feasible, DEKs, according to PCI DSS guidelines and risk assessments.[34]
- **Access Control to Keys:** Strictly control access to encryption keys. Only authorized personnel and processes should have permissions to manage or use keys. This involves strong authentication and authorization for accessing the KMS/HSM.
- **Separation of Duties:** Separate key management duties from other administrative roles where possible.[109]

When KKP is deployed, the choice of storage provider and CSI driver will dictate the available options for persistent volume encryption. It is the organization's responsibility to ensure that these options are configured to meet PCI DSS requirements for all PVs within the CDE. The security of the encryption keys is paramount; if keys are compromised, the encryption becomes ineffective. Therefore, a strong focus on key management practices, using trusted KMS/HSM solutions, is essential. Kubermatic's DPA mentions AES-256 encryption for data in transit and measures for data at rest encryption, indicating an awareness of these needs, though specific PV encryption is customer-configured.[30]

6. Security Operations for a PCI DSS Compliant KKP Environment

Effective security operations are essential for maintaining PCI DSS compliance in a dynamic Kubermatic Kubernetes Platform (KKP) environment. This involves continuous monitoring, logging, alerting, vulnerability management, intrusion detection, and a strong incident response capability, all tailored to the specifics of the KKP architecture and the Cardholder Data Environment (CDE).

6.1. Strong Monitoring, Logging, and Alerting

PCI DSS Requirement 10 mandates thorough logging and monitoring of all access to network resources and cardholder data. This includes daily log reviews and retaining logs for at least one year, with the last 90 days readily accessible for analysis.[52]

- **KKP Monitoring, Logging, and Alerting (MLA) Stack:**
  - KKP provides a built-in MLA stack based on Prometheus for metrics, Loki for logs, and Grafana for dashboards and visualization.[1] This stack is available for both Master/Seed management components and for User Clusters.[1]
  - **Configuration for PCI DSS:**
    - Ensure the User Cluster MLA stack is enabled for all User Clusters within the CDE.

        - Configure Prometheus to scrape relevant metrics from CDE components (nodes, pods, Kubernetes API server, etcd, CNI, applications).
        - Configure Loki (via Promtail or other log shippers) to collect logs from all CDE components, including Kubernetes audit logs, container logs (stdout/stderr from CDE applications), node OS logs (syslog/journald), and logs from security tools (e.g., WAF, IDS/IPS).[79]
        - Develop Grafana dashboards specifically for monitoring the security posture and operational health of the CDE.
        - Configure Prometheus Alertmanager with alerting rules for critical security events and operational issues within the CDE (e.g., CDE component failures, resource exhaustion, security policy violations, high rates of failed logins).[3]
    - KKP's MLA stack can be configured per seed, allowing for tailored data collection based on organizational structure or CDE requirements.[1]
- **Kubernetes Audit Logs:**
    - Enable and configure thorough audit logging on the API servers of all User Clusters within the CDE, as well as the KKP Master Cluster API server.[3]
    - Define an audit policy that captures:
        - User identities, source IPs, and timestamps for all requests.
        - Requests for creating, modifying, or deleting sensitive resources (Secrets, ConfigMaps, NetworkPolicies, RoleBindings, Pods with exec or port-forward).
        - Failed API requests, especially authentication and authorization failures.
        - Use RequestResponse level for critical write operations and Metadata or Request for others to balance detail with log volume.[74]
- **SIEM Integration:**
    - Centralize all relevant logs from the KKP environment into a Security Information and Event Management (SIEM) system. This includes:
        - Logs from KKP's User Cluster MLA (Loki).
        - Kubernetes audit logs from CDE User Cluster API servers.
        - Logs from worker nodes (OS logs, container runtime logs).
        - Application logs from CDE workloads.
        - Logs from network security controls (firewalls, WAFs, IDS/IPS).
        - Logs from vulnerability scanners and FIM tools.
    - Popular SIEM solutions include Splunk, Elasticsearch (ELK Stack), IBM QRadar (note: IBM divested its QRadar SaaS to Palo Alto Networks in 2024; the on-premises product remains

with IBM), Microsoft Sentinel (formerly Azure Sentinel), Google Security Operations (formerly Chronicle), Datadog, and LogRhythm.[3]

- The SIEM is used for log correlation, advanced threat detection, generating alerts for security incidents, long-term log retention (PCI DSS requires 1 year, 90 days readily accessible), and facilitating forensic investigations.[78]

- **Log Protection and Time Synchronization:**
  - Protect all logs from unauthorized modification or deletion.[91] This often involves write-once storage or strict access controls on the SIEM.
  - Ensure all systems within the KKP CDE, including Master, Seed, and User Cluster components, are synchronized to a common, accurate time source (NTP) as per PCI DSS Req 10.5.78 This is critical for correlating events across different log sources.

Strong logging and monitoring form the bedrock of security operations. In a KKP environment, this means not only using the platform's built-in MLA capabilities but also ensuring these are configured to capture PCI DSS-relevant data and are integrated with a centralized SIEM. The distributed nature of KKP, with potentially many User Clusters, makes centralized logging and analysis even more critical for maintaining visibility and control over the CDE.

6.2. Vulnerability Management and Patching

PCI DSS Requirements 5 and 6 mandate the protection of systems against malware, the development and maintenance of secure systems and software, and the prompt addressing of security vulnerabilities.[14] PCI DSS 4.0 Req 11.3.1.2 specifically requires authenticated internal vulnerability scans every three months.[16]

- **Authenticated Vulnerability Scanning:**
  - Implement a program for regular (at least quarterly) authenticated internal vulnerability scans of all in-scope KKP components (Master, Seed, User Cluster nodes) and CDE workloads.
  - Tools like Qualys, Tenable Nessus, Rapid7 InsightVM, or open-source scanners can be used.[16]
  - Authenticated scans provide deeper visibility into system vulnerabilities compared to unauthenticated scans by logging into systems to inspect configurations and installed software.[16] This often results in a higher number of findings, requiring strong prioritization and remediation processes.[17]
  - Securely manage credentials used for authenticated scans.
- **Container Image Scanning:**
  - Integrate image scanning throughout the container lifecycle:
    - **Build Time:** Scan images as part of the CI/CD pipeline before they are pushed to a registry.

        - **Registry:** Continuously scan images stored in private registries.
        - **Runtime:** Optionally, scan running containers for newly discovered vulnerabilities.
    - Tools like Aqua Security (with KKP integration 73), Sysdig, Trivy, Clair, and Snyk are commonly used.[42]
    - Focus on vulnerabilities in OS packages, application libraries, and base image layers.
- **Patch Management Process for KKP:**
    - **KKP Platform Components (Master, Seed Clusters):**
        - If using self-hosted KKP, the organization is responsible for patching KKP software components, the underlying Kubernetes clusters for Master/Seed, and their OS. Kubermatic releases KKP updates, and these should be applied promptly based on risk assessment.
        - If using Managed KKP, Kubermatic manages the patching of the KKP platform itself as per the service agreement.[29]
        - KKP automates aspects of cluster lifecycle management, including upgrades, which can simplify the patching process.[1]
    - **User Cluster Control Planes & Worker Nodes:**
        - KKP allows administrators to initiate upgrades for User Cluster control planes (running on Seed Clusters) and worker nodes.[1] This includes upgrading Kubernetes versions and, depending on the infrastructure provider, potentially OS patches for worker nodes.
        - KKP enforces kubelet version compatibility rules during upgrades. Per the Kubernetes version-skew policy, kubelet must not be newer than kube-apiserver and may be up to three minor versions older (the older two-version limit applied only to kubelet versions below 1.25); in HA setups kubelet must not be newer than the oldest kube-apiserver instance.[1]
        - Organizations are responsible for ensuring worker node OS patching is performed regularly, either through KKP-managed upgrades (if the provider integration supports it) or via separate OS patch management tools.
    - **Workloads within CDE:** The organization is always responsible for patching vulnerabilities in their application code and any custom or third-party software deployed within containers in the CDE. This also includes regularly updating container base images to incorporate security fixes.
- **Prioritization and Remediation:**
    - Vulnerabilities must be risk-ranked according to PCI DSS Req 6.3.1 (e.g., using CVSS scores and considering CDE impact).[16]
    - High-risk and critical vulnerabilities must be remediated promptly. PCI DSS 4.0 emphasizes addressing *all* vulnerabilities, with timelines based on risk.

    - Automated remediation tools or processes should be used where feasible to ensure timely patching and reduce manual effort.[104]
- **Anti-Malware (Req 5):** Deploy and maintain anti-malware solutions on all systems commonly affected by malware, particularly worker nodes within the CDE.[70] For containers, this often translates to thorough image scanning for malware and runtime threat detection.

Effective vulnerability and patch management in a KKP environment requires a clear understanding of responsibilities across the KKP platform layers, the underlying infrastructure, and the applications. The automation capabilities of KKP for cluster upgrades can aid in this process, but a thorough strategy involving scanning, risk assessment, and timely remediation is essential for PCI DSS compliance. The introduction of mandatory authenticated scans in PCI DSS 4.0 will necessitate more thorough vulnerability assessment processes.

6.3. Intrusion Detection, Prevention, and Response

Detecting and responding to intrusions in real-time is critical for protecting the CDE. This involves a combination of network and host-based intrusion detection/prevention systems (IDS/IPS), runtime security monitoring, and file integrity monitoring (FIM).

- **Runtime Security Monitoring for Containers and Nodes:**
    - Continuously monitor containers and worker nodes within the CDE for anomalous behavior, unauthorized processes, privilege escalations, and container escapes.[3]
    - **Tools:**
        - **Falco:** An open-source CNCF project for runtime threat detection in Kubernetes. It uses eBPF to monitor system calls and can detect suspicious activity based on predefined and custom rules.[3]
        - **Cilium with eBPF (and Tetragon):** Cilium, supported by KKP, can provide deep network visibility and enforcement using eBPF. Its Tetragon component extends this to runtime security, offering kernel-level visibility and enforcement capabilities, including process execution monitoring and file access monitoring.[25] Isovalent Enterprise for Cilium (now part of Cisco's Isovalent Enterprise Platform) documents file-integrity monitoring relevant to PCI DSS.[25]
        - **Aqua Security CNAPP:** Offers runtime protection for Kubernetes workloads, including behavioral analysis, drift prevention (preventing changes to running containers), malware detection, and intrusion prevention.[72] Aqua Security lists Kubermatic among its supported Kubernetes platforms.[73]
        - Other commercial CNAPPs/CWPPs (e.g., Sysdig, Wiz, Palo Alto Networks Prisma Cloud) provide similar runtime security features.
- **File Integrity Monitoring (FIM):**

- PCI DSS Req 10.3.4 (formerly 10.5.5, FIM/change-detection on audit logs) and Req 11.5.2 (formerly 11.5, FIM on critical system, configuration, and content files) require change-detection mechanisms to alert personnel to unauthorized modification.
- Implement FIM on KKP worker nodes within the CDE, and potentially for critical configuration files of KKP Master/Seed components if self-hosted and in-scope.
- Tools like Wazuh, Tripwire, OSSEC, or FIM capabilities within CNAPPs (e.g., Aqua Security 89, Cilium Tetragon 25) can be used.
- FIM should monitor for changes at least weekly (or more frequently based on risk for critical files).

- **Network Intrusion Detection/Prevention Systems (IDS/IPS):**
  - PCI DSS Req 11.5.1 (formerly 11.4) requires IDS/IPS to monitor traffic at the perimeter of the CDE and at critical points within the CDE.
  - Deploy IDS/IPS solutions to detect and/or prevent network-based attacks targeting CDE components or applications on KKP.
  - These can be traditional network appliances, virtual appliances, or cloud-native IDS/IPS services.
  - IDS/IPS should be kept up-to-date with the latest threat intelligence and signatures.
  - Logs and alerts from IDS/IPS must be sent to the SIEM for correlation and analysis.[35]

The combination of runtime security monitoring, FIM, and network IDS/IPS provides layered security for the KKP CDE. eBPF-based tools like Cilium/Tetragon are particularly well-suited for Kubernetes environments due to their kernel-level visibility and container awareness. Commercial CNAPPs often provide a suite of these capabilities, simplifying deployment and management. All alerts from these systems must feed into the incident response process.

6.4. Incident Response Planning and Execution for KKP CDE

A well-defined and tested Incident Response Plan (IRP) is mandated by PCI DSS Req 12.10.3 This plan must enable the organization to respond immediately to a system breach or security incident within the KKP CDE.

- **KKP Incident Response Framework Adaptation:**
  - Kubermatic provides a general framework for Kubernetes incident response, which includes phases of Identification, Coordination, Resolution, and Continuous Improvement.[95] This framework should be adapted and customized for the specifics of a PCI DSS CDE running on KKP.
- **Key Elements for a KKP CDE IRP:**
  - **Preparation:**
    - Define roles and responsibilities for incident response team members.

- Establish clear communication channels and escalation paths.
- Provide regular training to the incident response team on KKP architecture, CDE specifics, and PCI DSS breach notification requirements.
- Ensure necessary tools for investigation and containment are readily available.

- **Identification:**
  - Define what constitutes a security incident within the KKP CDE.
  - Use alerts from KKP's MLA stack, centralized SIEM (correlating logs from Kubernetes audit, applications, nodes, network devices, IDS/IPS, FIM), and other security tools to detect potential incidents.[95]
- **Containment:**
  - Develop procedures to quickly isolate compromised components to prevent further impact on the CDE. This could involve:
    - Isolating affected User Clusters or namespaces using KKP's management capabilities or by applying restrictive NetworkPolicies.
    - Quarantining compromised pods or worker nodes.[3]
    - Blocking malicious IP addresses at firewalls or WAFs.
  - Preserve evidence during containment (e.g., take snapshots of affected nodes/pods before shutting them down, collect memory dumps if feasible).[96]
- **Eradication:**
  - Identify and remove the root cause of the incident (e.g., malware, exploited vulnerability).
  - Apply necessary patches and harden affected systems.
  - Reset compromised credentials.
- **Recovery:**
  - Securely restore affected CDE services and data to normal operation.
  - KKP's automated backup and restore capabilities using Velero are essential here.[3] This includes the ability to restore specific namespaces or entire clusters, potentially to a different KKP instance if needed for DR.
  - Validate that systems are clean and functioning correctly before bringing them back online.
- **Lessons Learned (Post-Incident Activity):**
  - Conduct a post-incident review to analyze the cause, the effectiveness of the response, and identify areas for improvement in the IRP and overall CDE security on KKP.[95]
  - Update policies, procedures, and configurations based on lessons learned.

- **Data Breach Notification:**
  - The IRP must include procedures for notifying relevant parties in the event of a confirmed cardholder data breach. This includes payment brands, acquirers, legal counsel, and potentially regulatory bodies, as per PCI DSS requirements and applicable laws.[96]
- **Annual Testing:** The IRP must be tested at least annually to ensure its effectiveness.[96]

An effective IRP for a KKP CDE uses the platform's management and automation features (like backup/restore and cluster lifecycle management) while integrating with the broader security ecosystem (SIEM, IDS/IPS) to enable rapid detection, containment, and recovery from security incidents. The complexity of Kubernetes means that IRPs must be specifically tailored to containerized environments.

6.5. Managing Client-Side Security (Payment Page Scripts - Req. 6.4.3, 11.6.1)

PCI DSS 4.0 introduced new requirements to address the growing threat of e-skimming attacks, where malicious JavaScript on payment pages steals cardholder data directly from the consumer's browser. These are particularly relevant if applications hosted on KKP User Clusters serve payment pages.

- **PCI DSS 4.0 Requirements:**
  - **Requirement 6.4.3:** Mandates maintaining an inventory of all custom and third-party scripts that execute on payment pages. Each script must be authorized, and its integrity must be assured (e.g., via Subresource Integrity (SRI) or FIM).[11]
  - **Requirement 11.6.1:** Requires a change-and-tamper-detection mechanism to be implemented to alert personnel to unauthorized modification of HTTP headers and the content of payment pages as rendered in the consumer’s browser.[11]
- **Applicability to KKP:** If applications deployed within KKP User Clusters are responsible for rendering payment pages and handling cardholder data input directly in the user's browser, these requirements apply.
- **Implementation Strategies:**
  - **Inventory and Authorization:**
    - Maintain a documented inventory of all JavaScript files (first-party and third-party) loaded on payment pages.
    - For each script, document its purpose and obtain formal authorization for its use.
  - **Integrity Verification:**
    - **Subresource Integrity (SRI):** Use SRI attributes (<script integrity="sha384-...">) in HTML to ensure that fetched scripts have not been tampered with. The browser will only execute the script if its hash matches the specified integrity value.

        - **File Integrity Monitoring (FIM):** If scripts are served from your own infrastructure (hosted on KKP), use FIM to monitor the script files themselves for unauthorized changes.
    - **Change and Tamper Detection (Req 11.6.1):**
        - Implement solutions that monitor the payment page as rendered in the consumer's browser for any unauthorized changes to scripts or HTTP headers.
        - These solutions should generate alerts when unexpected modifications or behaviors are detected (e.g., new scripts appearing, existing scripts modified, data being exfiltrated to unknown domains).
    - **Client-Side Protection Tools:** Consider specialized third-party client-side security solutions designed to meet these requirements. Examples include:
        - Imperva Client-Side Protection.[6]
        - Akamai Client-Side Protection & Compliance.[13]
        - Other solutions that offer JavaScript behavior monitoring, script inventory, and alerting.
- **SAQ A Eligibility Changes:** For organizations that fully outsource their e-commerce payment processing to PCI DSS compliant third parties and meet the new SAQ A eligibility criteria (which now extend script-based-threat protection to the *entire website*, beyond the payment pages alone 13), the burden of these specific requirements might be shifted. However, this requires careful validation of the outsourcing arrangement and the third party's compliance.

Managing client-side script security is a critical new focus area in PCI DSS 4.0. For applications on KKP serving payment pages, this means extending security considerations beyond the server-side infrastructure to the end-user's browser environment. Implementing thorough inventory, integrity, and monitoring controls for payment page scripts is essential.

7. Using Third-Party Integrations for Enhanced PCI DSS Compliance on KKP

While Kubermatic Kubernetes Platform (KKP) and native Kubernetes features provide a foundational layer for security, achieving full PCI DSS 4.0 compliance often necessitates the integration of specialized third-party security solutions. These tools can augment KKP's capabilities in areas like advanced network security, runtime protection, policy enforcement, and secrets management. The strategic selection and integration of these tools are essential, as they can significantly simplify meeting specific PCI DSS requirements that might be challenging to address with KKP or Kubernetes native features alone.

7.1. Service Mesh (e.g., Istio, Linkerd)

A service mesh can provide critical security capabilities for applications running within KKP User Clusters, particularly for those within the Cardholder Data Environment (CDE).

- **Benefits for PCI DSS:**
  - **Mutual TLS (mTLS):** Service meshes like Istio can automatically enforce mTLS for all service-to-service (pod-to-pod) communication within the CDE.[33] This ensures that traffic is encrypted and both communicating parties are authenticated, directly supporting PCI DSS Req 4.1 (Protect cardholder data with strong cryptography during transmission) and Req 2.3 (Secure wireless networks, if applicable, and encrypt all non-console administrative access). The automated nature of mTLS in a service mesh simplifies implementation compared to manual certificate management for every microservice.
  - **Fine-grained Authorization Policies:** Istio's AuthorizationPolicy custom resource allows defining granular rules that control which services can communicate with each other, based on identity, namespace, HTTP methods, paths, etc..33 This enhances network segmentation within the CDE (Req 1.2.1, Req 2.2.1) and helps enforce the principle of least privilege for service interactions (Req 7).
  - **Traffic Monitoring and Auditing:** Service meshes generate detailed telemetry (metrics, logs, traces) for all service interactions.[33] This data is invaluable for PCI DSS Req 10 (Log and monitor all access), providing insights into traffic patterns, identifying anomalous communication, and aiding in forensic investigations. These logs can be exported to a central SIEM.
- **Integration with KKP:**
  - A service mesh is typically deployed within a User Cluster managed by KKP. Helm charts or operators are common methods for installation and management.
  - Organizations like Tetrate offer enterprise distributions of Istio specifically designed for PCI compliance, which can simplify deployment and policy management.[33]
  - Careful planning is needed for certificate management (e.g., integrating with an existing PKI or using the mesh's built-in CA) and resource overhead.

While advanced CNIs like Cilium (supported by KKP) are increasingly offering some service mesh-like capabilities (e.g., identity-based policies, potential for mTLS through its own mechanisms or integration with WireGuard/IPsec 25), a dedicated service mesh like Istio might still be preferred for its mature and broad feature set for traffic management, observability, and security, especially in complex microservice architectures. The choice depends on the specific needs and the capabilities of the CNI version deployed with KKP.

7.2. Cloud-Native Security Platforms (CNAPP/CWPP)

Cloud-Native Application Protection Platforms (CNAPPs) and Cloud Workload Protection Platforms (CWPPs) offer a suite of security tools designed for containerized and cloud environments.

- **Examples:** Aqua Security, Sysdig, Qualys, Wiz, Palo Alto Networks Prisma Cloud.
- **Key Capabilities Relevant to PCI DSS:**

- **Vulnerability Management:** Thorough scanning of container images (in CI/CD, registries, and runtime), host OS, and Kubernetes configurations for known vulnerabilities and misconfigurations.[42] This supports PCI DSS Req 5, Req 6, and Req 11.3.
- **Runtime Protection:** Real-time threat detection, behavioral analysis (identifying anomalous process activity, network connections, or file access), drift prevention (blocking unauthorized changes to running containers), and malware detection.[3] This is essential for Req 5, Req 10, and Req 11.5.
- **Compliance Auditing and Reporting:** Many CNAPPs provide pre-built checks and reports mapped to compliance standards like CIS Benchmarks, NIST, and PCI DSS.[57] This assists with continuous compliance monitoring and evidence generation for audits.
- **File Integrity Monitoring (FIM):** Some CNAPPs include FIM capabilities to monitor critical files for unauthorized changes, supporting PCI DSS Req 10.3.4 and 11.5.2.89
- **Network Visibility and Microsegmentation:** Some platforms offer enhanced network traffic visibility and tools to help define or enforce microsegmentation policies.

- **KKP Integration:**
  - Aqua Security lists Kubermatic Kubernetes Platform among its supported platforms.[73]
  - Integration typically involves deploying agents (often as DaemonSets) on KKP User Cluster worker nodes and potentially integrating with the Kubernetes API server for visibility and enforcement.
  - The data collected by CNAPP agents is sent to a centralized management console for analysis, alerting, and reporting.

CNAPPs/CWPPs provide a consolidated approach to many security challenges in Kubernetes. For PCI DSS compliance on KKP, these platforms can significantly help address requirements related to vulnerability management, runtime threat detection, FIM, and compliance monitoring. The ability to automate checks against PCI DSS controls and generate compliance reports is particularly valuable.

7.3. Policy Enforcement Engines (OPA Gatekeeper, Kyverno)

Policy enforcement engines provide a way to implement custom admission control policies in Kubernetes, ensuring that configurations and deployments adhere to security and compliance standards.

- **Purpose:** To validate and/or mutate Kubernetes resource configurations before they are persisted in etcd, thereby preventing non-compliant or insecure deployments. This directly supports PCI DSS Req 2 (Apply Secure Configurations) and can help enforce aspects of other requirements.
- **KKP Support:** KKP integrates both OPA Gatekeeper and Kyverno as Kubernetes-native policy engines (Kyverno integration was added in KKP v2.28), allowing administrators to enable and

enforce policies during cluster creation or management.[1] KKP recommends OPA or Kyverno as the replacement for the deprecated PodSecurityPolicy, with Pod Security Admission covering pod-level security standards.[100]

- **Policy Examples for PCI DSS using OPA/Gatekeeper or Kyverno:**
  - **Pod Security Standards:** Enforce profiles equivalent to or stricter than PSA Baseline or Restricted (e.g., disallow privileged containers, restrict hostPath mounts, limit capabilities, require runAsNonRoot).[56]
  - **Image Provenance:** Allow images only from approved/trusted registries.[55]
  - **Resource Configuration:** Require specific labels or annotations for all resources deployed within CDE namespaces (e.g., for cost allocation, ownership, or CDE identification).
  - **NetworkPolicy Enforcement:** Mandate that every CDE namespace must have a default-deny NetworkPolicy.
  - **Ingress Configuration:** Restrict Ingress hostnames or require specific TLS configurations.
  - **Secret Management:** Prohibit storing sensitive data in ConfigMaps or environment variables directly, guiding towards approved secrets management practices.
- **OPA/Gatekeeper:** Uses Rego as its policy language. ConstraintTemplates define the policy logic, and Constraints apply these templates to specific resources.[55]
- **Kyverno:** Policies are defined as Kubernetes Custom Resources (CRs), which can be more intuitive for Kubernetes administrators. Kyverno can validate, mutate, generate, and verify (image signatures) resources.[56]

Policy engines are powerful tools for proactively enforcing security configurations in a KKP CDE. By defining policies as code, organizations can ensure consistency, automate compliance checks, and prevent misconfigurations that could lead to PCI DSS violations. KKP's native support for OPA Gatekeeper simplifies its adoption.

7.4. External Secrets Management Solutions (e.g., HashiCorp Vault)

While Kubernetes offers native Secrets objects, external secrets management solutions provide enhanced security features essential for protecting highly sensitive data in a PCI DSS CDE.

- **Benefits over Native Kubernetes Secrets:**
  - **Stronger Encryption:** Secrets are encrypted at rest within the vault using strong mechanisms, often with HSM integration for master key protection. Kubernetes Secrets are only Base64 encoded by default in etcd, relying on etcd encryption for true protection.[67]
  - **Centralized Management:** A single source of truth for secrets across multiple KKP clusters and applications.

- **Dynamic Secrets:** Ability to generate secrets on-demand with short lifespans (e.g., database credentials, cloud IAM tokens), reducing the risk of stale or compromised static credentials.[67]
- **Detailed Auditing:** Detailed audit trails of who accessed which secret and when, supporting PCI DSS Req 10.67
- **Fine-grained Access Control Lists (ACLs):** Granular policies controlling access to secrets based on application identity, Kubernetes service accounts, or user roles.[67]
- **Automated Rotation:** Many external vaults can automate the rotation of secrets.

- **Integration Methods with KKP User Clusters:**
  - **Secrets Store CSI Driver:** This is a widely adopted method. The driver allows Kubernetes to mount secrets stored in external vaults (like HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, GCP Secret Manager) as volumes directly into pods. The secrets are fetched by the driver and made available to the pod in-memory or as files, without being stored in etcd.[66]
  - **Vault Agent Injector (for HashiCorp Vault):** A Kubernetes mutating webhook admission controller that intercepts pod creation requests and automatically injects a Vault Agent sidecar container. The agent authenticates with Vault and retrieves secrets for the main application container.
- **PCI DSS Alignment:**
  - Securely managing cryptographic keys and sensitive configuration data directly supports PCI DSS Req 3.6 (Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data) and Req 3.7 (Restrict access to cryptographic keys to the fewest number of custodians necessary).[108]
  - The external vault itself, if managing secrets for the CDE, becomes a critical CDE component and must be hardened, monitored, and audited according to PCI DSS requirements.[108]

For a KKP environment handling CHD, using an external secrets management solution like HashiCorp Vault, integrated via the Secrets Store CSI Driver, is highly recommended. This approach significantly enhances the security of sensitive data compared to relying solely on native Kubernetes Secrets, even when etcd encryption is enabled, by keeping the secrets out of the Kubernetes data store and providing more strong management and auditing features.

8. Achieving and Maintaining Continuous Compliance on KKP

PCI DSS 4.0 emphasizes that security is an ongoing process extending well beyond any single point-in-time assessment.[5] For dynamic environments like Kubermatic Kubernetes Platform

(KKP), this means implementing practices and tools that enable ongoing monitoring, validation, and adaptation of security controls.

8.1. Infrastructure as Code (IaC) and GitOps for Compliant Deployments

Managing KKP configurations and CDE deployments through code is fundamental for achieving consistent, repeatable, and auditable compliance.

- **Infrastructure as Code (IaC) Principles:**
  - Define and manage all aspects of the KKP CDE infrastructure and configuration using machine-readable definition files. This includes:
    - KKP Master, Seed, and User Cluster configurations (potentially using KKP Cluster Templates 1 or tools like Terraform if KKP provider for Terraform is used).
    - Kubernetes resources within User Clusters: NetworkPolicies, RBAC Roles/RoleBindings, Pod Security Admission configurations, Ingress rules, ConfigMaps, Deployments, Services, etc..129
    - Underlying cloud infrastructure (VPCs, subnets, firewalls, IAM roles) if KKP is deployed on public cloud, often managed with Terraform, CloudFormation, or ARM Templates.[131]
  - **Tools:** Terraform is a common IaC tool for provisioning and managing infrastructure across multiple providers.[130] KKP itself offers Cluster Templates for standardizing User Cluster configurations.[1] Kubernetes manifests (YAML) are inherently IaC for Kubernetes resources.
- **Benefits of IaC for PCI DSS:**
  - **Consistency and Repeatability:** Ensures that CDE environments are deployed with standardized, secure configurations every time, reducing the risk of human error and configuration drift.[130]
  - **Auditability:** All infrastructure and configuration changes are version-controlled in a Git repository, providing a clear audit trail of who changed what, when, and why. This supports PCI DSS Req 10 and simplifies evidence gathering for auditors.[130]
  - **Automated Validation:** IaC configurations can be automatically validated against security and compliance policies before deployment (e.g., using tools like tfsec, checkov, OPA, or CNAPP scanning of IaC templates).
  - **Rapid Recovery:** In case of an incident or misconfiguration, the CDE can be quickly redeployed to a known-good state from code.
- **GitOps for KKP CDE:**
  - GitOps is an operational framework that uses Git as the single source of truth for declarative infrastructure and applications.[129] Changes to the desired state are made via

Git commits, and automated processes ensure the live environment converges to this desired state.

- **Tools:**
  - **Argo CD:** A declarative, GitOps continuous delivery tool for Kubernetes. KKP documentation provides guidance on integrating Argo CD for managing KKP and its components.[112]
  - **Flux CD:** Another popular CNCF GitOps tool that automates synchronization between Git repositories and Kubernetes clusters.[135]
- **Process:**
  - Define KKP CDE configurations (cluster specs, NetworkPolicies, RBAC, application manifests) in a Git repository.
  - Use a GitOps operator (Argo CD, Flux CD) running in the KKP Master or Seed/User Clusters to monitor the Git repository.
  - When changes are merged to the main branch, the GitOps operator automatically applies these changes to the respective KKP clusters, ensuring the live state matches the Git state.

- **Automated Drift Detection and Reconciliation:**
  - GitOps tools continuously monitor for differences (drift) between the desired state in Git and the actual state in the KKP clusters.[130]
  - They can automatically reconcile drift or alert administrators, ensuring that compliant configurations are maintained.

Adopting IaC and GitOps practices is highly beneficial for maintaining a PCI DSS compliant KKP environment. It embeds compliance into the deployment lifecycle, provides strong auditability, and helps manage the complexity of Kubernetes configurations at scale. The combination of KKP's templating features and GitOps tools like Argo CD offers a powerful approach to continuous compliance.

8.2. Automated Compliance Checks and Reporting

Automating compliance checks and generating reports are essential for demonstrating ongoing adherence to PCI DSS 4.0 requirements.

- **Tools for Automated Compliance Checks:**
  - **Kubernetes Security Posture Management (KSPM) tools:** These tools, often part of CNAPPs (e.g., Aqua Security, Sysdig, Wiz, Prisma Cloud), continuously scan KKP User Clusters against security benchmarks (like CIS Benchmarks for Kubernetes) and compliance standards (including mappings to PCI DSS controls).[57]

- **Policy Engines (OPA/Gatekeeper, Kyverno):** While primarily for enforcement, these tools also provide audit capabilities. OPA Gatekeeper's audit function periodically evaluates existing resources against defined constraints and reports violations.[125] Kyverno policies can also be set to audit mode. KKP's OPA Gatekeeper integration can be used for this.[1]
- **Configuration Validation Tools for IaC:** Tools like tfsec, checkov, Terrascan, and KICS can scan Terraform and Kubernetes YAML files for misconfigurations and compliance violations before deployment.
- **Vulnerability Scanners:** Continuous scanning of images and running workloads for vulnerabilities (see Section 6.2).
- **Custom Scripts and Tools:** Organizations can develop custom scripts to check for specific PCI DSS configurations within their KKP environment, potentially querying the KKP API or Kubernetes APIs.

- **Generating Compliance Reports:**
  - Many CNAPPs and KSPM tools can generate on-demand or scheduled compliance reports, often mapping findings directly to PCI DSS requirements or other frameworks.[57] These reports are invaluable for internal reviews and providing evidence to QSAs.
  - SIEM systems can also be configured to generate reports based on log data and security events relevant to PCI DSS compliance.[88]
- **Integrating with CI/CD Pipelines:**
  - Embed automated compliance checks directly into CI/CD pipelines. For example, fail a build or deployment if IaC scans reveal critical misconfigurations, if image scans find high-risk vulnerabilities, or if proposed Kubernetes manifests violate OPA/Gatekeeper policies.[46] This "shifts left" compliance, catching issues early.
- **Continuous Monitoring and Alerting:**
  - Beyond periodic checks, implement continuous monitoring with real-time alerting for compliance deviations or security events that could impact PCI DSS status (see Section 6.1).

Automation is key to managing compliance in a dynamic KKP environment. Manual checks are prone to error and cannot keep pace with changes. By using automated tools for configuration validation, vulnerability scanning, policy auditing, and report generation, organizations can maintain a more consistent compliance posture and streamline audit preparations.

8.3. Regular Audits, Penetration Testing, and Reviews

PCI DSS mandates regular testing and review of security controls to ensure their ongoing effectiveness.

- **Internal and External Audits:**

- Conduct regular internal audits to assess compliance with PCI DSS requirements within the KKP CDE.
- Engage with a Qualified Security Assessor (QSA) for formal annual PCI DSS assessments if required (e.g., for Level 1 merchants or service providers).

- **Penetration Testing (Req 11.4):**
  - Perform external and internal penetration testing at least annually and after any significant changes to the KKP CDE or applications.[93]
  - Testing should cover the KKP infrastructure (if in scope and accessible), network segmentation controls, User Cluster components, and CDE applications.
  - Use a methodology that includes both network-level and application-level testing.
- **Vulnerability Scanning (Req 11.3):**
  - Quarterly external vulnerability scans by an Approved Scanning Vendor (ASV).[93]
  - Quarterly internal vulnerability scans. PCI DSS 4.0 Req 11.3.1.2 now mandates these internal scans be *authenticated*.16
  - Remediate identified vulnerabilities based on risk ranking.
- **Review of Security Controls and Policies:**
  - **Network Security Controls (Req 1):** Regularly review firewall and router rule sets (at least every six months per some interpretations, or as per targeted risk analysis).[37]
  - **Access Controls (Req 7 & 8):** Periodically review user access rights and authentication mechanisms.[9]
  - **Log Reviews (Req 10):** Daily review of security logs for critical systems, and periodic reviews for others.[78]
  - **Incident Response Plan (Req 12.10):** Test the IRP at least annually.[18]
  - **Cryptographic Standards (Req 3.3.2, 4.2.1.1 in PCI DSS 4.0):** Annually review cryptographic algorithms and protocols in use to ensure they remain strong and are not deprecated.[6]
  - **Security Policies (Req 12.1):** Review and update information security policies at least annually.[18]
  - **Third-Party Service Provider (TPSP) Compliance (Req 12.8, 12.9):** Annually review the PCI DSS compliance of TPSPs.[4]
- **Targeted Risk Analyses (New in PCI DSS 4.0):**
  - For requirements where the customized approach is used, or where frequencies of controls can be defined by the entity, targeted risk analyses must be performed to justify the approach and reviewed annually (Req 12.3.1, 12.3.2).[5]

Continuous compliance is an iterative cycle of assessment, remediation, and monitoring. Regular audits, penetration tests, and reviews are critical feedback mechanisms to ensure that security controls within the KKP CDE remain effective against evolving threats and that the organization maintains its PCI DSS compliance posture.

9. Appendix

9.1. PCI DSS 4.0 Requirements Checklist for KKP Environments

This checklist provides a high-level overview of key considerations for each of the 12 PCI DSS 4.0 requirements when using Kubermatic Kubernetes Platform (KKP). It is not exhaustive but serves as a starting point for a detailed gap analysis. Organizations should refer to the official PCI DSS 4.0 standard for full requirement details.[139]

Table 2: PCI DSS 4.0 Requirements Checklist for KKP

Requirement #	Goal	Summary of Requirement & Key KKP Considerations
1	Build and Maintain a Secure Network and Systems	**Install and Maintain Network Security Controls.** - Document network topology of KKP CDE (Master, Seed, User Clusters). - Implement firewalls/NSGs at infrastructure level. - Configure Kubernetes NetworkPolicies (e.g., via Cilium/Calico in KKP User Clusters) for CDE segmentation (default deny, allow specific). - Secure KKP API server access (allowed IP ranges). - Regularly review and update NSC rules. 4
2	Build and Maintain a Secure Network and Systems	**Apply Secure Configurations to All System Components.** - Harden KKP Master/Seed/User Cluster components (OS, Kubernetes services). - Change all vendor-default credentials and settings. - Use KKP OPA Gatekeeper integration or PSA for secure pod configurations in CDE. - Maintain inventory of CDE system components. - Develop configuration standards (e.g., based on CIS Benchmarks). 51
3	Protect Account Data	**Protect Stored Account Data.** - Minimize storage of CHD; define data retention and disposal policies. - Render PAN unreadable (encryption, truncation, tokenization) if stored in PVs or databases within KKP CDE. - Enable etcd encryption for Kubernetes Secrets containing sensitive data. - Implement strong key management for encryption keys (KMS, HSM). - Prohibit storage of SAD after authorization. 60
4	Protect Account Data	**Protect Cardholder Data with Strong Cryptography During Transmission.** - Use TLS 1.2+ for all CHD transmission over open/public networks (e.g., KKP UI, application frontends, API calls to payment gateways). - Implement mTLS (e.g., via service mesh or Cilium) for inter-service communication within KKP CDE. - Securely manage SSL/TLS certificates and keys. - Disable weak/deprecated cryptographic protocols. 68
5	Maintain a Vulnerability Management Program	**Protect All Systems and Networks from Malicious Software.** - Deploy and maintain anti-malware solutions on KKP worker nodes and other in-scope systems. - For containers, focus on image scanning for malware and runtime threat detection. - Regularly update anti-malware software and signatures. - Perform periodic scans. 70
6	Maintain a Vulnerability	**Develop and Maintain Secure Systems and Software.**

Requirement #	Goal	Summary of Requirement & Key KKP Considerations
	Management Program	- Implement a process to identify and risk-rank vulnerabilities in KKP components (if self-hosted), OS, Kubernetes, container images, and applications. - Apply security patches promptly. - Follow secure coding practices for applications in CDE. - Deploy WAF for public-facing web applications in CDE (Req 6.4.2). - Manage payment page scripts securely (inventory, authorization, integrity - Req 6.4.3). 12
7	Implement Strong Access Control Measures	**Restrict Access to System Components and Cardholder Data by Business Need to Know.** - Implement principle of least privilege using KKP user management and Kubernetes RBAC. - Define access needs for each role accessing KKP CDE or management interfaces. - Default "deny-all" access. - Regularly review access controls. 48
8	Implement Strong Access Control Measures	**Identify Users and Authenticate Access to System Components.** - Assign unique IDs for all users accessing KKP and CDE components. - Enforce MFA for ALL access into the CDE (KKP UI/API, kubectl, SSH, etc.) via KKP OIDC integration with MFA-capable IdP (Req 8.4.2). - Implement strong password/passphrase policies (12 characters min by Mar 2025 - Req 8.3.6). - Secure application and system accounts; no shared/generic accounts without justification and controls. 6
9	Implement Strong Access Control Measures	**Restrict Physical Access to Cardholder Data.** - For on-prem KKP, secure data centers housing CDE infrastructure. - For cloud KKP, rely on CSP physical security (verify AoC). - Control physical access to KKP console terminals or devices used to access CDE. - Securely manage and destroy media containing CHD. 77
10	Regularly Monitor and Test Networks	**Log and Monitor All Access to System Components and Cardholder Data.** - Enable thorough logging (KKP MLA, Kubernetes audit logs, OS, application, network) for CDE components. - Centralize logs in a SIEM. - Protect logs from tampering; retain for 1 year (90 days readily available). - Implement daily (or risk-based frequency via targeted risk analysis) log reviews. - Use FIM/change-detection for audit logs and critical files (Req 10.3.4, 11.5.2). - Synchronize time (NTP, Req 10.6). 78
11	Regularly Monitor and Test Networks	**Test Security of Systems and Networks Regularly.** - Conduct quarterly internal (authenticated - Req 11.3.1.2) and external (ASV) vulnerability scans. - Perform annual (or more frequent) internal/external penetration testing. - Deploy IDS/IPS (Req 11.5.1). - Deploy FIM (Req 11.5.2). - Implement change-and-tamper detection for payment pages/HTTP headers (Req 11.6.1). 15
12	Maintain an Information Security Policy	**Support Information Security with Organizational Policies and Programs.** - Develop, publish, and maintain a thorough information security policy. - Implement a formal security awareness program. - Conduct targeted risk analyses for applicable requirements (e.g., customized approach, control frequencies). - Manage TPSP risks (due diligence, contracts, monitoring). - Maintain and test an incident response plan annually. - Define clear roles and responsibilities for PCI DSS compliance. 18

9.2. Glossary of Terms

- **AES (Advanced Encryption Standard):** A symmetric block cipher widely used for data encryption.[141]
- **API (Application Programming Interface):** A set of rules and protocols for building and interacting with software applications.[141]
- **ASV (Approved Scanning Vendor):** A company approved by the PCI SSC to conduct external vulnerability scanning services.[143]
- **Authentication:** The process of verifying the identity of a user, process, or device.[141]
- **Authorization:** The process of granting or denying access rights to resources.
- **Cardholder Data (CHD):** At a minimum, consists of the full Primary Account Number (PAN). Cardholder data may also appear in the form of the full PAN plus any of the following: cardholder name, expiration date, and/or service code.[141]
- **Cardholder Data Environment (CDE):** The people, processes, and technology that store, process, or transmit cardholder data or sensitive authentication data, including any system components that may not store, process, or transmit CHD/SAD but have unrestricted connectivity to or could impact the security of CDE components.[4]
- **CI/CD (Continuous Integration/Continuous Delivery or Deployment):** Practices that automate the building, testing, and deployment of software.
- **CIS Benchmarks:** Consensus-based configuration guidelines for various technology groups to safeguard systems against cyber threats.[46]
- **CNAPP (Cloud-Native Application Protection Platform):** A unified security platform that consolidates multiple cloud security tools and capabilities.
- **CNI (Container Network Interface):** A CNCF specification for configuring network interfaces for Linux containers.[32]
- **Compensating Controls:** Alternative controls that can be considered when an entity cannot meet a requirement explicitly as stated, due to legitimate technical or documented business constraints, but has sufficiently mitigated the risk associated with the requirement.[5]
- **Container:** A standardized unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.[144]
- **CRI (Container Runtime Interface):** A plugin interface that enables kubelet to use a wide variety of container runtimes.
- **CSI (Container Storage Interface):** A standard for exposing block and file storage systems to containerized workloads on COs like Kubernetes.

- **Customized Approach:** A new approach in PCI DSS 4.0 that allows entities to meet the objective of a requirement using security controls different from those defined in the standard, provided they are supported by a targeted risk analysis and thorough documentation.[5]
- **CVSS (Common Vulnerability Scoring System):** An open framework for communicating the characteristics and severity of software vulnerabilities.[113]
- **CWPP (Cloud Workload Protection Platform):** Security solutions designed to protect server workloads in hybrid and multi-cloud data center environments.
- **DMZ (Demilitarized Zone):** A perimeter network that protects an organization's internal local-area network (LAN) from untrusted traffic.[94]
- **DNS (Domain Name System):** A hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network.
- **eBPF (extended Berkeley Packet Filter):** A technology that can run sandboxed programs in an operating system kernel, used by tools like Cilium for networking, security, and observability.[25]
- **etcd:** A consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.[3]
- **FIM (File Integrity Monitoring):** Technology that monitors and detects changes in files that may indicate a cyberattack.[15]
- **Firewall:** A network security device that monitors and filters incoming and outgoing network traffic based on an organization's previously established security policies.[141]
- **GitOps:** An operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.[112]
- **HSM (Hardware Security Module):** A physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing.[66]
- **IaC (Infrastructure as Code):** The practice of managing and provisioning infrastructure through machine-readable, version-controlled definition files that replace manual configuration.[129]
- **IAM (Identity and Access Management):** A framework of policies and technologies for ensuring that the right users have the appropriate access to technology resources.[6]
- **IDS/IPS (Intrusion Detection System/Intrusion Prevention System):** Devices or software applications that monitor a network or systems for malicious activity or policy violations.[37]
- **Ingress Controller:** A component in Kubernetes that manages external access to services in a cluster, typically HTTP.
- **KKP (Kubermatic Kubernetes Platform):** A Kubernetes management platform for automating operations of Kubernetes clusters.

- **KMS (Key Management Service):** A managed service that makes it easy for you to create and control the encryption keys used to encrypt your data.[61]
- **Kubernetes:** An open-source container orchestration system for automating software deployment, scaling, and management.[142]
- **Kubelet:** An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.[144]
- **Loki:** A horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus.[1]
- **Master Cluster (KKP):** The central KKP cluster managing users, projects, and Seed Clusters.[19]
- **MFA (Multi-Factor Authentication):** A security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction.[6]
- **mTLS (Mutual TLS):** A process where both the client and server in a communication authenticate each other using TLS certificates.[33]
- **Namespace:** A Kubernetes feature to partition cluster resources between multiple users or teams.[47]
- **NetworkPolicy (Kubernetes):** A specification of how groups of pods are allowed to communicate with each other and other network endpoints.[3]
- **NSC (Network Security Controls):** A broader term in PCI DSS 4.0 replacing "firewalls, " encompassing various technologies that protect network traffic.[4]
- **OIDC (OpenID Connect):** An identity layer on top of the OAuth 2.0 protocol, allowing clients to verify the identity of an end-user based on authentication performed by an authorization server.[1]
- **OPA (Open Policy Agent):** An open-source, general-purpose policy engine that enables unified, context-aware policy enforcement.[3]
- **PAN (Primary Account Number):** The unique payment card number that identifies the issuer and the particular cardholder account.[141]
- **PCI DSS (Payment Card Industry Data Security Standard):** A set of security standards designed to ensure that all companies that accept, process, store or transmit credit card information maintain a secure environment.[4]
- **Pod (Kubernetes):** The smallest deployable units of computing that can be created and managed in Kubernetes; a group of one or more containers.[144]
- **Prometheus:** An open-source systems monitoring and alerting toolkit.[3]
- **PSA (Pod Security Admission):** A Kubernetes admission controller that enforces Pod Security Standards at the namespace level.[53]

- **PSP (PodSecurityPolicy):** A deprecated Kubernetes feature that allowed fine-grained authorization of pod creation and updates. Replaced by PSA and policy engines like OPA/Gatekeeper or Kyverno.[3]
- **PV (Persistent Volume):** A piece of storage in the Kubernetes cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.[106]
- **QSA (Qualified Security Assessor):** A company authorized by the PCI SSC to validate an entity's adherence to PCI DSS.
- **RBAC (Role-Based Access Control):** A method of restricting network access based on the roles of individual users within an enterprise.[3]
- **SAD (Sensitive Authentication Data):** Security-related information used to authenticate cardholders and/or authorize payment card transactions. Includes full track data, card validation codes/values (CVV2, CVC2, CID, CAV2), and PINs/PIN blocks. Must not be stored after authorization.[11]
- **SAQ (Self-Assessment Questionnaire):** A validation tool for merchants and service providers to self-evaluate their PCI DSS compliance.[5]
- **Seed Cluster (KKP):** A KKP cluster that hosts the control planes of User Clusters.[1]
- **Service Mesh:** A dedicated infrastructure layer for handling service-to-service communication, often providing features like mTLS, traffic management, and observability.[33]
- **SIEM (Security Information and Event Management):** Software solutions that aggregate and analyze activity from many different resources across an entire IT infrastructure.[3]
- **SRI (Subresource Integrity):** A security feature that enables browsers to verify that resources they fetch (for example, from a CDN) are delivered without unexpected manipulation.[15]
- **SSDLC (Secure Software Development Lifecycle):** A process that embeds security practices into each phase of the software development lifecycle.[14]
- **Terraform:** An open-source infrastructure as code software tool.[130]
- **TLS (Transport Layer Security):** A cryptographic protocol designed to provide communications security over a computer network.[3]
- **Tokenization:** A process by which a sensitive data element (e.g., PAN) is replaced with a non-sensitive equivalent, referred to as a "token, " that has no extrinsic or exploitable meaning or value.[6]
- **TPSP (Third-Party Service Provider):** An entity that is directly involved in the processing, storage, or transmission of cardholder data on behalf of another entity, or that could impact the security of the CDE.[4]
- **User Cluster (KKP):** Kubernetes clusters managed by KKP where end-user applications are deployed.[1]

- **Velero:** An open-source tool to safely backup and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes.[28]
- **WAF (Web Application Firewall):** A firewall that monitors, filters, or blocks HTTP traffic to and from a web application, designed to protect against web-based attacks.[6]

9.3. References and Further Reading

- PCI Security Standards Council Document Library: https://www.pcisecuritystandards.org/document_library/ 139 (Specifically PCI DSS v4.0.1 Requirements and Testing Procedures 139)
- Kubermatic Kubernetes Platform Documentation (v2.30): https://docs.kubermatic.com/kubermatic/v2.30/
- Kubermatic Website: https://www.kubermatic.com/
- Kubernetes Documentation: https://kubernetes.io/docs/
- CIS Benchmarks: https://www.cisecurity.org/cis-benchmarks/
- Open Policy Agent (OPA) Documentation: https://www.openpolicyagent.org/docs/latest/
- Kyverno Documentation: https://kyverno.io/docs/
- Cilium Documentation: https://docs.cilium.io/
- Istio Documentation: https://istio.io/latest/docs/
- HashiCorp Vault Documentation: https://developer.hashicorp.com/vault/docs
- Secrets Store CSI Driver Documentation: https://secrets-store-csi-driver.sigs.k8s.io/

Specific articles and blog posts referenced throughout this guide are cited in-line using their respective snippet IDs.

10. Conclusion

Achieving and maintaining PCI DSS 4.0 compliance within a Kubermatic Kubernetes Platform (KKP) environment is a multifaceted undertaking that requires a deep understanding of both KKP's architecture and the intricacies of the PCI DSS standard. As of 2026, with PCI DSS v4.0.1 the active standard and all requirements (including the formerly future-dated ones) fully in effect, organizations must adopt a continuous, defense-in-depth security posture.

KKP provides a strong foundation for managing Kubernetes clusters at scale, offering features like centralized user management with OIDC integration, a built-in monitoring and logging stack, OPA Gatekeeper integration for policy enforcement, and automated cluster lifecycle management. These capabilities can be instrumental in addressing various PCI DSS 4.0 requirements, particularly those related to secure configurations, access control, and operational monitoring. The platform's architecture, with its Master, Seed, and User Cluster distinctions, allows for logical and

network segmentation strategies essential for defining and isolating the Cardholder Data Environment (CDE).

However, KKP alone does not guarantee PCI DSS compliance. The shared responsibility model is paramount; organizations must meticulously configure KKP, secure the underlying infrastructure (whether on-premises or cloud-based), and harden all components within the CDE, including worker nodes and containerized applications. This guide has detailed specific hardening measures for KKP control plane elements, User Clusters, container images, network configurations, API servers, etcd data stores, secrets management, and persistent volumes.

The evolving requirements of PCI DSS 4.0, with its emphasis on MFA for all CDE access, mandatory WAFs for public web applications, stringent management of payment page scripts, and authenticated internal vulnerability scanning, necessitates a proactive and adaptive security strategy. Using KKP's integrations with third-party security solutions—such as service meshes (Istio) for mTLS and fine-grained authorization, CNAPPs/CWPPs (Aqua Security, Sysdig) for thorough vulnerability management and runtime protection, external secrets managers (HashiCorp Vault) for enhanced data protection, and advanced CNI capabilities (Cilium)—is often essential to meet these advanced requirements.

Continuous compliance, facilitated by Infrastructure as Code (IaC), GitOps practices, automated compliance checks, and regular audits, is critical. KKP's support for cluster templating and its compatibility with GitOps tools like ArgoCD can significantly aid in maintaining a consistent and verifiable compliant state.

Ultimately, organizations must conduct a thorough gap analysis against PCI DSS 4.0, tailor the recommendations in this guide to their specific KKP deployment model and CDE scope, and implement a holistic security program. The "Customized Approach" offered by PCI DSS 4.0 provides flexibility for innovative solutions in Kubernetes environments but demands rigorous targeted risk analysis and documentation. By combining KKP's strengths with diligent security practices and appropriate third-party tools, organizations can build and operate a PCI DSS 4.0 compliant environment, safeguarding cardholder data effectively.

Works cited

1. What is Kubermatic Kubernetes Platform (KKP)?, https://docs.kubermatic.com/kubermatic/v2.30/
2. Kubermatic: Multi Cloud Kubernetes Management Platform, https://www.kubermatic.com/
3. Kubernetes Compliance: NIST, CIS & PCI- Actionable Guide | Pomerium, https://www.pomerium.com/blog/kubernetes-compliance-nist-cis-pci-actionable-guide
4. Kubernetes Compliance: PCI DSS Pointers for K8s - 360 Cloud Platforms, https://360cloudplatforms.com/blog/kubernetes-compliance-pci-dss-pointers-for-k8
5. What's New in PCI DSS 4.0? Key Updates Explained - Secureframe, https://secureframe.com/blog/pci-dss-4.0
6. Preparing for PCI DSS 4.0 Compliance in 2025 - Thales, https://cpl.thalesgroup.com/blog/encryption/pci-dss-4-0-compliance-2025
7. Compliance & Security Prioritized: A cloud-based approach to PCI DSS v4 migration - Deloitte, https://www2.deloitte.com/content/dam/Deloitte/us/Documents/consulting/us-compliance-security-prioritized-v2.pdf
8. PCI DSS Scoping and Segmentation for Modern Network Architectures - Bright Defense, https://www.brightdefense.com/resources/pci-dss-modern-network-architectures/
9. PCI DSS 4.0 Requirement 8: How to Identify Users and Authenticate Access to System Components - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-8-how-to-identify-users-and-authenticate-access-to-system-components
10. Make Your AWS Environment PCI DSS v4.0.1 Compliant with F5, https://www.f5.com/company/blog/make-your-aws-environment-pci-dss-v4-0-1-compliant-with-f5
11. Data Privacy & Cybersecurity in 2025: PCI DSS 4.0, https://www.mwe.com/insights/data-privacy-and-cybersecurity-in-2025-pci-dss-4-0/
12. PCI DSS 4.0 Compliance Deadline: Why WAF is Mandatory Under Requirement 6.4.2, https://www.sitewall.net/pci-dss-4-0-compliance-deadline-why-waf-is-mandatory-under-requirement-6-4-2/
13. PCI DSS v4.0.1: The Changes You Need to Know to Qualify for SAQ ..., https://www.akamai.com/blog/security/pci-dss-v4-0-1-changes-qualify-saq-a
14. PCI DSS 4.0 Requirement 6: How to Develop and Maintain Secure Systems and Software, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-6-how-to-develop-and-maintain-secure-systems-and-software

15. PCI DSS 4.0.1: A Comprehensive Guide to Successfully Meeting Requirements 6.4.3 and 11.6.1 - Feroot Security, https://www.feroot.com/blog/pci-dss-4-0-requirement-6-4-3-and-11-6-1/

16. Explanation of New Authenticated Scanning PCI DSS Requirement 11.3.1.2 in PCI DSS V4.0 and how InsightVM can help meet the Requirement - Rapid7, https://www.rapid7.com/blog/post/2024/02/20/explanation-of-new-authenticated-scanning-pci-dss-requirement-11-3-1-2-in-pci-dss-v4-0-and-how-insightvm-can-help-meet-the-requirement/

17. Managing the Overload of Vulnerabilities in PCI DSS 4.0.1 Authenticated Scans req - Reddit, https://www.reddit.com/r/pcicompliance/comments/1idt91l/managing_the_overload_of_vulnerabilities_in_pci/

18. PCI DSS 4.0 Requirement 12: How to Support Information Security with Organizational Policies and Programs - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-12-how-to-support-information-security-with-organizational-policies-and-programs

19. Add Seed Cluster to Kubermatic Kubernetes Platform (KKP) CE, https://docs.kubermatic.com/kubermatic/v2.30/installation/install-kkp-ce/add-seed-cluster/

20. Installation - KKP Documentation - Docs - Kubermatic Documentation, https://docs.kubermatic.com/kubermatic/v2.30/installation/

21. Kubermatic Kubernetes Platform | Kubermatic, https://www.kubermatic.com/products/kubermatic-kubernetes-platform/

22. What is the Shared Responsibility Model? - ARMO, https://www.armosec.io/glossary/shared-responsibility-model/

23. Shared Responsibility Model - Amazon Web Services (AWS), https://aws.amazon.com/compliance/shared-responsibility-model/

24. PCI Compliance for Containers and the Cloud - Sysdig, https://sysdig.com/learn-cloud-native/pci-compliance-for-containers-and-the-cloud/

25. Isovalent Enterprise for Cilium: Observability, Security, Networking, https://isovalent.com/product/

26. *(Removed)*

27. API Server Access Control - Docs - Kubermatic Documentation, https://docs.kubermatic.com/kubermatic/v2.30/tutorials-howtos/networking/apiserver-policies/

28. Automated Kubernetes Backups with KKP | Kubermatic, https://www.kubermatic.com/products/kubermatic-kubernetes-platform/backups/

29. Managed Kubermatic Kubernetes Platform, https://www.kubermatic.com/products/managed-kubermatic-kubernetes-platform/

30. DATA PROCESSING AGREEMENT - Kubermatic, https://www.kubermatic.com/static/terms/Kubermatic_DPA_V202401.pdf

31. PCI DSS - Compliance - Google Cloud, https://cloud.google.com/security/compliance/pci-dss

32. The Four Cs of Cloud-Native Security, https://cloudnativenow.com/features/the-four-cs-of-cloud-native-security/

33. Istio for PCI Compliance: Implementing PCI DSS 4.0.1 with Service Mesh Security - Tetrate, https://tetrate.io/blog/istio-for-pci-compliance-implementing-pci-dss-4-0-1-with-service-mesh-security/

34. PCI Compliance Network Segmentation: A Guide - FireMon, https://www.firemon.com/blog/pci-compliance-network-segmentation/

35. 10 Steps to Ensure PCI DSS-Compliant Container Deployment - Cloud Native Now, https://cloudnativenow.com/topics/cloudnativesecurity/10-steps-to-ensure-pci-dss-compliant-container-deployment/

36. 7 Steps to Implementing Network Segmentation for PCI DSS Compliance - Tigera.io, https://www.tigera.io/learn/guides/microsegmentation/network-segmentation-pci-dss/

37. PCI DSS 4.0 Requirement 1: How to Install and Maintain Network Security Controls, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-1-how-to-install-and-maintain-network-security-controls

38. AKS regulated cluster for PCI-DSS 3.2.1 - Network segmentation - Azure Architecture Center, https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/containers/aks-pci/aks-pci-network

39. Architecture of an AKS regulated cluster for PCI-DSS 3.2.1 (Part 2 of 9) - Learn Microsoft, https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/containers/aks-pci/aks-pci-ra-code-assets

40. Getting Started with PCI for Kubernetes - RAD Security, https://blog.rad.security/blog/getting-started-with-pci-for-kubernetes

41. Use PCI-DSS v3.2.1 policy constraints | Policy Controller - Google Cloud, https://cloud.google.com/kubernetes-engine/enterprise/policy-controller/docs/how-to/using-pci-dss-v3

42. Kubernetes Security Best Practices, https://www.mirantis.com/blog/kubernetes-security-best-practices/

43. The cni-cilium module - deckhouse.io, https://deckhouse.io/products/kubernetes-platform/documentation/v1/modules/cni-cilium/

44. Overview of Network Policy — Cilium 1.17.3 documentation, https://docs.cilium.io/en/stable/security/policy/index.html

45. Bridging Kubernetes, Cilium & eBPF to Networking Concepts for Network Engineers: Part 2 Security - WWT, https://www.wwt.com/blog/bridging-kubernetes-cilium-and-ebpf-to-networking-concepts-for-network-engineers-part-2-security?utm_source=social&utm_medium=linkedin&utm_campaign=platform_share

46. A Practical Guide to the Different Compliance Kubernetes Security Frameworks and How They Fit Together, https://cloudsecurityalliance.org/articles/a-practical-guide-to-the-different-compliance-kubernetes-security-frameworks-and-how-they-fit-together

47. A Quick Refresher on Kubernetes Security Best Practices - Wiz, https://www.wiz.io/academy/kubernetes-security-best-practices

48. PCI DSS 4.0 Requirement 7: How to Restrict Access to System Components and Cardholder Data by Business Need to Know - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-7-how-to-restrict-access-to-system-components-and-cardholder-data-by-business-need-to-know

49. The Practitioner's Guide to Kubernetes Security - Annoyed Engineer, https://annoyed.engineer/2025/01/19/the-practitioners-guide-to-kubernetes-security/

50. Kubernetes Security Best Practices | Kubermatic, https://www.kubermatic.com/blog/kubernetes-security-best-practices/

51. PCI DSS 4.0 Requirement 2: How to Apply Secure Configurations to All System Components - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-2-how-to-apply-secure-configurations-to-all-system-components

52. Five Best Practices for PCI DSS Compliance in the Cloud - Orca Security, https://orca.security/resources/blog/5-best-practices-pci-dss-compliance-cloud/

53. Chapter 9: Pod Security Admission - Kubernetes Guides - Apptio, https://www.apptio.com/topics/kubernetes/best-practices/pod-security-admission/

54. Kubernetes Security - OWASP Cheat Sheet Series, https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html

55. OPA Gatekeeper Bypass Reveals Risks in Kubernetes Policy Engines - Aqua Security, https://www.aquasec.com/blog/risks-misconfigured-kubernetes-policy-engines-opa-gatekeeper/

56. DevSecOps for Kubernetes - Wiz, https://www.wiz.io/academy/devsecops-for-kubernetes

57. Compliance | Sysdig Docs, https://docs.sysdig.com/en/sysdig-secure/compliance/

58. Qualys PCI DSS 4.0 Compliance Whitepaper, https://www.qualys.com/whitepapers/qualys-pci-dss-4-0-compliance-whitepaper/

59. Vulnerability Management: Definition, Process, and Tools - Aqua Security, https://www.aquasec.com/cloud-native-academy/vulnerability-management/vulnerability-management/

60. PCI DSS 4.0 Requirement 3: How to Protect Stored Account Data - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-3-how-to-protect-stored-account-data

61. Default envelope encryption for all Kubernetes API Data - Amazon EKS, https://docs.aws.amazon.com/eks/latest/userguide/envelope-encryption.html

62. Encrypting etcd data | Security and compliance | OKD 4.18, https://docs.okd.io/4.18/security/encrypting-etcd.html

63. Encrypting etcd data | Security and compliance | OpenShift Container Platform 4.16, https://docs.openshift.com/container-platform/4.16/security/encrypting-etcd.html

64. Encrypting etcd data | Security and compliance | OpenShift Container Platform 4.12, https://docs.openshift.com/container-platform/4.12/security/encrypting-etcd.html

65. Encrypting Confidential Data at Rest - Kubernetes, https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

66. Protecting Secrets in Kubernetes using Fortanix CSI Provider, https://www.fortanix.com/blog/protecting-secrets-in-kubernetes-using-fortanix-csi-provider

67. How To Centralize Kubernetes Secrets Management With Vault - Firefly, https://www.firefly.ai/blog/how-to-centralize-kubernetes-secrets-management-with-vault

68. PCI DSS 4.0 Requirement 4: How to Protect Cardholder Data in Transit - HeroDevs Blog, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-4-how-to-protect-cardholder-data-in-transit

69. The Kubernetes API, https://kubernetes.io/docs/concepts/overview/kubernetes-api/

70. PCI Compliance: Merchant Levels, 12 Requirements & PCI in the Cloud - Tigera.io, https://www.tigera.io/learn/guides/pci-compliance/

71. PCI DSS Compliance Requirements and Tech that Can Help - Aqua Security, https://www.aquasec.com/cloud-native-academy/cloud-compliance/pci-dss-compliance/

72. Cloud workload protection in Runtime - Aqua Security, https://www.aquasec.com/products/cwpp-cloud-workload-protection/

73. Integrations - Aqua Security, https://www.aquasec.com/integrations/

74. Making PCI DSS Compliance Cloud-Native | Schellman, https://www.schellman.com/blog/pci-compliance/making-pci-compliance-cloud-native

75. Understanding New PCI Guidance on MFA, https://blog.pcisecuritystandards.org/understanding-new-pci-guidance-on-mfa

76. "Information Supplement" on multi-factor authentication - PCI Security Standards Council, https://www.pcisecuritystandards.org/pdfs/Multi-Factor-Authentication-Guidance-v1.pdf

77. PCI DSS – Requirement 9 – Restrict Physical Access to Cardholder Data - ISMS.online, https://www.isms.online/pci-dss/requirement-9/

78. PCI DSS 4.0 Requirement 10: How to Log and Monitor All Access to System Components and Cardholder Data - HeroDevs, https://www.herodevs.com/blog-posts/pci-dss-4-0-requirement-10-how-to-log-and-monitor-all-access-to-system-components-and-cardholder-data

79. Example stack - kdb products, https://code.kx.com/insights/1.13/enterprise/monitoring/example-monitoring-stack.html

80. Alert Management with Prometheus Alertmanager and Log Insights with Grafana Loki - Civo.com, https://www.civo.com/learn/advanced-kubernetes-monitoring

81. Chapter 9: Grafana Loki - Kubernetes Guides - Apptio, https://www.apptio.com/topics/kubernetes/monitoring/grafana-loki/

82. Grafana Loki OSS | Log aggregation system, https://grafana.com/oss/loki/

83. How To Monitor Kubernetes Audit Logs with LogRhythm SIEM - Exabeam, https://www.exabeam.com/blog/security-operations-center/how-to-monitor-kubernetes-audit-logs-with-logrhythm-siem/

84. Kubernetes Compliance: Frameworks & 5 Critical Best Practices - Tigera.io, https://www.tigera.io/learn/guides/kubernetes-security/kubernetes-compliance/

85. SIEM for Containerized Environments - Cloud Native Now, https://cloudnativenow.com/features/siem-for-containerized-environments/

86. Kubernetes Audit Logs - Datadog Docs, https://docs.datadoghq.com/integrations/kubernetes_audit_logs/

87. SIEM and security monitoring for Kubernetes explained - LevelBlue, https://levelblue.com/blogs/labs-research/security-monitoring-for-managed-cloud-kubernetes

88. PCI DSS Compliance Software - PCI Audit Trail Tools - SolarWinds, https://www.solarwinds.com/security-event-manager/use-cases/pci-dss-compliance-tool

89. Kubernetes Security for Google Cloud Platform - Aqua, https://www.aquasec.com/solutions/google-cloud-kubernetes-security/

90. Cloud VM Security with Aqua CSP, https://www.aquasec.com/blog/secure-vm-cloud-native-security/

91. What logs to retain for PCI-DSS? - Information Security Stack Exchange, https://security.stackexchange.com/questions/11024/what-logs-to-retain-for-pci-dss

92. What Are the PCI Audit Log Retention Requirements? - ZenGRC, https://www.zengrc.com/blog/what-are-the-pci-audit-log-retention-requirements/

93. PCI DSS – Requirement 11 – Regularly Test Security Systems and Processes - ISMS.online, https://www.isms.online/pci-dss/requirement-11/

94. PCI Compliance for Kubernetes | Atlantic.Net, https://www.atlantic.net/pci-compliant-hosting/pci-compliance-for-kubernetes/

95. A Framework for Kubernetes Incident Response | Kubermatic, https://www.kubermatic.com/blog/a-framework-for-kubernetes-incident-response/

96. Responding to a Cardholder Data Breach - PCI Security Standards Council, https://listings.pcisecuritystandards.org/documents/Responding_to_a_Cardholder_Data_Breach.pdf

97. Comparing Kubernetes Security Frameworks - ARMO, https://www.armosec.io/blog/kubernetes-security-frameworks-and-guidance/

98. Kubernetes security standard | Yandex Cloud - Documentation, https://yandex.cloud/en/docs/security/standard/kubernetes-security

99. Container and Kubernetes compliance considerations - Red Hat, https://www.redhat.com/en/topics/containers/compliance

100. Pod Security Policy - KKP Documentation, https://docs.kubermatic.com/kubermatic/v2.30/architecture/concept/kkp-concepts/kkp-security/pod-security-policy/

101. Goodbye Pod Security Policy - Hello alternatives | PPT - SlideShare, https://www.slideshare.net/slideshow/goodbye-pod-security-policy-hello-alternatives/252090548

102. Demystifying the Challenges of Kubernetes PCI Compliance - ComplianceCow, https://www.compliancecow.com/compliance/kubernetes-pcidss/

103. JFrog Artifactory Integrations - SourceForge, https://sourceforge.net/software/product/Artifactory/integrations/

104. Kubernetes Vulnerability Scanning: Best Practices and Tools - SentinelOne, https://www.sentinelone.com/cybersecurity-101/cloud-security/kubernetes-vulnerability-scanning/

105. Getting Started with the Cilium Chainguard Containers, https://edu.chainguard.dev/chainguard/chainguard-images/getting-started/cilium/

106. How Does Kubernetes Container Encryption Work in Public Clouds? - Randtronics, https://randtronics.com/how-does-kubernetes-container-encryption-work-in-public-clouds/

107. How to enforce least privilege to secure cloud and on-prem Kubernetes - Tenable, https://es-la.tenable.com/blog/taking-control-of-kubernetes-enforcing-least-privilege-to-secure-your-kubernetes-environment

108. PCI-DSS Compliance for HashiCorp Vault Secrets, https://discuss.hashicorp.com/t/pci-dss-compliance-for-hashicorp-vault-secrets/73042

109. Encrypting data in Kubernetes environments., https://www.cncf.io/wp-content/uploads/2020/08/ZettasetCNCFWebinar20200506.pdf

110. Integrate Prometheus Alertmanager - AI Operations Management - Containerized, https://docs.microfocus.com/doc/115/25.2/prometheusalertintegration

111. Prometheus Alertmanager for Kubernetes - Devtron, https://devtron.ai/blog/prometheus-alertmanager-for-kubernetes/

112. GitOps via ArgoCD - KKP Documentation, https://docs.kubermatic.com/kubermatic/v2.30/tutorials-howtos/gitops-argocd/

113. What is Automated Vulnerability Remediation? - SentinelOne, https://www.sentinelone.com/cybersecurity-101/cloud-security/what-is-automated-vulnerability-remediation/

114. What Is Kubernetes? - Aqua Security, https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-complete-guide/

115. Cloud VMs Security Across Cloud Native Environments - Aqua, https://www.aquasec.com/products/cloud-vm-security/

116. The Leading Container Security Solution for Cloud Native Apps - Aqua Security, https://www.aquasec.com/products/container-security/

117. https://www.aquasec.com/platform/container-security/

118. https://www.aquasec.com/products/cnapp/

119. https://www.aquasec.com/products/cloud-native-security-platform/

120. PCI DSS 4.0 Compliance checklist in case it's helpful for others : r/pcicompliance - Reddit, https://www.reddit.com/r/pcicompliance/comments/1ixvs2u/pci_dss_40_compliance_checklist_in_case_its/

121. Achieving compliance with Kubernetes - ARMO, https://www.armosec.io/Kubernetes_compliance.pdf

122. How to Validate PCI Compliance for Containers and Kubernetes - YouTube, https://www.youtube.com/watch?v=AMVQGmdLdxA

123. Achieving Continuous Compliance for Kubernetes – PCI, GDPR, SOC2 and more, https://blogs.oracle.com/developers/post/achieving-compliance-for-kubernetes

124. devops-cheatsheet/Security/AquaSec.md at master - GitHub, https://github.com/NotHarshhaa/devops-cheatsheet/blob/master/Security/AquaSec.md

125. Beyond OPA Gatekeeper: Enterprise-scale Admission Control for Kubernetes - Styra, https://www.styra.com/blog/beyond-opa-gatekeeper-styra-das-admission-control-for-kubernetes/

126. Policies - Kyverno, https://kyverno.io/policies/

127. Policies | Kyverno, https://release-1-8-0.kyverno.io/policies/

128. OPA Gatekeeper: Policy and Governance for Kubernetes, https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes/

129. Kubernetes and Infrastructure as Code - Palo Alto Networks, https://www.paloaltonetworks.com/cyberpedia/kubernetes-infrastructure-as-code

130. IaC in Kubernetes: 6 Tools, Tutorial and Tips for Success - Codefresh, https://codefresh.io/learn/infrastructure-as-code/iac-kubernetes/

131. Fully Automated Kubernetes Operations - Sony Interactive Entertainment, https://sonyinteractive.com/en/innovation/technology/research-academia/research/fully-automated-kubernetes-operations/

132. Compliance Made Simple with Terraform | ControlMonkey, https://controlmonkey.io/blog/compliance-made-simple-with-terraform-cloud-governess/

133. bridgecrewio/terraform-pci-starter: PCI Deployable Architecture on GCP with Terraform - GitHub, https://github.com/bridgecrewio/terraform-pci-starter

134. Kubernetes GitOps with Argo: Where to Go Next - Plural.sh, https://www.plural.sh/blog/argo-kubernetes-guide/

135. 15 Best Kubernetes Management Tools for Cluster Control in 2025 - StrongDM, https://www.strongdm.com/blog/kubernetes-management-tools

136. How to Automate Kubernetes Compliance and Policy Governance - Rafay Systems, https://rafay.co/the-kubernetes-current/how-to-automate-kubernetes-compliance-policy-governance/

137. How to automate compliance and security with Kubernetes: 3 ways, https://enterprisersproject.com/article/2020/9/how-automate-compliance-and-security-kubernetes

138. PCI DSS 4.0: The Ultimate Guide to the 12 Requirements - HeroDevs Blog, https://www.herodevs.com/blog-posts/pci-dss-4-0-the-ultimate-guide-to-the-12-requirements

139. PCI-DSS-v4_0_1.pdf, https://www.middlebury.edu/sites/default/files/2025-01/PCI-DSS-v4_0_1.pdf?fv=AKHVQBp6

140. Document Library - PCI Security Standards Council, https://www.pcisecuritystandards.org/document_library/

141. PCI DSS v4.0 Glossary - ManageEngine, https://www.manageengine.com/log-management/compliance/pci-dss-glossary.html

142. What is Kubernetes? - Definition - CyberArk, https://www.cyberark.com/what-is/kubernetes/

143. Manage PCI DSS v4.0.1 requirements with Datadog, https://www.datadoghq.com/blog/pci-dss-compliance/

144. Kubernetes monitoring - definition & overview - Sumo Logic, https://www.sumologic.com/glossary/kubernetes-monitoring/