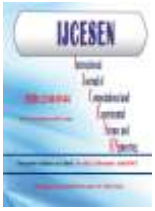
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.2 (2026) pp. 351-358
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

Secure Data Services on Container Platforms: Protecting Distributed Query Engines, Messaging Platforms, and Search Systems

Navaneeth Komirisetty*

Sr.Cybersecurity Architect at American Express Travel Related Services inc., USA
* **Corresponding Author Email:** navaneethksetty@gmail.com - **ORCID:** 0000-0002-5247-7001

Article Info:

DOI: 10.22399/ijcesen.5208
Received : 21 February 2026
Revised : 25 April 2026
Accepted : 28 April 2026

Keywords

Container Security,
Kubernetes Security,
Distributed Query Engines,
Kafka,
Elasticsearch

Abstract:

As organizations rapidly migrate their data services and other workloads to containers, the responsibility for securing them cannot fall into the hands of the application teams alone. Distributed query engines, messaging systems, search engines, and data stores amass huge troves of organizational data and provide rich query interfaces, making them a target for external and insider adversaries alike. The article describes a reference architecture that serves as an implementation guide for securing data services in cloud-native environments. Based on documentation from the most popular open-source projects in the cloud-native ecosystem and guidelines from the National Institute of Standards and Technology, the reference architecture consists of recommendations across four security domains: control plane identity and access management, data plane cryptography and network segmentation, secrets and key lifecycle management, and operational auditability. We present an anonymized enterprise implementation pattern that assembles such controls into a defensible and auditable posture across portfolios of heterogeneous services.

1. Introduction

In the last decade there has been an accelerated adoption of enterprise data services in containers with the benefits of Kubernetes orchestration, image-based deployment, and the scalability of cloud-native infrastructure. As a result, distributed query engines such as Presto and Trino, large-scale messaging brokers such as Kafka, and search applications are now commonly deployed in shared clusters alongside general-purpose applications. Although co-located systems can provide economic advantages, they also introduce a number of security concerns not present in systems with dedicated, physically separated data .

Data service abstractions are favored in enterprise architectures. They can aggregate sensitive data, span domains, expose query and streaming APIs, and host hard-coded credentials or session material that, allegedly, if compromised, could allow lateral movement. Unlike stateless application services, which handle one request/response cycle, compromised distributed query engines or messaging brokers expose cached organization-wide data over weeks or months. The NIST Application Container Security Guide (NIST SP 800-190) identified data exposure as one of the main risks for containerized applications. Container images, orchestration errors, and weak runtime security controls expose a larger attack surface compared to non-containerized deployments [1].

Platform-level security, implemented on behalf of the application team by the orchestration layer rather than negotiated with each application team, is the most scalable and consistent solution. Security controls are bundled with platform templates, network policies, and secrets management pipelines, which provide a good level of security without requiring deep security expertise in each application team. More importantly, platform-level enforcement also makes evidence collection, incident response, and compliance reporting more straightforward.

The article outlines a reusable security architecture along four main dimensions: control plane identity and access management, data plane encryption and network isolation, secrets and key lifecycle management, and operational auditability and monitoring. Each dimension is based on best practices of widely used open-source data services implementations and the NIST container security framework. The article closes with a discussion of a

mostly anonymized, large-enterprise example of implementation in practice and how the four dimensions come together.

2. Background and Related Work

2.1 Container Platforms as a Security Boundary

Container orchestration solutions like Kubernetes support several security primitives such as namespace isolation, role-based access control (RBAC) for API operations, network policy to control traffic between pods, and pod security admission controls to limit pod capabilities and increase cluster security posture. Although these primitives can help provide additional security to a cluster, in themselves they are not sufficient protection for a cluster. NIST SP 800-190 distinguishes between orchestrator-level controls (powers of the platform that will be enforced regardless of what the application running on that platform does) and application-level controls (ones that must be knocked into place by the teams that run the service). This difference helps understand the difference between what application security looks like in the real world and what it looks like in the cloud: primitives exist but are not used, or defaults are insecure and never overridden.

2.2 Data Services as Attack Targets

Distributed query engines, such as Trino (formerly known as PrestoSQL), provide SQL-like access across many backend types, including object stores, RDBMS, and columnar filesystems. Their ability to provide a query federation layer makes them an attractive target: a query engine can be abused to extract information from multiple backends through a single compromise point. The Trino documentation explicitly states that the server runs with no security by default and that security is the operator's responsibility [2].

Apache Kafka is another widely used open-source distributed event streaming software. Kafka does not ship with security enabled, and its documentation lists encrypted communication and certificate-based authentication as optional production requirements [3]. Search platforms such as Elasticsearch do provide authentication, authorization, and transport encryption, but these must be manually enabled because earlier versions often had insecure defaults, which led to unintentional data exposure [4].

2.3 Existing Security Frameworks

The NIST Cybersecurity Framework and its container extension in NIST SP 800-190 comprise the most-cited platform-level security framework [1]. The CNCF Security Technical Advisory Group has published cloud-native security whitepapers expanding on the NIST work in the context of Kubernetes-native workloads, zero-trust networking, workload identity, and software supply chain integrity and security. Though these frameworks inform the blueprint presented here, this particular article focuses on data service operational security and not the entire life cycle of the platform.

3. Control Plane: Identity, Authentication, and Authorization

3.1 Workload Identity

Strong and verifiable identity is the starting point of effective access control for human users and machine workloads. Kubernetes workload identity is represented as a namespace- and service account-scoped service account token that is mounted into a container as projected volume mounts. Also, for data services that communicate with other internal services, tokens provide a way to authenticate API requests without using long-lived credentials.

Platform teams should provide Kubernetes service accounts for different data service components (e.g., the query engine coordinator vs. worker nodes) and bind them to least-privilege role bindings. Read-only analytics user accounts should not have publish privileges for message queues or egress files, or write privileges for search indices.

3.2. Data services authentication configuration

For each of these service categories, the authentication process must be explicitly defined. For example, Trino needs a secure transport layer before it can enable any authentication mechanisms. Its specification says that TLS must be enabled before password files, LDAP, Kerberos, or JSON Web Token-based authentication methods may be enabled [2]. Transport security must be in place before authentication methods. A good architectural principle is that credentials sent over the wire unencrypted are still vulnerable, even if validated correctly.

Apache Kafka supports various authentication methods layered over its SSL/TLS transport encryption, including SASL/PLAIN, SASL/SCRAM, and client identity verification through mutual SSL certificate authentication. The Kafka documentation provides instructions for configuring SSL keystore/truststore material for

brokers and clients, with SASL being a further layer of security on top of SSL [3]. In enterprise environments, SASL/SCRAM or mutual TLS client authentication is preferred over SASL/PLAIN, as they do not send plaintext credentials even over an encrypted channel.

Thus, search engines such as Elasticsearch have native security features for user authentication, role-based access control (RBAC), and field- and document-level security. According to Elastic, their security is a unified set of features that must all be explicitly enabled, including authentication domains (native, LDAP, and PKI), role mappings, and programmatic access through API key management [4]. For shared clusters, platform teams deploying search services should federate their identity provider with the search system's authentication domains; this helps avoid the proliferation of user databases, which can become difficult to audit.

3.3 Role-Based Authorization

Authorization is the information about what actions an authenticated identity can perform. In distributed query engines, authorization is implemented according to the organization's data governance model, with schema-level and catalog-level access controls to restrict which data sources a given user or service can query. For messaging platforms, authorization is scoped to topics and controls which producers and consumers are allowed to read and write to specific streams. For search platforms, it is scoped to indexes and fields.

This approach enables platform teams to maintain a single source of truth of authorization policies, even if the decision point is still within the data service. With federated policy management, the main repository distributes the authorization policies to the services, preventing divergence and allowing for auditing of changes.

4. Data Plane: Encryption and Network Isolation

4.1 Encryption in Transit

At a minimum, all production data services should encrypt data in transit. All three types of data services discussed here—query engines, messaging brokers, and search engines—support the TLS protocol to encrypt data in transit, both between clients and servers and between nodes in a cluster.

To enable TLS in Trino, the user generates certificate, keystore, and truststore files and enables HTTPS listener configuration in both the coordinator and worker nodes. Trino documentation describes keys used by each component. It also recommends using a certificate authority with a shared trust chain amongst the nodes in the cluster [2]. If the cluster handles sensitive data, the communication from the coordinator to the workers in the cluster must be encrypted.

For Apache Kafka, SSL is configured separately for the replication channel (the network between brokers) and the producer/consumer connections (the network between clients and brokers). The documentation highlights how to configure SSL per listener, allowing Kafka administrators to keep SSL usage on external listeners required while relaxing configuration for internal cluster communication channels. It is generally recommended that all channels be encrypted [3].

4.2 Mutual TLS for Internal Traffic

Conversely, where communication takes place across a shared network, service-to-service, mutual TLS (mTLS) provides better assurance than one-way server authentication, as both the client and the server authenticate each other. This enables protection against impersonation by third parties that may have access to a valid server certificate. Service meshes such as Istio and Linkerd have implemented mTLS transparently for traffic between workloads on the mesh, avoiding the need to configure TLS per service.

For mTLS without a service mesh, platform teams can enforce the use of client certificates by including the creation of client certificates as part of the service deployment template. The only requirement is to use a separate certificate authority (CA) to issue those client certificates, besides the public CAs, to ensure that possession of a client certificate implies membership in the platform environment.

4.3 Network Policy Enforcement

Kubernetes Network Policies provide declarative controls for which workloads can initiate and receive connections [5]. For data services, the recommended posture is for a namespace-level default deny ingress policy with explicit allow policies for consumers of that service. For instance, a query engine namespace could allow ingress traffic from the analytics application namespace and from the monitoring namespace but not from any other namespaces.

The Container Network Interface (CNI) plugin is used to implement network policy. A cluster that does not have a CNI plugin capable of enforcing policies (Calico, Cilium, or Weave) in the relevant

namespace, regardless of policy rules, cannot enforce network policies. Platform teams should expect and verify that the CNI can enforce policies. They should also periodically audit the policy that is enforced by the CNI. Kubernetes Network Policies are a layer 3/4 (IP and port) primitive. Application layer controls and mTLS provide a complementary security function for traffic after the network policy filter.

Network segmentation between namespaces becomes even more imperative in multi-tenant shared cluster scenarios where application teams belonging to separate business units co-exist. Without namespace-level network isolation, a compromised application workload might be able to reach the data service endpoints of other teams directly. Platform teams should treat network policy configuration in the namespaces where data services run as a mandatory, non-delegable control.

5. Secrets and Key Management Patterns

5.1 Credential sprawl and its risks

Data services make use of a range of credentials, including passwords and API keys for authentication, TLS certificates and private keys for transport, and connection strings to the backend data source. In the absence of centralized secrets management, these are frequently present in the layers of container images, Kubernetes ConfigMaps, Helm value files in version control, and pod specifications as environment variables. The highest priority risk is the presence of sensitive information in container images, which NIST SP 800-190 highlights because they are highly likely to move from one environment to another and because they may persist long after the associated credentials have changed [1].

5.2 Centralized Secrets Management

In a mature container platform, secrets management should be done with a central solution like HashiCorp Vault or the secret manager of the cloud provider (AWS Secrets Manager, Azure Key Vault, Google Cloud Secret Manager, and others). The data services should be able to access these secrets at runtime via init containers that create ephemeral volume mounts with credentials, via sidecar containers that mount a shared file with the credentials, or via a call to a secrets manager API by the application.

Secrets management integrations should generate secrets dynamically (when possible), only issue short-lived credentials, log every access to credentials, and allow credential revocation. Dynamic secrets, which provide a different secret for every requesting workload, and revocation, which forces the workload to stop making use of the secret as soon as it is stopped, prove to be especially effective at reducing the blast radius when secrets are leaked.

5.3 Certificate Lifecycle Management

Managing TLS certificates for data services is a challenging and labor-intensive task. Each data service requires a TLS certificate, which has a limited lifetime. Outdated certificates can cause service interruptions, which may be difficult to diagnose in production. To avoid such scenarios, the platform teams should standardize certificate issuance and rotation around a certificate authority embedded in the platform's orchestration layer.

Tools such as cert-manager (a certificate controller for Kubernetes) can be used to obtain certificates from internal CAs or cloud provider certificate services and to automatically renew them before expiration without operator intervention. Services with a requirement for JKS certificates, such as Kafka or older JVM-based services, may require additional automation to convert PEM certificates from the certificate controller to JKS format.

Certificate lifecycle management includes certificate expiration alerts, rotation workflows and procedures, and validation of rollout after rotation to verify that services have reloaded new certificate material. Most certificate-related outages are not due to failure of rotation tools but because services cache certificate material in memory and do not reload new material when the actual on-disk certificate is replaced.

6. Auditability and Operational Monitoring

6.1 Audit Logging Requirements

Audit is an essential part of both security incident response and regulatory compliance. For data services, audit-relevant activities include authentication successes and failures; authorization decisions, especially to deny access; administrative configuration changes; and, when possible, query-level logging of which users accessed which data. Some systems like Kafka support auditing at a topic level. For example, Kafka can audit producers and consumers, recording which services wrote or read sensitive event streams [3]. Elastic's audit logs capture authentication events, access control decisions, and index-level actions [4].

The Kubernetes API server records all accesses or modifications to secrets, ConfigMaps, and network policies in its control plane audit log, in addition to

the application-level audit log, which captures actions happening at the platform layer (orchestration). The orchestration and data service audit logs emitted by the platform team should be pushed to a central tamper-clear log store with retention based on compliance requirements.

6.2 Security Monitoring and Anomaly Detection

Security-relevant signals include not just audit logging but also failures in authentication, blocked network connections reported by CNI plugins that expose various policy decision metrics, certificate validity periods, and unusual patterns in query volumes and data export volumes. Monitoring signals in aggregate helps security operations teams distinguish between expected operational patterns and ones that provide strong evidence of an attempted compromise of the service.

Likewise, it can be helpful to define observable security metrics for a data service, in the same way that service-level objectives might be defined for availability and latency. These could be the number of failed logins for the last hour (alert when above normal), the number of network connections blocked in the last day (open an investigation only when above normal), or the number of certificates in need of rotation (alert when in the next 30 days). These indicators can also be used monthly to create a historical record of the effectiveness of controls and, in security program reporting, to provide evidence of compliance.

6.3 Incident Response Considerations

Data service security events may require incident response playbooks. If a query engine were compromised, it may have exfiltrated from multiple backend services at the same time. Responders need to know how to review the query engine's catalog configuration and connection credentials. Alternatively, a compromised message broker could have given the attacker access to the internal event streams and event queues by injecting messages.

All data services use the same identity, network, and secrets architecture, resulting in a more predictable platform. Incident responders dealing with data service environments can rely upon a known audit log format, a known network policy model, and a known credential rotation procedure, regardless of which data service they are dealing with. This form of predictability may be hard when security configurations are done ad hoc by individual service teams.

7. Anonymized Implementation Pattern

One of the larger enterprise environments for these sizes has the organization hosting a shared Kubernetes platform with a portfolio of data services: a distributed query engine to run analytics workloads, a Kafka-based event streaming infrastructure to integrate applications, and an Elasticsearch cluster for operational search and log analytics needs.

This includes three layers of default behavior for all deployments of a service that exposes data: an encrypted transport layer, a certificate management layer where services use certificates signed by the platform's internal certificate authority, and an automated and monitored system for certificate rotation. An alerting rule is fired if a certificate in a data service namespace is about to expire within 21 days. There are also identity and access controls integrated with the identity provider. Human access to data service administrative endpoints is protected by a single sign-on system. Service-to-service authentication is performed using short-lived tokens or mutually TLS client certificates issued by the platform. Finally, each data service namespace is protected by a default-deny network policy to which explicit allow rules are committed to a policy-as-code repository and reviewed as part of the preceding change management process.

The major lesson learned is that security posture and operational complexity are driven more by the consistency of the overall security model rather than the sophistication of the individual security control. By adopting a consistent set of certificate management, identity integration, and network policy patterns across these three types of data services, the security team has a common set of tools to assess compliance and collect evidence for services rather than manual processes for each service. Incident response drills run on this environment consistently have shorter containment times than ad hoc, heterogeneous security environments built in the field.

Another takeaway: security controls should be first-class operational requirements from day one. The data services were deployed without standardized security templates, which were later retrofitted (notably for certificate management and configuring network policies), leaving the services in question vulnerable. Those services deployed from a standardized, security-hardened template from the start required far less remediation.

8. Discussion

We have chosen to optimize for enforcement at the platform level rather than for enforcement through the use of application-specific negotiation because

the platform team needs to create templates, policy-as-code infrastructure, and automation tooling before other data service teams can inherit these controls. The initial cost is outweighed by long-term reductions in security debt, audit complexity, and incident response workload.

Platform-level security cannot be a substitute for proper application-level security hygiene. Data services still need to be patched with security updates, as the NIST container security guide has flagged the use of deprecated software in container images as a persistent source of vulnerability [1]. Although security updates to query engines, messaging brokers, and search systems are released sporadically, platform teams will need to bake image updates into their operation.

Because this design is service agnostic, the principles of encrypted transport, strong authentication, network isolation, centralized secrets management, and operational auditability apply equally to the open source projects and the commercial versions based on or similar to them. When platform teams onboard new data services, they should consult the documentation for the service and see how it aligns with the principles of this design. They should be used as a checklist of configuration work rather than secure defaults .

Table 1: *Service-to-security dimensions capability matrix. This table maps each service against the four security dimensions, showing what is native, what requires configuration, and what the platform must supply [5] [6] [8] [9]*

	Control plane (Identity & AuthN/Z)	**Data plane (Encryption & isolation)**	**Secrets and key life cycle**	**Auditability & monitoring**
Trino	Requires config	Requires config	Platform-supplied	Requires config
	No security by default; TLS must be enabled before any AuthN type (password, LDAP, Kerberos, JWT) can be activated	TLS for coordinator–worker and client–coordinator traffic; keystore/truststore provisioning is the operator's responsibility.	No native secrets manager; credentials and certs must be injected via platform secrets pipeline	Query-level logging available; event forwarding to SIEM requires operator configuration
Apache Kafka	Requires config	Requires config	Platform-supplied	Requires config
	SASL/PLAIN, SASL/SCRAM, and mutual TLS client auth supported; none enabled by default	SSL configurable per listener (replication vs. client); all channels should be encrypted, but it is not enforced by default	Keystore/truststore management and rotation are operator responsibilities; cert-manager integration recommended	Topic-level producer/consumer audit logging available; must be explicitly enabled and forwarded
Elastic-search	Native feature	Native feature	Platform-supplied	Native feature
	The unified security feature set covers native, LDAP, and PKI auth realms, RBAC, and API key management; must be explicitly enabled	TLS for HTTP and transport layers built in; field- and document-level encryption available at the application layer	Certificate provisioning and rotation still platform responsibility despite native TLS support	Audit logging covers AuthN events, access control decisions, and index-level operations natively
Platform (K8s)	Native feature	Native feature	Native feature	Native feature
	RBAC for API operations; service account token projection for workload identity	Network Policies enforced by CNI plugin; mTLS via service mesh (Istio/Linkerd)	Secrets API, integration with Vault or cloud secret managers, and cert-manager for certificate lifecycle	API server audit log captures all control-plane operations on secrets, ConfigMaps, and network policies

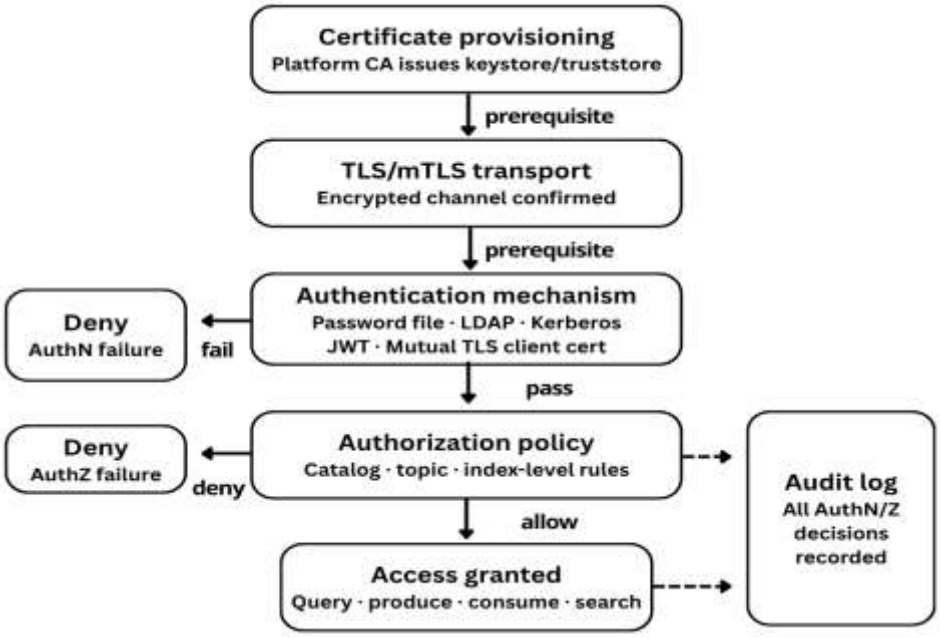


Figure 1: *Control plane authentication sequence flowchart. Shows the mandatory transport-before-authentication ordering dependency that applies across all three services*

Table 2: *Security monitoring indicators [4] [5] [6] [12] [13]*

Indicator	Category	Source	Alert trigger	Response action
Failed authentication attempts	AuthN	Data service auth log; Kubernetes API audit log	The rate exceeds rolling baseline within measurement window	Investigate source identity; check for credential stuffing or misconfigured service accounts.
Network connections denied by policy	Network	CNI plugin policy decision metrics (Calico, Cilium)	The count exceeds baseline per namespace per day	Review denied source/destination pairs and determine if there is a policy gap or attempted lateral movement
Certificates expiring within 21 days	Cert	cert-manager certificate status; platform monitoring	Any certificate in a data service namespace enters the 21-day window	Initiate rotation workflow; validate service reload post-rotation
Authorization denials (data-layer)	AuthZ	Elasticsearch audit log; Kafka ACL deny events	The deny rate for a given identity exceeds normal access pattern	Correlate with change management record; escalate if unrecognized actor or out-of-window timing

9. Conclusions

The architecture seeks to deliver security to data services in container-based platforms as a coherent layered approach in which the platform functions as the main security boundary. The article describes the architecture along four dimensions (control plane identity and access management, data plane encryption and network isolation, secrets and key lifecycle management, and auditability) derived from ubiquitous open-source data services , and the NIST container security reference architecture.

To achieve this network effect, one needs to implement platform-level templates for deploying controls and enforcing policy across all data distribution tools. When distributed query engines, messaging platforms, and search systems adopt a common security model, the overall environment is easier to audit, respond to incidents, and defend an organization's data assets.

Future work may investigate the interaction of data service security with supply chain security (how to verify provenance and integrity of data service container images in a cloud platform) and emerging workload identity standards like SPIFFE/SPIRE that express richer identity semantics than Kubernetes Service Account tokens. The security architecture in this article can be a foundation upon which additional security controls can be built as enterprise data service portfolios grow in scale and heterogeneity.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Murugiah Souppaya, et al., "Application Container Security Guide," Special Publication 800-190, 2017. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

[2] Trino Documentation, "TLS and HTTPS," Trino Security Configuration. [Online]. Available: https://trino.io/docs/current/security/tls.html

[3] Trino Documentation, "Authentication Types," Trino Security Configuration. [Online]. Available: https://trino.io/docs/current/security/authentication-types.html

[4] Apache Kafka Documentation, "Encryption and Authentication Using SSL," Apache Kafka 4.1 Documentation, 2025. [Online]. Available: https://kafka.apache.org/41/security/encryption-and-authentication-using-ssl/

[5] Elastic Documentation, "Security Overview and Setup," Elastic Cloud and Self-Managed Documentation. [Online]. Available: https://www.elastic.co/docs/deploy-manage/security

[6] Kubernetes Documentation, "Network Policies," Kubernetes Concepts: Services, Load Balancing, and Networking, 2024. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/

[7] Cloud Native Computing Foundation, "CNCF Cloud Native Security Whitepaper," v2, 2022. [Online]. Available: https://www.cncf.io/wp-content/uploads/2022/06/CNCF_cloud-native-security-whitepaper-May2022-v2.pdf

[8] Kubernetes Documentation, "Using RBAC Authorization," Kubernetes Reference Documentation, 2024. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[9] Kubernetes Documentation, "Managing Service Accounts," Kubernetes Tasks, 2024. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/

[10] HashiCorp, "Vault Documentation: Secrets Engines," HashiCorp Developer. [Online]. Available: https://developer.hashicorp.com/vault/docs/secrets

[11] cert-manager Documentation, "Introduction to cert-manager," cert-manager Project Documentation. [Online]. Available: https://cert-manager.io/docs/

[12] Istio Documentation, "Security: Mutual TLS Migration," Istio Concepts. [Online]. Available: https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/

[13] Kubernetes Documentation, "Auditing," Kubernetes Tasks: Cluster Administration, 2024. [Online]. Available: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/

[14] SPIFFE Project, "SPIFFE: Secure Production Identity Framework for Everyone," SPIFFE Documentation. [Online]. Available: https://spiffe.io/docs/latest/spiffe-about/overview/

[15] Adrian Mouat, Docker Security: Using Containers Safely in Production. O'Reilly Media, 2016. Available: https://theswissbay.ch/pdf/Books/Computer%20science/O'Reilly/docker-security.pdf