**6**

**Securing Your Workloads and Infrastructure**

*“Kubernetes has become the de-facto standard for managing containerized applications. It is used by many organizations to deploy, scale, and manage their applications in a distributed environment. However, with the rise of Kubernetes, there has been an increased focus on Kubernetes security. Security is essential to any infrastructure, and Kubernetes is no exception.”*

—Anonymous

Securing workloads on Kubernetes and the connected infrastructure is one of the most important—and, at the same time, one of the most forgotten and procrastinated—steps in deployments. The goal of this chapter is to highlight the various security aspects of running Kubernetes in production. These sections look at the various facets of security related to Kubernetes and dive deeper into some of the more common tools and practices of securing workloads and infrastructure when using Kubernetes.

**Kubernetes Security Challenges**

A Kubernetes cluster is a complex system made up of a large number of moving parts, ranging from the core Kubernetes components and controllers to third-party and open-source software add-ons. Moreover, Kubernetes is a distributed system that uses networks to communicate and orchestrate work. It also relies on more fundamental building blocks, such as a container runtime and operating system features. This means that the security challenges of running Kubernetes are distributed across several strata:

* **Access control:** Kubernetes provides several access control mechanisms, such as role-based access control (RBAC) and network policies, to control access to the Kubernetes application programming interface (API) and resources. Misconfigurations or incorrect use of these mechanisms can lead to unauthorized access.
* **Network security:** Kubernetes is designed to run containerized applications in a networked environment; however, this introduces several challenges, such as handling network segmentation, securing the network, and managing network policies.
* **Container security:** Kubernetes uses containers to run applications, which means that the security of the containers is critical. Vulnerabilities in the containers can be exploited to gain unauthorized access to the Kubernetes environment.
* **Supply chain security:** The container image supply chain involves several parties, including developers, maintainers, and third-party software vendors. Any of these parties can introduce vulnerabilities into the container images, which can then be exploited to gain unauthorized access.

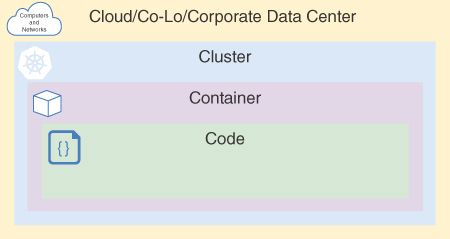
**Concepts of Kubernetes Security**

With the expansive nature of Kubernetes security, it is often easier to think about security pillars for your Kubernetes cluster instead of the nebulous concept of securing your cluster as a whole. With this view you can identify and segregate some of the primary security pillars, as follows:

* **Authentication:** The process of verifying the identity of a user or process. Kubernetes supports several authentication mechanisms, such as X.509 client certificates, static token files, bootstrap tokens, and OpenID Connect tokens.
* **Network security:** The process of protecting the network and its resources from unauthorized access, misuse, modification, and denial. Kubernetes supports several network security mechanisms, such as network policies and network segmentation.
* **Container security:** The process of protecting the container and its resources from unauthorized access, misuse, modification, and denial. Kubernetes supports several container security mechanisms, such as container vulnerability scanning, image signing, and least privilege.
* **Threat modeling:** The process of identifying, understanding, and mitigating the security risks to a system or application. Threat modeling is a critical step in the software development lifecycle.

**4Cs of Kubernetes Security**

The Kubernetes project promotes a layered view for cloud native security that is complementary to the defense-in-depth approach to security. The four layers of security, proposed by the Kubernetes project, is the recommendation to secure the cloud, clusters, containers, and code. In most cases, this model cuts down on cross-team dependencies to implement security because each layer is owned and managed by an individual team. Furthermore, this layered view of securing your workloads can make it easy to keep a tight focus on the individual areas or layers. [Figure 6-1](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig01) illustrates the 4Cs of Kubernetes security.



**Figure 6-1** 4Cs of Kubernetes Security

It is important to note that vulnerabilities at the cluster layer can significantly affect the layers within, including the code layer. This means that even if you try to protect your code layer, it might still be vulnerable if there are issues with the cluster layer.

Each component of a cluster has its own potential security concerns, which the next section explores. Container security relies on trusted code. By combining container vulnerability scanning and image signing, you can ensure that nothing has been modified, thus preventing any malicious activity that could bypass the least privileges required.

Numerous cloud providers exist, each with their own security best practices and settings. Although most are secure, it’s always a good idea to research the settings yourself instead of relying solely on the provider’s security measures. This is especially important if you are using a multicloud strategy with different security configurations. The next section covers Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) and the supporting infrastructure.

**Securing Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE)**

Securing the cloud infrastructure for the managed Kubernetes is the first *C* of the 4Cs of Kubernetes security, and this is the first layer to secure your workloads.

Oracle Cloud Infrastructure is a security-focused cloud provider that offers a highly secure computing environment. It starts with off-box virtualization and is complemented by secure defaults for all infrastructure components at every level. This focus on security extends to the Kubernetes clusters created by OKE. These clusters have secure configurations with minimal access, by default, and feature different options and configurations to accommodate corporate policies and methodologies.

With Oracle managing the Kubernetes Control Plane, the security implementations remain on the cloud layer, allowing abstraction for the cluster to use standard Kubernetes security features—the second *C* layer.

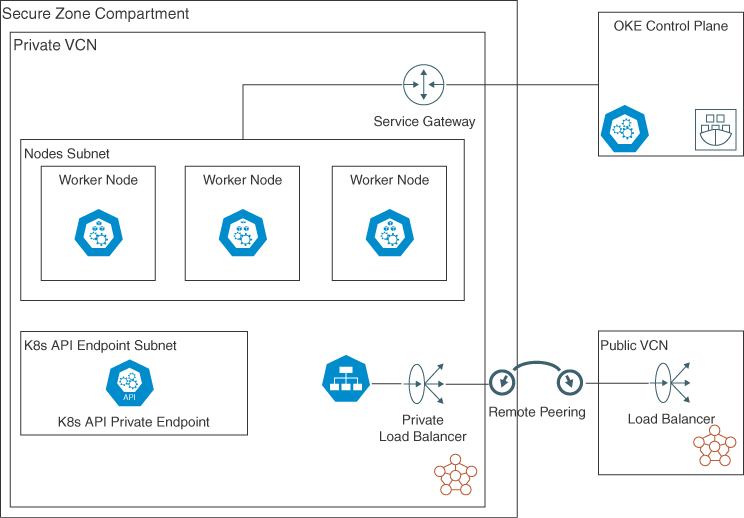
**Private Clusters**

Creating a private cluster is an option when provisioning a new OKE cluster, making the Kubernetes API endpoint private and accessible only from within the cluster’s virtual cloud network (VCN). Using this option, you can have a private cluster with no public IP address. To access Kubernetes APIs from tools such as kubectl, you need to use a bastion host or a connection that has access to the private network.

The section “[Creating a Cluster](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04lev1sec8)” in [Chapter 4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04), “[Understanding Container Engine for Kubernetes](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04),” covers the creation of a private cluster using the OCI console, CLI, and Terraform.

Using a private endpoint and private workers, you isolate the nodes and pods from the Internet, by default, for either managed or virtual node pools.

Other extra measures to guarantee that your cluster will stay private include using security zones with the Maximum-Security Recipe, which prevents any user, including administrators, from exposing the cluster to the Internet. In this case, if you need to create Load Balancers that expose services to the Internet, you need to use a subnet that is not part of the security zone or use a subnet from a different VCN and configure the peering. [Figure 6-2](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig02) illustrates the separation of resources on a private cluster.



**Figure 6-2** Private Cluster Diagram

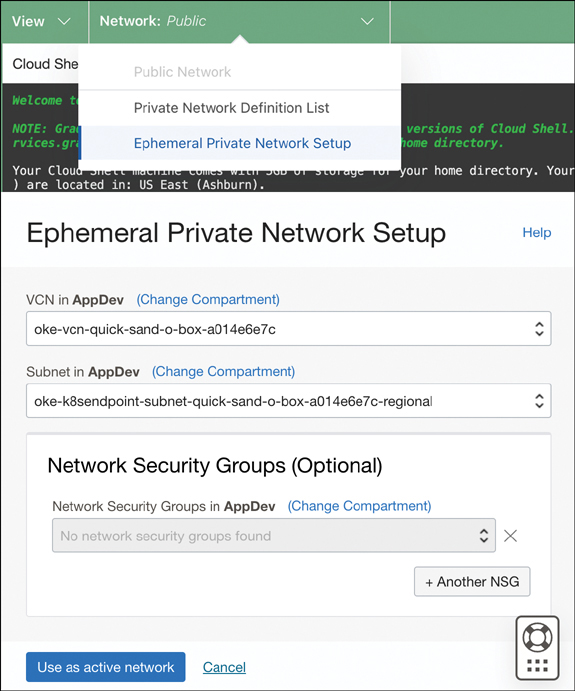
For an extra level of security for a private cluster, you can consider limiting the images used by the deployments by using a private registry, such as Oracle Cloud Infrastructure Registry (OCIR), or a private repository, such as Harbor, to ensure that only trusted images are used.

**Accessing Private Clusters**

Bastion hosts (also known as jump boxes) are the most common way to access private clusters. Bastion hosts are virtual machines that are deployed in the same VCN as the cluster. Bastion hosts are used to access the cluster’s Kubernetes API endpoint. Bastion hosts are not part of the cluster and are not managed by Kubernetes. Bastion hosts are not required to be running at all times; they can be started and stopped as needed. To create a new bastion host, you can use the OCI Console, OCI CLI, or OCI Terraform Provider to create a new compute instance that is either in the same VCN as the cluster or that has connectivity to the cluster’s VCN.

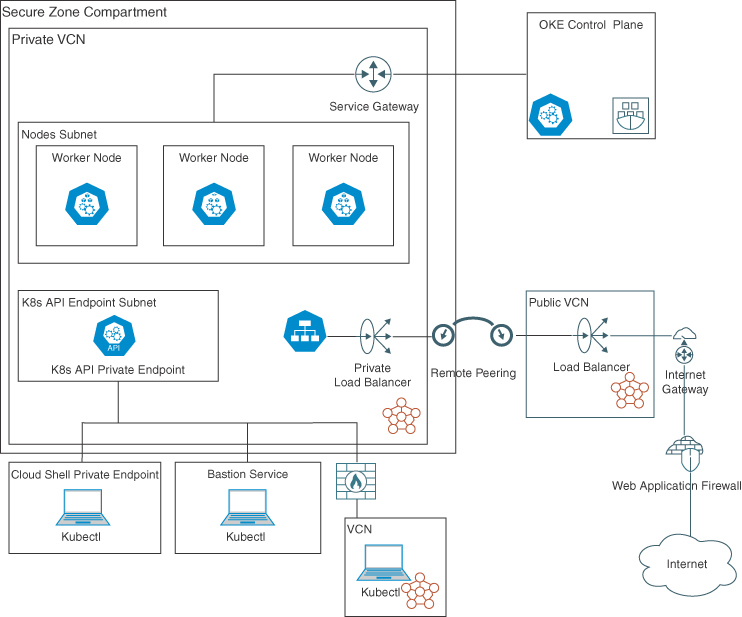
Another option for accessing private clusters is Bastion Service, a service that runs in the cluster and provides access to the cluster’s Kubernetes API endpoint. Different from the Bastion Host, the Bastion Service does not need a compute instance to run and does not need to be in the cluster’s VCN. As with the other resources, you can create the Bastion Service with any OCI tool.

Another powerful tool to access private clusters is the OCI Cloud Shell, a browser-based shell that you can use to access OCI resources. OCI Cloud Shell is a fully managed service that runs in the cloud and provides access to OCI resources without the need to install and configure any software on your local machine. After configuring the kubeconfig file, you can change the OCI Cloud Shell network to be the same VCN as the Cluster, selecting a subnet that has access to the cluster’s Kubernetes API endpoint. [Figure 6-3](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig03) illustrates the OCI Cloud Shell option to access a private network.



**Figure 6-3** OCI Cloud Shell Ephemeral Private Network Setup

In some cases, you might not want to use bastion hosts or jump boxes to access your private clusters. Instead, you can use a VPN connection to access the cluster’s Kubernetes API endpoint. To create a VPN connection or a remote peering connection with access to a different cloud or on-premises network, you need to configure the connection according to your network topology. [Figure 6-4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig04) illustrates different options to access and maintain OKE private clusters.



**Figure 6-4** Different Access Options for OKE Private Clusters

When deploying apps to private clusters and using tools such as Terraform (using the Terraform Kubernetes/Helm providers) or OCI Ansible modules, you can use the OCI Bastion Service or OCI Cloud Shell to access the cluster’s Kubernetes API endpoint. When using OCI Resource Manager, you can create a private endpoint and configure the Terraform Kubernetes Provider to use the private endpoint.

[Listing 6-1](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_1) provides an example of a Terraform Kubernetes Provider configuration using the OCI Resource Manager private endpoint.

**Listing 6-1** OCI Resource Manager Private Endpoint Terraform HCL Script

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0247-01a)

...

resource "oci\_resourcemanager\_private\_endpoint" "private\_kubernetes\_endpoint" {

compartment\_id = local.oke\_compartment\_ocid

display\_name = "Private Endpoint for OKE"

description = "Resource Manager Private Endpoint for OKE"

vcn\_id = var.vcn\_id

subnet\_id = var.k8s\_endpoint\_subnet\_id

freeform\_tags = var.cluster\_tags.freeformTags

defined\_tags = var.cluster\_tags.definedTags

}

# Resolves the private IP of the customer's private endpoint to a NAT IP.

data "oci\_resourcemanager\_private\_endpoint\_reachable\_ip" "private\_kubernetes\_

endpoint" {

private\_endpoint\_id = var.create\_new\_oke\_cluster ? oci\_resourcemanager\_

private\_endpoint.private\_kubernetes\_endpoint.id : var.existent\_oke\_cluster\_

private\_endpoint

private\_ip = trimsuffix(oci\_containerengine\_cluster.oke\_cluster[0].

endpoints.0.private\_endpoint, ":6443")

}

...

provider "kubernetes" {

host = (var.cluster\_endpoint\_visibility == "Private") ?

("https://${data.oci\_resourcemanager\_private\_endpoint\_reachable\_ip.private\_

kubernetes\_endpoint.ip\_address}:6443") : (yamldecode(module.oke.kubeconfig)

["clusters"][0]["cluster"]["server"])

cluster\_ca\_certificate = base64decode(yamldecode(module.oke-quickstart.kubeconfig)

["clusters"][0]["cluster"]["certificate-authority-data"])

insecure = (var.cluster\_endpoint\_visibility == "Private") ? true :

false

exec {

api\_version = "client.authentication.k8s.io/v1beta1"

args = ["ce", "cluster", "generate-token", "--cluster-id",

yamldecode(module.oke-quickstart.kubeconfig)["users"][0]["user"]["exec"]["args"]

[4], "--region", yamldecode(module.oke-quickstart.kubeconfig)["users"][0]

["user"]["exec"]["args"][6]]

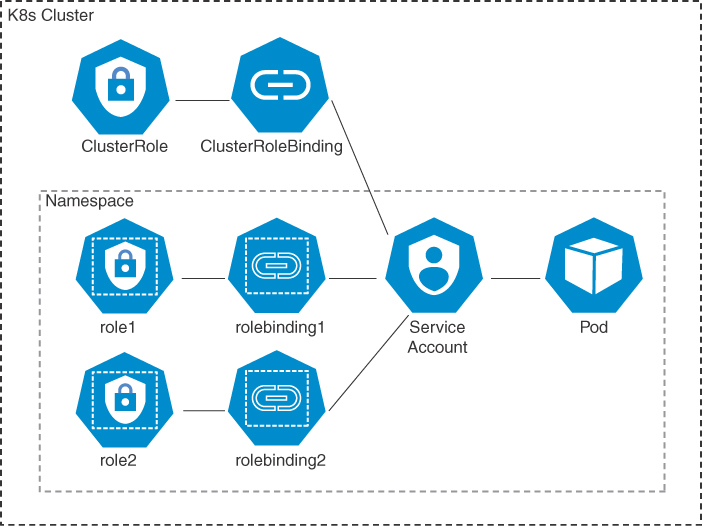
command = "oci"

}

}

**Kubernetes Role-Based Access Control (RBAC) with OCI IAM Groups**

Kubernetes offers different ways to manage access to the cluster. The most common way is to use Kubernetes RBAC, a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. RBAC enables you to define roles and assign them to users or groups. You can then use these roles to control access to Kubernetes resources. [Figure 6-5](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig05) illustrates the Kubernetes RBAC relationship.



**Figure 6-5** Kubernetes RBAC

OKE natively integrates the clusters with the OCI Identity and Access Management (IAM) service. OCI IAM provides strong user authentication to access your clusters and gives you authorization to use the OKE API so that you can define cluster administrators and cluster users.

[Chapter 4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04) covers the integration between OCI IAM and Kubernetes RBAC in detail, including identity federation.

Following the principle of least privilege, your users can access only the Kubernetes resources that they’re authorized to. Kubernetes RBAC is aware of OCI user identities and can bind Kubernetes roles to OCI users. To streamline the configuration of Kubernetes RBAC, OKE added support for binding Kubernetes roles to OCI IAM groups. As a result, the Kubernetes RBAC configuration is greatly simplified. Moreover, you can apply the Kubernetes role definition and the binding to OCI groups across multiple clusters.

Consider the example of a product inventory application. This application runs in the Kubernetes namespace inventory. An OCI group named inventory-app-admin includes OCI users responsible for the lifecycle of the inventory application. A Kubernetes role allows full access to the namespace inventory. This role is bound to the OCI group inventory-app-admin. As a result, all users in the OCI group inventory-app-admin can create and delete any applications in the namespace inventory, but not in any other namespaces.

Administrators can simply add or remove members to or from groups using the Oracle Cloud Console or API.

**OCI IAM Multifactor Authentication (MFA) for Kubernetes API**

The authentication of a user who makes Kubernetes API requests through the kubectl CLI relies on an RSA public key in PEM format (minimum 2,048 bits). Although this authentication method is strong, you might want to add a second factor to complete the authentication. OKE supports OCI IAM MFA. With MFA enabled in the IAM service, when a user connects to a cluster Kubernetes API, the RSA key is checked, which is the first factor. The user is then prompted to provide a second verification code from a registered MFA device, such as a phone, as the second factor. The two factors work together, offering an extra layer of security to verify the user’s identity and complete the authentication process.

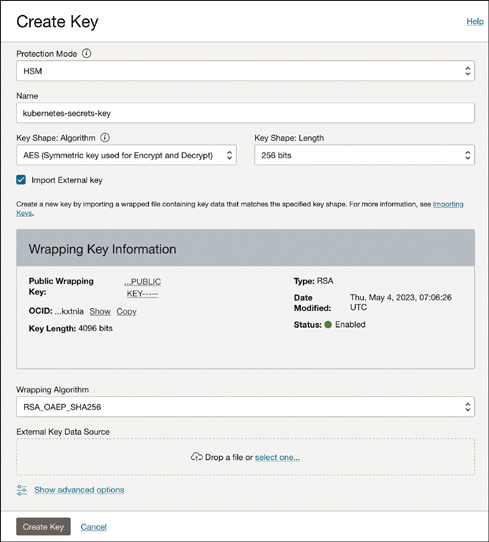
Note that, to use MFA, you must have a registered MFA device and the OCI policies to enforce the security. You also need to update the kubeconfig file to use the MFA token. For more information, see the OCI documentation.

**Data Encryption and Key Management Service**

By default, OCI provides data encryption for all data stored in the cloud. This includes block volumes, boot volumes, file storage, volume backups, and other services. OKE consumes these services transparently to Kubernetes, including encryption at rest.

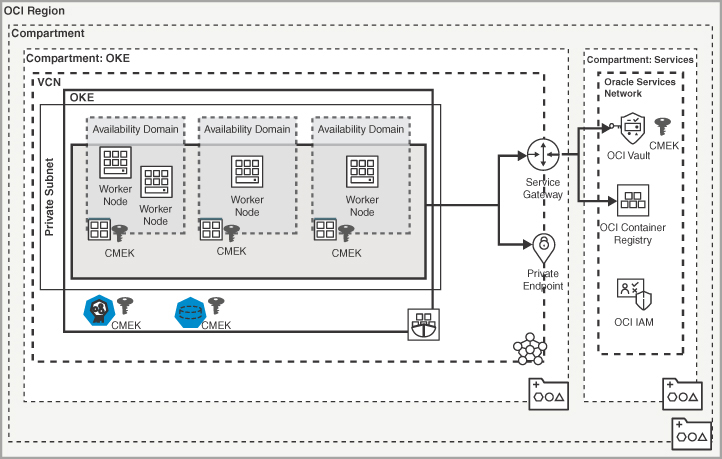
You might want to manage the encryption keys for some use cases. For example, you can use OCI Vault, a Key Management Service (KMS), to create, manage, and use encryption keys to encrypt and decrypt data. You can also use the OCI KMS service to manage the lifecycle of the encryption keys. OCI Vault offers keys in highly available and durable hardware security modules (HSMs) that meet Federal Information Processing Standards (FIPS) 140-2 Security Level 3 security certification.

Using customer-managed encryption keys (CMEK) for OKE with OCI Vault, you first need to create a vault and a key in OCI Vault. Then you enter the OCID of the key on the OKE cluster provisioning for each component that you want to use with CMEK instead of Oracle-managed keys, as illustrated in [Figure 6-6](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig06).



**Figure 6-6** OCI Vault Create Key Example

After creating the vault and key, you can enable them on different parts of your Kubernetes cluster. You can reuse the same vault/key for all resources or select a different one for each resource. [Figure 6-7](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig07) illustrates the resources where you can use customer-managed encryption keys.



**Figure 6-7** Resources That Can Use CMEK

**Managing Secrets**

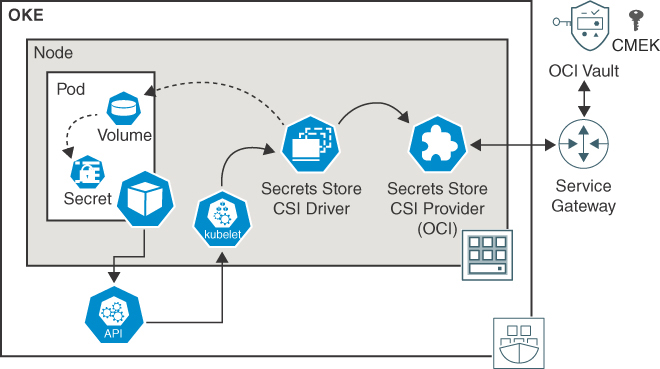
In Kubernetes, secrets are used to securely store and manage sensitive information. Secrets can store credentials, API keys, tokens, and other sensitive data that applications require for secure communication or access to external services. Kubernetes secrets are Base64 encoded and stored as an API resource within the cluster.

When using Container Engine for Kubernetes to create Kubernetes clusters, you have two options for storing application secrets:

* Kubernetes secret objects that are stored and managed inside etcd
* External secrets that use the Kubernetes Secrets Store CSI driver

Kubernetes secret objects can be stored and managed in etcd, an open-source distributed key-value store that Kubernetes uses for cluster coordination and state management. In Kubernetes clusters created by Container Engine for Kubernetes, etcd writes data to and reads data from block storage volumes in the Oracle Cloud Infrastructure Block Volume service. By default, Oracle encrypts data in block volumes at rest; this includes etcd and Kubernetes secrets. This encryption is managed by Oracle using a master encryption key, meaning that no action is required on your part. You can also bring your own encryption keys through the OCI Vault service to encypt these secrets in etcd as well.

External secrets can be securely stored and managed through the Kubernetes Secrets Store CSI driver (secrets-store.csi.k8s.io). This driver integrates secrets stores with Kubernetes clusters as CSI volumes, to enable the mounting of multiple secrets, keys, and certificates stored externally into pods. When this driver is attached, the data in the volume is mounted into the application container’s file system. OCI Vault is an example of an external secrets store. Oracle offers the open-source OCI Secrets Store CSI Driver Provider to enable Kubernetes clusters to access secrets in Vault. [Figure 6-8](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig08) illustrates the interaction of Kubernetes secrets with the OCI Vault using the Kubernetes CSI.



**Figure 6-8** Kubernetes Secrets Connecting to the OCI Vault Using CSI

In this model, the Kubernetes Secrets Store CSI Driver Provider is deployed to the Kubernetes cluster as a DaemonSet. The DaemonSet ensures that a CSI driver pod is running on each node in the cluster. The Kubernetes Secrets Store CSI Driver Provider communicates with the Vault service to retrieve secrets. The Kubernetes Secrets Store CSI Driver Provider mounts the secrets as a volume in the pod. Now the application running in the pod can access the secrets as files in the volume. The Kubernetes Secrets Store CSI Driver Provider can also be configured to renew and rotate the secrets before they expire automatically.

To configure this in your cluster, you need to create the appropriate OCI policies and decide whether you will use user principals or instance principals to access the OCI Vault. You can then install the CSI driver and create a Kubernetes resource of the kind SecretProviderClass to link the secret volume to the OCI Vault secret. The specific steps for configuring the OCI secret store driver can be found at the oci-secrets-store-csi-driver GitHub repository.[**1**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_1a)

Both aforementioned options enable you to use master encryption keys from other tenancies. You simply need to include the appropriate policies to authorize the tenancy running OKE to use keys from other tenancies. The OCI documentation[**2**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_2a) covers the specific policies required and the detailed steps for configuration.

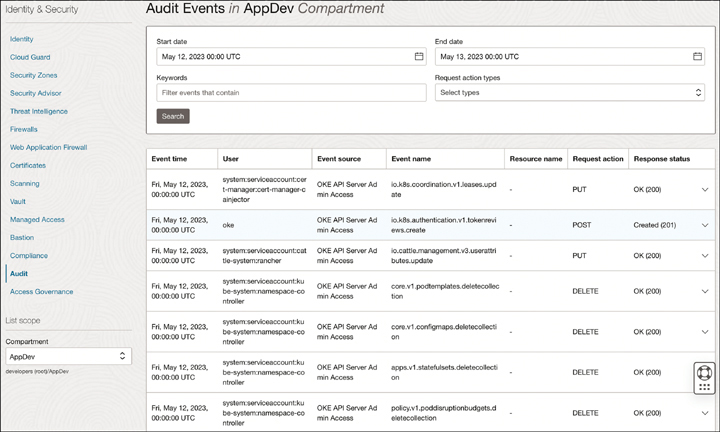
**Audit Logging**

Audit logging plays a crucial role in maintaining the security, compliance, and operational integrity of Kubernetes clusters. It provides a detailed record of all activities within the cluster, including API calls, configuration changes, and resource access. Audit logs enable organizations to do the following:

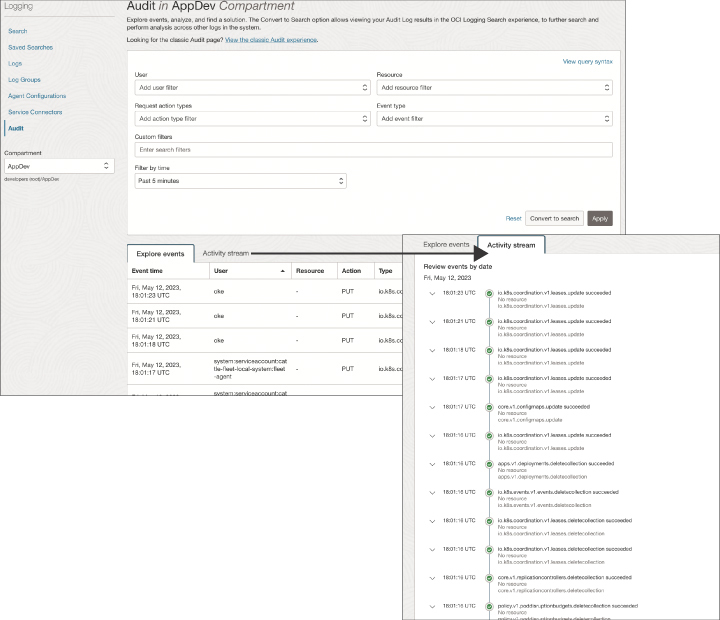
* **Identify and investigate security incidents:** Audit logs provide a valuable source of information for detecting and investigating potential security breaches or unauthorized access attempts.
* **Ensure compliance:** Many industry regulations and standards, such as PCI-DSS and HIPAA, require organizations to maintain comprehensive audit logs to demonstrate compliance with security and privacy requirements.
* **Support incident response and forensics:** If a security incident occurs, audit logs serve as a valuable resource for understanding the sequence of events, identifying the root cause, and conducting forensic analysis.

On OKE, all Kubernetes audit events are made available in the OCI Audit service. It’s crucial to monitor user and application activity on your Kubernetes cluster, to detect any unusual behavior or security breaches. This service provides a unified overview of all user activity across your applications on OCI. You can easily spot security incidents by monitoring successful and unsuccessful login attempts, such as identifying whether your cluster is under attack. Additionally, you can link Kubernetes audit events to other audit events in your OCI tenancy, such as updates to your clusters or resources.

To visualize the logs, you have two standard interfaces on the OCI console: Audit Events,[**3**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_3a) shown in [Figure 6-9](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig09), and Audit Logs,[**4**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_4a) shown in [Figure 6-10](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig10). The information is also available for other tools, including OCI Logging Analytics and third-party tools such as Datadog.



**Figure 6-9** Example of Audit Events from OKE API Server Audit Logs



**Figure 6-10** Example Activity Stream of the Audit Events from the Kubernetes API

More information on the OCI Audit service can be found in [Chapter 8](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch08.xhtml#ch08), “[Observability](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch08.xhtml#ch08).”

**Security Zones**

Oracle Cloud Infrastructure (OCI) security zones offer a comprehensive approach to securing workloads and data in the cloud. A security zone is a logically isolated and self-contained compartment within OCI that provides enhanced security and regulatory compliance capabilities. The following list describes the benefits of using OCI security zones:

* **Enhanced security isolation:** OCI security zones provide a high level of security isolation between different compartments within an OCI tenancy. Each security zone operates as an independent security boundary, allowing organizations to isolate sensitive workloads and data from other parts of their cloud infrastructure. This isolation helps mitigate the risk of lateral movement and unauthorized access, protecting critical assets from potential security threats.
* **Regulatory compliance:** Security and compliance are top priorities for many organizations. OCI security zones are designed to meet the stringent security and regulatory requirements of various industries, such as finance, healthcare, and government. These zones incorporate security controls and practices that align with industry standards, including data sovereignty, data residency, and compliance certifications. By leveraging security zones, organizations can address specific regulatory requirements and confidently deploy workloads that require adherence to strict compliance frameworks.
* **Network segmentation:** Network segmentation is a crucial aspect of cloud security. OCI security zones offer granular control over network traffic flow, allowing organizations to define and enforce access policies between different security zones. This segmentation helps limit the lateral movement of threats within the cloud environment, minimizing the impact of potential security breaches. Administrators can define security rules and access controls specific to each security zone, ensuring that only authorized traffic is allowed to enter or leave the zone.
* **Enhanced access control:** OCI security zones provide fine-grained access controls, enabling organizations to enforce strict permissions and privileges within each zone. Administrators can define security policies and identity and access management (IAM) rules tailored to the specific requirements of each security zone. This level of access control ensures that only authorized users and applications have the necessary permissions to interact with resources within a given zone, reducing the risk of unauthorized access and data breaches.
* **Monitoring and logging:** Visibility into security events and activities is crucial for maintaining a secure cloud environment. OCI security zones offer robust monitoring and logging capabilities, providing organizations with detailed visibility into the security posture of each zone. Administrators can monitor network traffic, access logs, and security events within security zones, enabling them to promptly detect and respond to potential security incidents. OCI’s Logging Service and other monitoring tools can be integrated with security zones to provide real-time insights and proactive threat detection.
* **Disaster recovery and business continuity:** OCI security zones support disaster recovery and business continuity strategies by providing isolation and redundancy across different zones. Organizations can deploy resources and replicate data across multiple security zones within the same region or different regions. This redundancy ensures that, if a localized failure or disaster occurs, critical workloads and data will remain accessible and operational in a separate security zone, minimizing downtime and preserving business continuity.

**Network Security Groups (NSGs)**

Network security groups (NSGs) are essential components of the overall network security architecture within the Oracle Cloud ecosystem. They play a vital role in safeguarding cloud resources and protecting critical workloads from unauthorized access, network threats, and potential vulnerabilities. NSGs provide several benefits and features to secure your network infrastructure:

* **Network segmentation and isolation:** NSGs enable network segmentation by allowing administrators to define security rules at the subnet level. Administrators can restrict traffic flow between subnets or between on-premises networks and the cloud by specifying ingress and egress rules. This segmentation enhances security by isolating different components of an application, preventing unauthorized lateral movement within the network, and limiting the potential impact of security breaches.
* **Access control and allowlisting:** NSGs provide fine-grained control over network traffic by allowing administrators to define specific rules to allow or deny access based on IP addresses, ports, and protocols. This capability enables administrators to create allowlists of trusted sources, effectively blocking traffic from unknown or untrusted sources. By implementing access control policies through NSGs, organizations can enforce the principle of least privilege and reduce the attack surface.
* **Defense against external threats:** NSGs act as a first line of defense against external threats by filtering and blocking malicious traffic at the network level. Administrators can define rules to block common attack vectors, such as denial-of-service (DoS) attacks, port scanning, or brute-force attempts. By proactively preventing such attacks, NSGs contribute to maintaining the availability, integrity, and confidentiality of cloud resources.
* **Traffic monitoring and logging:** NSGs offer comprehensive logging and monitoring capabilities. They allow administrators to track network traffic patterns, monitor access attempts, and identify potential security incidents or anomalies. By analyzing NSG logs and leveraging security monitoring tools, organizations can proactively detect and respond to security threats, ensuring timely incident response and minimizing the impact of potential breaches.
* **Compliance and regulatory requirements:** Many industries and organizations have specific compliance and regulatory requirements concerning data protection, privacy, and security. NSGs can help meet these requirements by implementing security controls and enforcing network policies aligned with industry standards. They enable organizations to demonstrate due diligence and adherence to security best practices, facilitating compliance audits and ensuring a solid security posture.
* **Dynamic and scalable security:** OCI network security groups provide flexibility and scalability in managing security policies. Administrators can easily modify security rules to accommodate changing business needs, add or remove rules to reflect application updates, and scale security configurations alongside resource scaling. This dynamic nature of NSGs ensures that security measures remain effective even in dynamic cloud environments with evolving requirements.

NSGs are essential for Kubernetes deployments on OKE. They provide critical security capabilities, such as pod-to-pod communication security, protection against external threats, access control and segmentation, compliance adherence, monitoring and auditing capabilities, and dynamic scalability. By leveraging NSGs effectively, organizations can enhance the overall security posture of their OKE clusters and protect critical workloads and data.

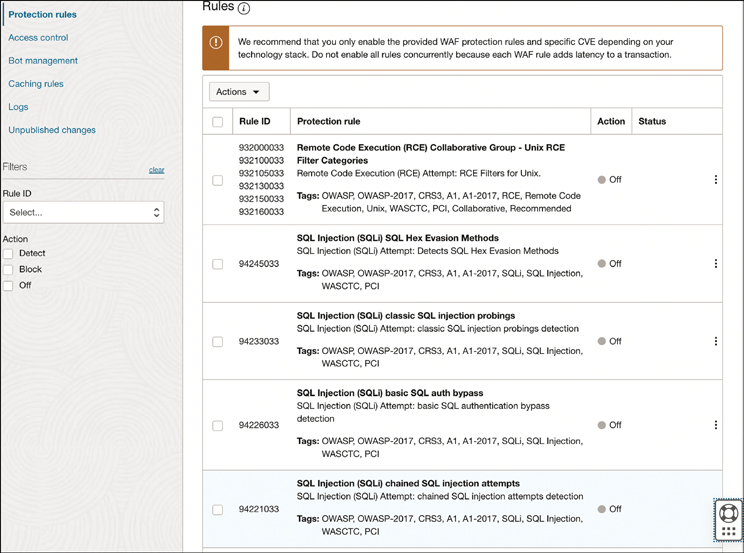
The section “[Setting Up NSG Rules and Security List Rules](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04lev3sec11),” in [Chapter 4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04), covers how to configure NSG rules and security list rules to secure OKE.

**Web Application Firewall (WAF)**

The OCI Web Application Firewall (WAF) can help secure web applications deployed on Oracle Kubernetes Engine (OKE) on first-mile security, offering robust protection against web-based attacks (including attacks to REST APIs), mitigating vulnerabilities, and enhancing the overall security posture of web applications.

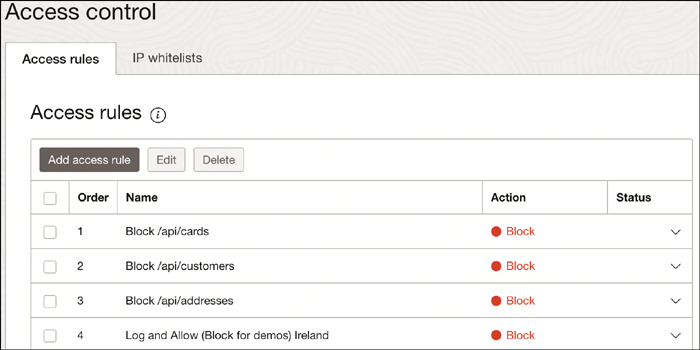
WAF operates at the application layer (Layer 7) of the network stack, enabling it to inspect and filter HTTP/S traffic directed at web applications running on OKE. By analyzing the content of web requests and responses, WAF can detect and prevent a wide range of common web-based attacks, such as cross-site scripting (XSS), SQL injection, and malicious file uploads. WAF also provides Layer 7 DDoS mitigation provided by Specialists in Oracle Cloud customer support. To use the mitigation service, you must allow the specialists to access your tenancy using IAM service policies. Layer 3/4 (L3/4) DDoS attack mitigation is part of the regular infrastructure service.

WAF also provides intelligent threat detection and prevention by using sophisticated threat-detection algorithms and security rules to identify and block malicious traffic. It leverages machine learning and behavioral analytics to detect anomalies, including advanced threats that traditional signature-based security mechanisms might not recognize. This proactive approach helps protect web applications from evolving attack techniques and zero-day vulnerabilities. It allows administrators to define and enforce custom security policies tailored to the specific needs of their web applications. These policies include rulesets, allowlist/blocklist configurations, and rate-limiting thresholds. By customizing the security policies, organizations can strike a balance between application security and legitimate traffic, ensuring protection without impacting the functionality and performance of the web application. WAF also provides a built-in set of security rules that cover common vulnerabilities and attack patterns. Oracle security experts regularly update and maintain these rules to address emerging threats. With automatic rule updates, organizations can protect their web applications against the latest attack vectors without manual intervention. [Figure 6-11](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig11) illustrates the recommended and preset WAF rules, with the option to turn them on or off.



**Figure 6-11** Predefined WAF Rules Available to Activate

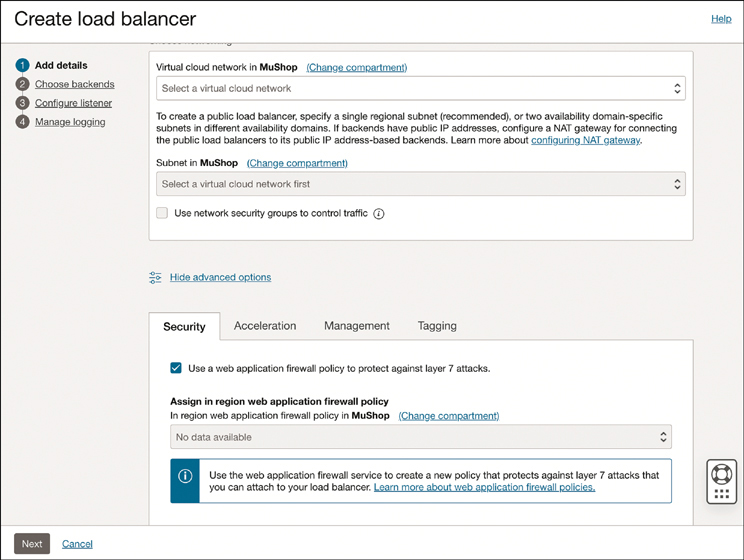
OCI WAF offers access rules (see [Figure 6-12](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig12)) that can intercept inbound web traffic before it reaches the OKE cluster. These rules can be applied to a specific Kubernetes workload with HTTP metadata (such as HTTP header, method, user agent, and URL with particular content) or a geographical location. The actions on the access rule include Log and Allow, Detect Only, Block, Redirect, Bypass, and Show CAPTCHA. This integration ensures that web application traffic is inspected and protected at the edge, reducing the risk of attacks reaching the underlying applications and infrastructure.



**Figure 6-12** WAF Access Rules

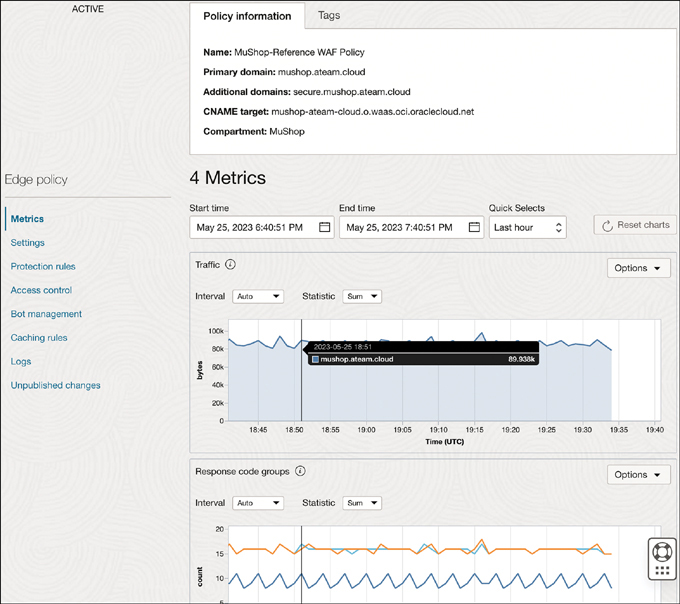
OCI WAF offers comprehensive monitoring capabilities, providing real-time visibility into web traffic and security events. Administrators can access detailed logs, metrics, and reports to gain insights into web application traffic patterns, attack attempts, and overall security posture. Additionally, OCI WAF can generate alerts and notifications when suspicious activity or potential security incidents are detected, enabling timely response and mitigation. It is also designed to handle high volumes of web traffic, and it can scale dynamically to meet application demands. OCI WAF operates across multiple availability domains within OCI, providing high availability and fault tolerance. This ensures that web applications are protected even during periods of high traffic or when infrastructure failures occur.

You can provision WAF in two different ways. One way, you can create a Load Balancer that is not managed by Kubernetes, enable the web application firewall under the Security tab, and configure it to be in front of the LoadBalancer[**5**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_5a) managed by a Kubernetes resource (Ingress Controller or Service Type LoadBalancer). [Figure 6-13](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig13) illustrates the option for the WAF as a Service (WaaS) configured directly on the Load Balancer.



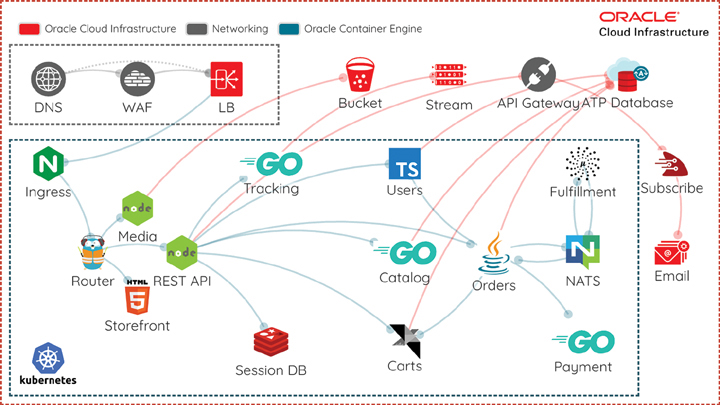
**Figure 6-13** OCI Load Balancer Creation UI, with the Option to Enable WAF Policy

Alternatively, you can create a new WAF edge policy[**6**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_6a) and configure the DNS to point to the endpoints you want to protect on Kubernetes, as illustrated in [Figure 6-14](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig14). This is the recommended option if you want full control on the rules and configurations.



**Figure 6-14** Example WAF Edge Policy

The architecture diagram in [Figure 6-15](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig15) shows an application that is protected by the Web Application Firewall. The diagram shows that the microservices within Kubernetes continue to function seamlessly despite the addition of the WAF. When a request is made, the DNS routes it to the WAF for validation before being directed to the Load Balancer managed by Kubernetes’ Ingress Controller. In a multicluster setting (not depicted), the OCI Traffic Manager’s Global Load Balancer collaborates with the WAF without any impact on Kubernetes cluster workloads. This means that the WAF can be added to any existing application and can be removed from the architecture without any code changes or deployments.



**Figure 6-15** Ecommerce App (MuShop) Diagram Example Showing Multiple OCI Services, Including WAF

**Network Firewall**

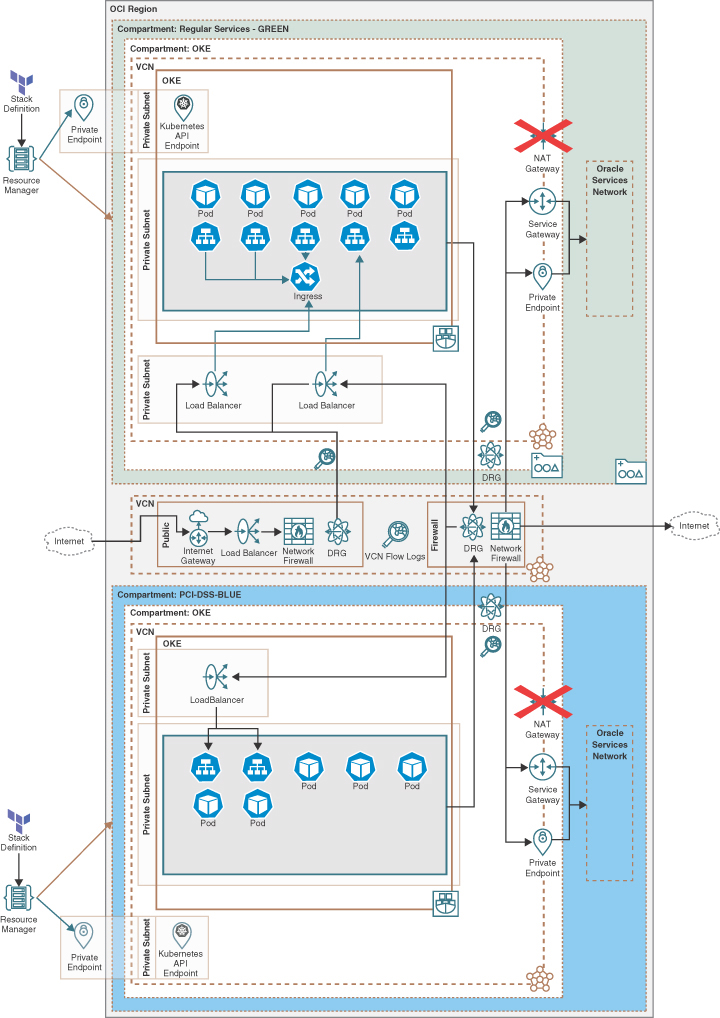
Network Firewall is a cloud-based network security service that allows customers to filter traffic based on IP addresses, ports, and protocols. A stateful firewall enables customers to define rules to allow or deny traffic to and from their resources in the cloud. Network Firewall is a fully managed service that is integrated with other Oracle Cloud Infrastructure services, such as the Compute, Load Balancing, and Kubernetes Engine services. It is combined with the OCI Logging service to give customers visibility into their network traffic. OCI Network Firewall is powered by Palo Alto Networks.

Network Firewall acts as a security control point between the Internet and the resources within an OCI virtual cloud network (VCN). The network firewall allows organizations to define and enforce granular access control policies based on source IP addresses, destination IP addresses, ports, and protocols. It helps protect applications and data from unauthorized access and network-based threats.

By combining the capabilities of OCI Network Firewall and OKE, organizations can establish a secure environment for their Kubernetes deployments. The integration between these services enables the following security measures:

* **Network segmentation:** OCI Network Firewall allows organizations to define security rules that control traffic flow between different VCNs, subnets, and resources. Administrators can enforce network segmentation by properly configuring the firewall rules and isolating OKE clusters from other resources within the VCN. This ensures that only authorized traffic can reach the Kubernetes clusters.
* **Access control policies:** OCI Network Firewall provides fine-grained access control policies that can be applied to inbound and outbound traffic. By defining firewall rules based on source IP addresses, destination IP addresses, ports, and protocols, organizations can restrict access to OKE clusters, allowing only trusted sources to communicate with the Kubernetes API server and worker nodes. This helps prevent unauthorized access and protects against potential attacks.
* **Application layer security:** Beyond network-level security, OCI Network Firewall allows organizations to define rules based on specific application protocols and payloads. This enables deep packet inspection and the capability to detect and block malicious activities at the application layer. Organizations can identify and block any unauthorized or suspicious activities by defining rules that match the traffic patterns of Kubernetes API calls and container communications.
* **Logging and monitoring:** OCI Network Firewall provides detailed logging and monitoring capabilities, allowing organizations to track and analyze network traffic. By integrating with OCI logging and monitoring services, administrators can gain visibility into network activities, detect anomalies, and respond to security incidents promptly.

[Figure 6-16](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig16) illustrates an example production architecture using the OCI Network Firewall service to manage all traffic into and out of a multicluster deployment.



**Figure 6-16** Multicluster Deployment with Network Firewall Architecture

**Allowed Registries**

Allowed registries refer to the container image registries that are permitted to be used within a cluster. These registries are the trusted sources from which Kubernetes nodes can pull container images for deploying and running applications. Using allowed registries alone does not ensure that only approved images can be used. To accomplish this, you need controllers acting on Kubernetes.

Controlling the allowed registries helps enforce security and compliance policies, ensuring that only approved and verified images are used in the cluster. To enforce allowed registries, you can use admission controllers that intercept registry requests for validation before the Kubernetes API Server commits them.

One recommended option is to use the Open Policy Agent (OPA) to create a policy that denies resources to container registries that do not match specific registries.

[Listing 6-2](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_2) shows an example of OPA configuration using Rego to deny pods in a specific namespace from using container registries that do not match a specific registry.

**Listing 6-2** Example of OPA Policy Written in Rego, Allowing Only ocir.io Registries on the Cluster

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0264-01a)

package admission

import data.k8s.matches

deny[{

"id": "container-image-allowlist",

"resource": {

"kind": "pods",

"namespace": namespace,

"name": name

},

"resolution": {"message": msg},

}] {

matches[["pods", namespace, name, matched\_pod]]

container = matched\_pod.spec.containers[\_]

not re\_match("^ocir.io/.+$", container.image) # The actual validation

msg := sprintf("invalid container registry image %q", [container.image])

}

The Rego Policy file can be directly loaded into OPA Gatekeeper as a configmap or as a ConstraintTemplate, illustrated in [Listing 6-3](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_3).

**Listing 6-3** OPA Gatekeeper Manifest with Rego Policies

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0264-02a)

apiVersion: templates.gatekeeper.sh/v1

kind: ConstraintTemplate

metadata:

name: k8sallowedrepos

spec:

crd:

spec:

names:

kind: K8sAllowedRepos

validation:

openAPIV3Schema:

type: object

properties:

repos:

type: array

items:

type: string

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8sallowedrepos

violation[{"msg": msg}] {

container := input.review.object.spec.containers[\_]

satisfied := [good | repo = input.parameters.repos[\_] ; good =

startswith(container.image, repo)]

not any(satisfied)

msg := sprintf("container <%v> has an invalid image repo <%v>, allowed

repos are %v", [container.name, container.image, input.parameters.repos])

}

violation[{"msg": msg}] {

container := input.review.object.spec.initContainers[\_]

satisfied := [good | repo = input.parameters.repos[\_] ; good =

startswith(container.image, repo)]

not any(satisfied)

msg := sprintf("container <%v> has an invalid image repo <%v>, allowed

repos are %v", [container.name, container.image, input.parameters.repos])

}

After the new ConstraintTemplate creation, the new constraint can be consumed and configured, as illustrated in [Listing 6-4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_4).

**Listing 6-4** Example of yaml manifest to consume the Gatekeeper manifest ConstraintTemplate

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0265-01a)

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: K8sAllowedRepos

metadata:

name: allowed-container-registries

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Pod"]

namespaces:

- "production"

parameters:

repos:

- "ocir.io/tenancy/namespace/\*"

- "ghcr.io/account/\*"

The section “[Open Policy Agent (OPA)](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06lev2sec19),” later in this chapter, provides more information about OPA, OPA gatekeeper, and OPA Rules/Rego.

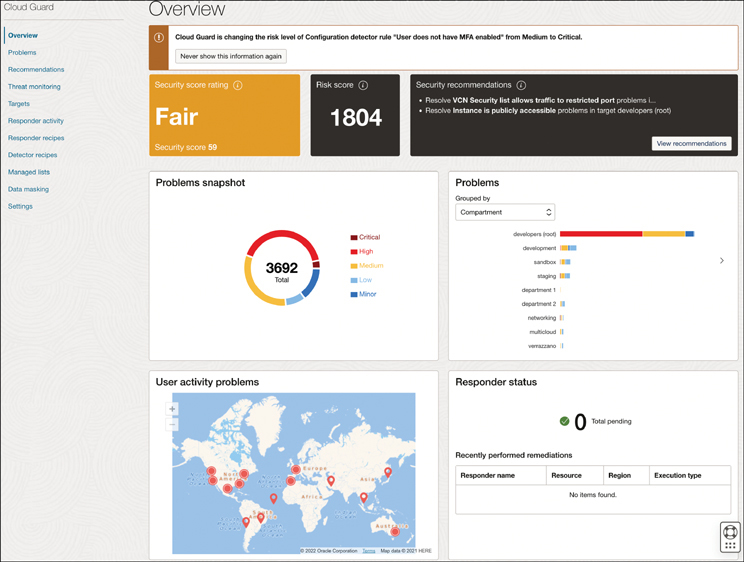
Controlling the allowed registries on OKE ensures that container images used within the cluster come from trusted and approved sources. This is one step in enforcing security and compliance, preventing unapproved sources, promoting standardization, optimizing resources, and offering configuration flexibility for a secure and controlled deployment environment for containerized applications.

**Cloud Guard**

OCI Cloud Guard is a comprehensive, unified security and compliance monitoring service. It helps organizations proactively identify, prioritize, and resolve security issues across their OCI environments, including OKE. By leveraging machine learning and AI technologies, Cloud Guard continuously analyzes telemetry data and configuration settings to detect potential security risks, threats, misconfigurations, and compliance violations. Cloud Guard is always active and evaluating the default rules. You can add detectors and responder recipes to extend the feature set for Cloud Guard to have custom notifications and monitor your own custom conditions for your infrastructure security posture. The key features of OCI Cloud Guard include the following:

* **Security monitoring:** Cloud Guard continuously monitors the security posture of OCI resources, including compute instances, storage, networks, and databases, to identify security vulnerabilities and threats.
* **Automated security policies:** Predefined security policies are provided, based on industry best practices and regulatory standards. These policies can be customized or extended to align with specific security requirements.
* **Real-time notifications:** Cloud Guard sends real-time alerts and notifications when security incidents are detected, enabling prompt action and mitigation.
* **Threat intelligence integration:** This feature integrates with Oracle’s Threat Intelligence service, which offers up-to-date threat intelligence feeds, enhancing the detection and response capabilities of Cloud Guard.
* **Compliance monitoring:** This feature provides built-in compliance frameworks and checks, such as CIS benchmarks, to ensure adherence to security standards and regulatory requirements.
* **Security recommendations:** Cloud Guard provides actionable recommendations to address security issues and guides users on best practices for securing their OCI resources.

[Figure 6-17](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig17) shows Cloud Guard in action, with its dashboard showing the security score, the risk score, and an overview of the security posture for the entire tenancy. Administrators can use this reporting to examine and analyze the data, to quickly zone in on the resources that are security threats and/or challenges. Administrators then can create detector and responder recipes to automate how certain conditions are detected and addressed.



**Figure 6-17** OCI Cloud Guard Dashboard Overview

**Hardening Containers and OKE Worker Nodes**

Containers and OKE worker nodes are critical components of a Kubernetes cluster that attackers frequently target. Containers are designed to be lightweight and portable, but they can also introduce security risks if they are not properly secured. Kubernetes worker nodes are the compute instances in a cluster that are responsible for running containers; if these nodes are compromised, an attacker can gain access to the entire cluster.

Hardening measures, such as limiting container privileges, implementing RBAC, configuring network security, and monitoring Kubernetes cluster activity, can help prevent attacks and detect malicious activity.

**Container Scanning**

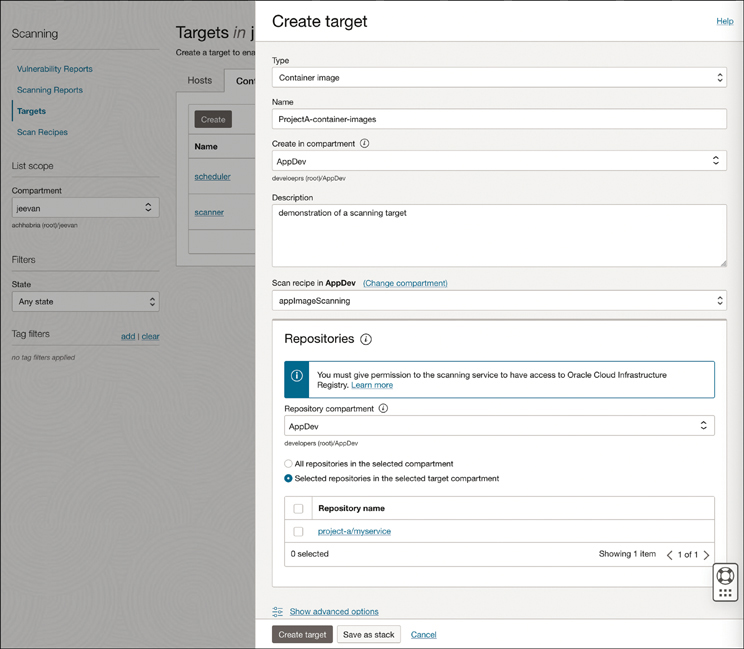
Containers are essential to modern application development because they provide a lightweight, scalable, and portable way to package and deploy applications. However, container images can also contain security vulnerabilities that attackers can exploit. In a Kubernetes environment, where containers are managed and orchestrated by Kubernetes, it is critical to scan container images before deployment, to identify any security vulnerabilities.

Container scanning tools can detect various security issues, including known vulnerabilities in the software components and libraries used in the container image, misconfigurations in the container, and other security-related issues. By detecting and addressing these issues, container scanning can significantly reduce the risk of security breaches in Kubernetes environments.

**How to Implement Container Scanning on OKE**

Container scanning can be implemented on OKE using various tools and techniques. These are some of the popular approaches:

* **OCIR container image scanning:** The Oracle Cloud Infrastructure Registry (OCIR) enables users or systems to push container images to repositories and enable the scanning of container images stored in the OCIR for published security vulnerabilities in the publicly available Common Vulnerabilities and Exposures (CVE) database. When repository scanning is enabled, the OCI Vulnerability Scanning service scans any images that you push into the repository and any images that are already present. Repositories with scanning enabled are automatically rescanned when new vulnerabilities are added to the list of threats. For every scanned image you can view the scan results, the risk level for each scan, the description of each vulnerability, and the link to the CVE database. [Figure 6-18](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig18) illustrates the creation of a target for Container Image Scanning, with the option to select the scanning recipe and the repositories that will be automatically scanned.



**Figure 6-18** OCIR Container Image Scanning Target Creation Screen

* **External container image scanning for container repositories:** Kubernetes can integrate with container image scanning tools such as Aqua Security, Twistlock, NeuVector, and Alcide, which scan container images for security vulnerabilities before they are deployed. This is important if you are not using OCIR or if you want to check for container image vulnerabilities from external repositories.
* **Kubernetes resources image scanning:** Some tools integrate into OKE to check for vulnerabilities on the container images defined on Kubernetes resources such as deployments, StatefulSets, and DaemonSets. Tools such as Snyk and Sysdig provide this kind of service. Quickstarts to deploy these solutions are available on the Oracle Quickstarts repo. Here are some examples:
  + Snyk.io (<https://snyk.io/>)[**7**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_7a)
  + Sysdig.com (<https://sysdig.com/>)[**8**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_8a)
* **Continuous integration/continuous deployment (CI/CD) pipelines:** CI/CD pipelines can include container scanning as part of the automated build process, to ensure that container images are scanned before deployment.
* **Kubernetes admission controllers:** Admission controllers enforce policies on Kubernetes objects before they are created or modified. By using admission controllers, organizations can implement policies that require container images to be scanned before deployment. OKE supports the use of the ImagePolicyWebhook admission controllers, to allow only images that pass the enforced rule.

**Best Practices for Kubernetes Container Scanning**

The following are some best practices to follow when implementing container scanning on OKE:

* **Enable OCIR image scanning:** Enabling image scanning for container images hosted on the ocir.io is the first step, allowing better control on the same infrastructure that is running the OKE.
* **Use multiple scanning tools:** Running multiple container scanning tools increases the likelihood of detecting security vulnerabilities. Each tool has strengths and weaknesses, so using multiple tools can provide a comprehensive scan of container images.
* **Implement scanning in the CI/CD pipeline:** Implement container scanning as part of the CI/CD pipeline to ensure that container images are scanned before deployment. This helps catch vulnerabilities early in the development process, reducing the risk of security breaches.
* **Monitor scanning results:** Monitor the results of container scanning to identify trends and patterns. This helps identify potential risk areas and allows organizations to take appropriate measures to address them.
* **Stay up to date:** Keep scanning tools and libraries up to date, to ensure that they can detect the latest security vulnerabilities. Container scanning tools should be updated regularly to ensure that they can detect newly discovered vulnerabilities.

Container scanning is an essential security practice for Kubernetes environments. By implementing container scanning on OKE, organizations can identify and mitigate potential security vulnerabilities in container images before deployment. Container scanning tools can detect various security issues, including known vulnerabilities, misconfigurations, and other security-related issues. By following best practices, organizations can ensure that container scanning is implemented effectively and efficiently in Kubernetes environments, reducing the risk of security breaches.

**Container Image Signing**

To enhance your runtime security, you can also sign your container images with cryptograph keys. You can configure OKE to pull container images that are signed and its signature verified. This ensures that only signed images that have not been tampered with can be deployed to OKE. The section “[Enabling Image Signature Verification](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04lev4sec2),” in [Chapter 4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04), has more details on container image signing.

**Center for Internet Security (CIS) Kubernetes Benchmarks**

The Center for Internet Security (CIS) is a community-driven nonprofit organization that provides many free and paid resources to improve IT security. The CIS Kubernetes Benchmark is a set of security recommendations designed to help organizations secure their Kubernetes deployments. This section provides an overview of the CIS Kubernetes Benchmark and shows how to implement its recommendations to secure an OKE cluster. You also see practical examples and best practices to help organizations comply with the CIS Kubernetes Benchmark. A free downloadable benchmark can be found at [www.cisecurity.org/benchmark/kubernetes](http://www.cisecurity.org/benchmark/kubernetes).[**9**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_9a) For OKE, the benchmark is named Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE). On this benchmark, you will find a comprehensive set of best practices and security controls designed to help organizations secure their Kubernetes deployments; it also provides guidelines for securing the Kubernetes worker nodes and some Kubernetes control plane components.

The benchmark covers a wide range of security controls, including these:

* Cluster configuration
* Network security
* Authentication and authorization
* Cluster hardening
* Logging and auditing
* Secure communication between Kubernetes components
* Pod security
* Node security

The benchmark can be consumed as a reading guide or can be made available for tools such as CIS-CAT Pro and Kube-bench to automate security checks and remediations. References on both tools can be found in the “[Supporting Tools](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06lev1sec7)” section.

The CIS Benchmark has two levels for OKE. Level 1 has practical recommendations, and the OKE Control Plane follows what is most beneficial. The minimum recommendations necessary for the worker nodes is applied for a secure node without impeding the usage and performance. The customer is responsible for hardening the worker nodes to the next level. Level 2 extends Level 1 and is intended for environments or use cases in which security is the most important concern. Applying Level 2 recommendations can affect the Kubernetes functionality and performance.

One of the primary and most basic purposes of the CIS Kubernetes Benchmark is to ensure that your cluster is configured with secure settings. This includes enforcing TLS communication for all components, disabling insecure communication protocols, and disabling anonymous access. Similarly, the benchmark evaluates the network security aspects of your cluster. Kubernetes provides a rich set of network security features that can be leveraged to secure your cluster. For example, you can configure network policies to control traffic between pods, implement network segmentation, and enforce secure communication over the network. The benchmark can provide feedback on the current networking configuration in your cluster and best-practice suggestions to improve it. It also evaluates the security for the communication between Kubernetes components. Kubernetes components communicate with each other using various protocols, and it is important to ensure that these communications are secure. The CIS Kubernetes Benchmark provides recommendations for securing communication between Kubernetes components, including configuring Transport Layer Security (TLS) and using secure communication protocols.

The benchmark also evaluates the Kubernetes authentication and authorization configuration. Kubernetes provides several authentication and authorization mechanisms, such as role-based access control (RBAC), that can be used to control access to the Kubernetes API server and other cluster components. The CIS Kubernetes Benchmark provides recommendations for implementing secure authentication and authorization practices.

Logging and auditing are critical for detecting and investigating security incidents in your Kubernetes cluster. The CIS Kubernetes Benchmark provides recommendations for configuring Kubernetes logging and auditing to ensure that all security-related events are captured.

The best practices for implementing the CIS Kubernetes Benchmark are as follows:

1. Regularly review and update your Kubernetes cluster configuration to ensure that it remains compliant with the CIS Kubernetes Benchmark.
2. Automate as much of the benchmark implementation as possible using tools such as Kubernetes manifests and configuration management tools.
3. Regularly review Kubernetes logs and audit logs to detect and investigate potential security incidents.
4. Use RBAC to control access to Kubernetes components and resources.
5. Use network segmentation to limit the impact of potential security breaches.

**Using SELinux with OKE**

SELinux is a Linux kernel security module that provides a mechanism for enforcing mandatory access control policies. This section covers how SELinux can be used with Kubernetes to enhance the security of your Kubernetes environment. SELinux is based on the principle of mandatory access control (MAC), which means that every process and user on the system is assigned a security context that determines its level of access to system resources. SELinux defines security contexts for files, processes, sockets, and other system resources. Each security context is associated with a set of rules that governs the interactions between processes and system resources.

SELinux has three modes of operation:

* **Enforcing:** SELinux enforces the security policies defined in the security context. Any attempt to violate the security policies results in an access denial.
* **Permissive:** SELinux logs all the security violations but does not enforce the policies.
* **Disabled:** SELinux is disabled, and no security policies are enforced.

By default, SELinux is enabled on OKE worker nodes and is set to run in permissive mode. When run in permissive mode, SELinux does not enforce access rules; it only performs logging. To enforce access rules, set SELinux to run in enforcing mode. When run in enforcing mode, SELinux blocks actions that are contrary to the policy and logs a corresponding event in the audit log. To set SELinux to run in enforcing mode, you must set a custom cloud-init for your worker nodes, as discussed in [Chapter 4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch04.xhtml#ch04).

Consider the cloud-init script in [Listing 6-5](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_5), which sets the SELinux policy on OKE nodes to enforcing mode.

**Listing 6-5** Setting the SELinux Policy on OKE Nodes to Enforcing

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0273-01a)

#!/bin/bash

curl --fail -H "Authorization: Bearer Oracle" -L0 http://169.254.169.254/opc/v2/

instance/metadata/oke\_init\_script | base64 --decode >/var/run/oke-init.sh

bash /var/run/oke-init.sh

setenforce 1

sed -i 's/^SELINUX=.\*/SELINUX=enforcing/' /etc/selinux/config

When SELinux is running in enforcing mode, you can define security contexts for Kubernetes objects such as pods, containers, and volumes. You can use the securityContext field in the pod specification to define the security context for a pod. The security context can include the SELinux options field, which allows you to specify the SELinux security context for the pod.

Consider the example in [Listing 6-6](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_6).

**Listing 6-6** SELinux Configuration in a Pod YAML Manifest

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0273-02a)

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

spec:

containers:

- name: nginx

image: nginx

securityContext:

seLinuxOptions:

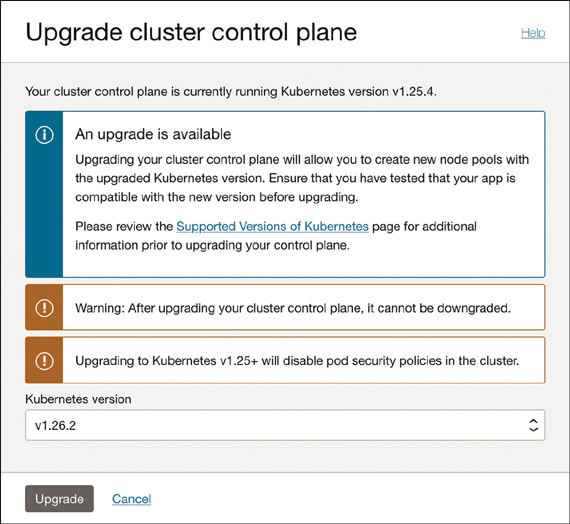
level: s0

categories: c123,c456

In this example, you define a pod named nginx-pod and set the security context for the nginx container using the seLinuxOptions field. The level option sets the SELinux security level to s0, and the category options c123 and c456 define the SELinux categories.

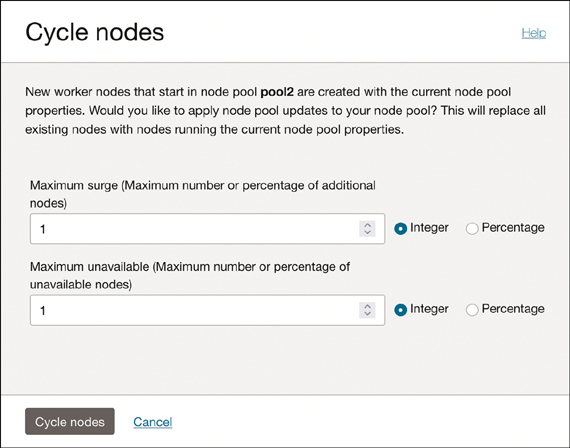
**Patching Worker Nodes**

OKE provides a managed control plane where the CVEs and other security tasks are done regularly and automated. Kubernetes version updates can be requested by customers and done through a button on the OCI Console, as illustrated in [Figure 6-19](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig19).



**Figure 6-19** Upgrading the Kubernetes Version

For the customer-managed worker nodes, the Kubernetes version updates are also done at the customer’s convenience. Still, the customer manages the operating system, and major upgrades need to be planned. OCI provides tools and agents to scan and report CVEs and other possible vulnerabilities. [Figure 6-20](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig20) illustrates the version upgrade lifecycle of the worker nodes on the same node pool.



**Figure 6-20** Worker Node Cycling for a Node Pool Can Safely Update Worker Nodes

The importance of regularly updating Kubernetes worker nodes and patching vulnerabilities to maintain security cannot be understated. New vulnerabilities are constantly being discovered, and attackers are always developing new methods of attack.

If the workload needs public worker nodes, monitoring and constant patching need to be mandatory and must be part of the organization’s DevSecOps strategy.

**Worker Nodes Limited Access**

Regarding security, limiting external access to the production environment is crucial. It is essential to strike a balance between granting end users appropriate access and avoiding unnecessary access. This involves several layers of protection, with the number of layers and their complexity varying, based on the sensitivity of the information being safeguarded.

Securing network traffic requires more than just making edge decisions. A series of firewalls working together must protect every connection between nodes, as well as intranode communication.

A dynamic CI/CD environment presents unique security challenges because of its constantly changing nature. To address this, scanning and verification tools should be integrated into the pipeline, with ongoing assessments carried out to quickly identify and resolve issues.

In addition to using container-specific security tools, such as SELinux, Kerberos, and SAML, noncontainer specific tools should be utilized to manage access at each layer, from the hardware to the application.

**Securing Your Workloads**

This section emphasizes the need for robust security measures, to protect Kubernetes workloads from potential vulnerabilities and attacks.

**Security Context**

Container security at scale can be challenging: The annual CNCF survey consistently ranks security as one of the top challenges for organizations that have adopted containers for most of their workloads. One particular aspect might contribute to this complexity: When you run containers, Linux operating system security concepts can bleed into the developer realm, whereas it previously existed only on the infrastructure and DevOps realms. This means that the change in the developer workflow, with developers now delivering an application container instead of an application package, is not just a packaging change. The new packaging format, containers, also needs to account for the security of the container environment. Thus, it is essential for developers who build and package their code to do so while still paying attention to the security posture. It is equally important for administrators to understand container settings and PodSpecs, to identify potentially misconfigured workloads that could pose a security risk.

Container runtimes and the underlying Linux features they are built upon, such as cgroups and namespaces, expose a plethora of options. For example, to run the processes inside a container as a specific user, Docker provides the --user flag. Within Kubernetes, the securityContext for a container or a pod determines how these are configured. Effectively using the features provided by the securityContext is an essential but often overlooked aspect. This could be because the options on offer are often unfamiliar to application developers, who largely focus on the app development stack, not on the Linux security best practices. This section describes the most common configuration options for securityContext; the Kubernetes documentation covers all the allowed configuration parameters.[**10**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_10a)

Kubernetes securityContext settings can be defined in both the spec.pod and spec.containers portions of the resource manifest (PodSpec and ContainerSpec APIs). Formally, the securityContext set at the PodSpec is called the PodSecurityContext, and the term SecurityContext refers to the configuration in the ContainerSpec. However, these are always applied at a container level, and specifying PodSecurityContext is a quick way to uniformly apply the same securityContext configuration to all containers running within the pod. If the securityContext setting is configured for both the PodSpec and the ContainerSpec, the configuration at the ContainerSpec takes precedence; you can think of this as the PodSecurityContext configuration being overridden by the SecurityContext configuration.

runAsNonRoot

As the name indicates, the runAsNonRoot setting set to true tells the kubelet to validate that the container is not running as root. Processes inside containers are run as root by default when no alternatives are provided. Because containers share a kernel, this also means that the process has the same access as the root account on the host. Containers use Linux features such as cgroups and namespaces to isolate the container; however, vulnerability exploits and misconfigurations can potentially give an attacker a path for escaping the container. As a general rule, there should be no reason for the vast majority of workloads to run as root. The runAsNonRoot configuration provides a mechanism to enforce this for your workloads.

As a best practice, your Dockerfile should create a user and set the default user as well, as shown in [Listing 6-7](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_7).

**Listing 6-7** Example Dockerfile Not Using Root User To Be Able To Be Deployed as runAsNonRoot on Kubernetes

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0276-01a)

RUN groupadd -g 10001 myapp && \

useradd -u 10000 -g myapp \

&& chown -R myapp:myapp /app

USER myapp:myapp

**Note**

The UID and GID specified are larger numbers, to avoid the possibility of these UIDs and GIDs existing on the host and inheriting privileges. Using a larger number—say, 10,000 or higher—reduces the chance of a user or group existing on the host with the same UID/GID.

Occasionally, the image you are running might not define a nonroot user, and you might not have direct control over the Dockerfile. This can often happen when the image comes from a vendor that no longer exists or that is refusing to add and set a default nonroot user. In these cases, as a last resort, you can create your own image based on the original by extending the Dockerfile, adding the new UID/GID, and setting those as the default.

To maintain a good security posture for the cluster, you set runAsNonRoot: true for your workloads. This ensures that the processes inside the container cannot use UID 0. If the runAsNonRoot is omitted, it has the same effect as setting it to false. If possible, you should always set runAsNonRoot explicitly, even if it is set to false for a workload that absolutely must run as root. This clearly communicates the intention and ensures that changes to the values can easily be audited.

runAsUser**and**runAsGroup

These configurations are used to run containers using the provided values for the UID and GID. They override the default UID and GID set within the container image and are often used in conjunction with the runAsNonRoot option. It is common for developers to provide an image with a nonroot user created but not set as the default. In this case, you can easily use the runAsUser and runAsGroup attributes to configure the included users. Specifying these values explicitly, even if they are set as the default in the image, makes them more visible and easier to track changes. These options should be used with care, and the values should not be set randomly. Many workloads have specific requirements on the UID and GID for the processes. For example, a container might need access to files that are accessible only by the user or group set by default in the image. In such cases, overriding these values could cause the application to malfunction.

As a best practice, do not rely on the runAsUser or runAsGroup settings to set the container processes’ UID and GID: They could cause incompatibilities or could potentially be removed later. Instead, the user and group should be created within the container image, and runAsUser or runAsGroup should be used to explicitly state the UID and GIDs, along with setting runAsNonRoot to true.

readOnlyRootFilesystem

If you have a stateful workload that needs to write data, you can take advantage of the storage mechanisms in Kubernetes, including ephemeral volumes and persistent volumes. Alternatively, your application might store state in a database. Writes to the container’s filesystem are made to the container’s top writable layer. The data is ephemeral and is not easily portable across nodes; any data written locally on the container will not persist when the pod is restarted. For this reason, containers in your workload generally do not need the capability to write to the container filesystem. However, the container filesystem opens some security concerns. If attackers are able to gain access to the container, they can update the configuration of the container when the container filesystem is writable. For example, an attacker could install new software packages within the container and open up new attack vectors. To avoid this, you can set the container’s root filesystem to be read-only. Note that this attribute is available only on the SecurityContext, not on the PodSecurityContext.

privileged

Setting the privileged attribute to true runs the container in privileged mode, which is essentially equivalent to running as root on the host. The default value for this attribute is false; it is available only on the SecurityContext, not on the PodSecurityContext. As a best practice, you should consider explicitly setting this to false so that the intent of running a container without privileged mode is clear and changes can be easily tracked.

capabilities

POSIX capabilities are a kernel feature that allows for granular control over the kernel calls a process can make. The idea is to group all the privileged kernel calls into related categories and then assign processes only to the categories they need. With the capabilities attribute, you have the capability to drop or add capabilities to containers. The default set of capabilities that a container gets is determined by the container runtime. Using the capabilities.add and capabilities.drop attributes, you can provide one or more capabilities as a comma-separated list. You use the -all keyword to cover all capabilities. As a best practice, consider dropping all capabilities and adding only the capabilities that your workload actually needs, as shown in [Listing 6-8](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_8).

**Listing 6-8** Example of SecurityContext.capabilities on a Deployment YAML Manifest

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0278-01a)

securityContext:

capabilities:

drop:

- all

add: ["NET\_ADMIN", "SYS\_TIME"]

**syscalls and seccomp**

System calls are the fundamental interface between user-level programs and the operating system kernel, often abbreviated as syscalls. They provide a means for applications to request services from the operating system, such as file operations, network communication, process management, and more. Every time an application needs to interact with the underlying system, it uses syscalls to perform the necessary operations.

**Container Isolation and syscalls**

Containerization technologies such as Docker and CRI-O leverage the underlying Linux kernel’s features, such as namespaces and cgroups, to provide isolation between containers. syscalls play a crucial role in enabling this isolation by acting as the gatekeepers between the container and the host operating system. When a containerized application makes a syscall, it is intercepted by the container runtime, such as Docker or containerd, before it reaches the underlying kernel. The runtime validates the syscall, ensuring that it complies with the container’s defined isolation policies. This validation includes checking whether the syscall is allowed, restricted, or prohibited, based on the container’s configuration.

**seccomp and syscall Filtering**

Kubernetes employs several mechanisms to control and filter syscalls. One such mechanism is seccomp, or secure computing mode, which enables administrators to define a policy for a container that specifies which syscalls are permitted. Restricting the set of allowed syscalls significantly reduces the attack surface of a container and enhances its security. seccomp works by leveraging Linux’s seccomp-bpf (Berkeley Packet Filter) mechanism, which filters syscalls using a predefined filter expression. This expression can be customized to allowlist or blocklist specific syscalls, based on the container’s requirements. Kubernetes provides a default seccomp profile, but users can also define their own profiles, for fine-grained control.

The example in [Listing 6-9](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_9) uses a seccomp profile for syscall auditing.

**Listing 6-9** Example of a YAML Pod Manifest File Using securityContext.seccompProfile

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0279-01a)

apiVersion: v1

kind: Pod

metadata:

name: audit-pod

labels:

app: audit-pod

spec:

securityContext:

seccompProfile:

type: Localhost

localhostProfile: profiles/audit.json

containers:

- name: test-container

image: hashicorp/http-echo:0.2.3

args:

- "-text=just made some syscalls!"

securityContext:

allowPrivilegeEscalation: false

By default, seccomp allows only restricting specific system calls, such as read(), write(), exit(), and sigreturn(). All other system calls are prohibited.

Mode 2 enables the utilization of BPF and eBPF filters to determine which system calls are permitted. After the eBPF program is created and installed into the kernel, all system calls pass through the filter. Additional features and generalizations of BPF have been added throughout the years.

**Capabilities and Privileged Containers**

In addition to seccomp, Kubernetes employs capabilities to further control container permissions. Capabilities allow containers to perform specific privileged operations, such as changing network settings or accessing kernel-level resources. By default, Kubernetes drops most capabilities, ensuring that containers run with the minimum necessary privileges; however, administrators can selectively enable or disable capabilities, based on the requirements of their applications.

**Syscall Interception and User-Space Proxies**

In some cases, Kubernetes deployments include user-space proxies such as Envoy (used by Istio), for traffic management and observability. These proxies often intercept network-related syscalls to enable advanced networking features such as service discovery, load balancing, and mutual TLS authentication. By intercepting syscalls, these proxies can modify network behavior at runtime, enabling a more flexible and dynamic networking environment for Kubernetes clusters.

**Open Policy Agent (OPA)**

Open Policy Agent (OPA) is a general-purpose policy engine that provides a flexible and policy-driven approach to enforce rules and regulations in Kubernetes deployments. OPA allows organizations to define and enforce policies such as access control, data filtering, and configuration validation across various domains. OPA relies on the declarative language Rego to express policies. These policies are expressed as rules and can be evaluated against data, to determine compliance.

**OPA Integration with Kubernetes**

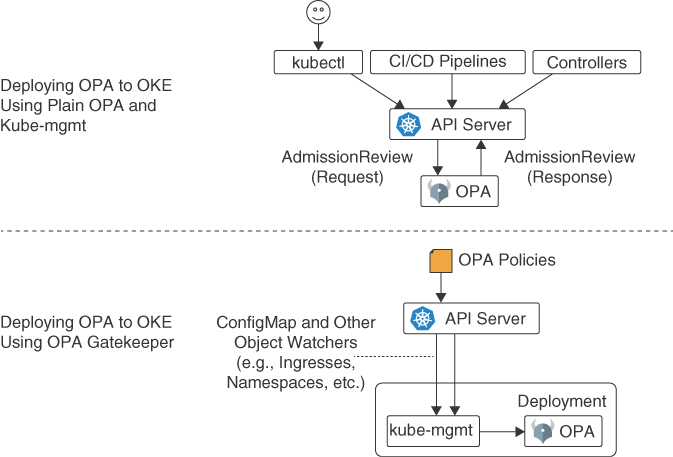
OPA can be integrated with Kubernetes in various ways to enforce policies across different aspects of the cluster:

* **Admission control:** OPA can be integrated with Kubernetes admission controllers to validate and mutate requests made to the cluster. Admission controllers intercept requests for creating or modifying Kubernetes resources and apply policy-based checks before allowing or rejecting them. OPA can evaluate admission policies using Rego and enforce custom policies for resource creation and modification.
* **Policy enforcement:** OPA can be used to enforce policies at runtime by integrating with dynamic admission control in Kubernetes. This enables fine-grained policy enforcement based on real-time data and context. For example, OPA can be used to enforce pod-to-pod communication policies, network policies, or even access control policies based on user roles and permissions.
* **Configuration management:** OPA can be used to manage and enforce configuration policies for Kubernetes resources. With OPA, organizations can define policies for resource configurations, such as ensuring that containers use specific versions, enforcing resource limits, or validating security-related configurations. OPA can evaluate the desired state against defined policies and reject or modify configurations that violate the policies.
* **Governance and compliance:** OPA can help organizations maintain governance and compliance within their Kubernetes environments. By defining policies that align with industry regulations or internal security standards, OPA can ensure that Kubernetes resources adhere to these policies. This includes policies for image scanning, secrets management, auditing, and more.

**Deploying OPA on OKE**

OPA integrates seamlessly with Kubernetes, enabling administrators to define and enforce policies specific to their Kubernetes deployments. This integration is typically achieved using the Kubernetes admission control mechanism, which intercepts API requests and evaluates them against OPA policies before allowing them to proceed. OPA can be deployed as a standalone service, integrated as a sidecar container alongside other application containers, or used as a library within an application. The chosen deployment method depends on the organization’s specific requirements and use cases.

You can deploy OPA to OKE in two ways, as illustrated in [Figure 6-21](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig21). The first is to use plain OPA and Kube-mgmt, which is enabled as the Kubernetes Admission Controller. This option is more complex and needs to make changes to the ValidationAdmissionController and some mutating controllers; it is not covered in this book. For more information on the supported admission controllers on OKE, check the documentation.[**11**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_11a)



**Figure 6-21** OPA Deployment Options

The second approach is to use OPA Gatekeeper, an extension of the Open Policy Agent (OPA) project. OPA Gatekeeper offers a first-class integration between OPA and Kubernetes for policy enforcement.

OPA Gatekeeper includes additional features on top of the basic OPA:

* An extensible, parameterized policy library
* Native Kubernetes CRDs for instantiating the policy library (constraints)
* Native Kubernetes CRDs for extending the policy library (constraint templates)
* Audit functionality

The upcoming section “[OPA Gatekeeper](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06lev2sec20)” provides additional details.

**Defining Policies with Rego**

Rego is the declarative language used by OPA to express policies. It provides a flexible syntax for defining rules and constraints that govern Kubernetes resources. Rego policies can be written to validate resource configurations, enforce naming conventions, restrict image usage, ensure compliance with security standards, and perform custom checks tailored to the organization’s requirements.

[Listing 6-10](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_10) provides an example of a Rego declarative policy.

**Listing 6-10** Example Rego Declarative Policy

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0282-01a)

package kubernetes.admission

import future.keywords

deny contains msg if {

input.request.kind.kind == "Pod"

some container in input.request.object.spec.containers

image := container.image

not startswith(image, "hooli.com/")

msg := sprintf("image '%s' comes from untrusted registry", [image])

}

**Policy Decision and Enforcement**

A request to the Kubernetes API server triggers the admission control webhook configured with OPA. The webhook forwards the request to OPA, which evaluates the request against the defined policies in Rego. OPA returns a policy decision to the admission control webhook, indicating whether the request is allowed or denied, based on policy evaluation. The webhook then enforces the decision by either allowing or rejecting the request.

**Dynamic Policy Updates and Live Configuration Changes**

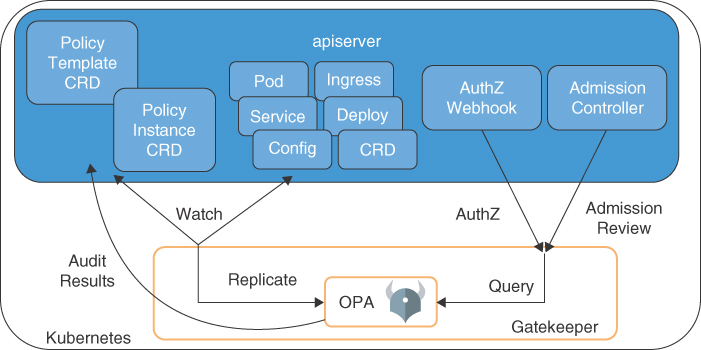
One of the advantages of using OPA with Kubernetes is its capability to dynamically update policies without requiring a cluster restart. This dynamic behavior enables administrators to make policy changes on the fly and adapt to evolving requirements. OPA can fetch policy updates from external sources or integrate them with configuration management tools, ensuring that policy changes are propagated efficiently throughout the cluster.

**Auditing and Compliance Reporting**

OPA’s policy engine provides organizations with valuable insights into the compliance status of their Kubernetes deployments. Organizations can generate compliance reports and audit trails by logging policy decisions and evaluations. These reports help demonstrate adherence to regulatory standards and provide visibility into any potential policy violations or security gaps.

**OPA Gatekeeper**

OPA Gatekeeper is an admission controller for Kubernetes that integrates with OPA to enforce policies during resource creation and modification. It enables administrators to define and enforce policies using the Rego language, ensuring that Kubernetes resources comply with specific requirements and constraints (see [Figure 6-22](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig22)).



**Figure 6-22** OPA Gatekeeper

**Benefits of OPA Gatekeeper**

OPA Gatekeeper offers several benefits for policy enforcement in Kubernetes deployments:

* **Declarative policy enforcement:** Policies can be defined using the Rego language in a declarative manner, making it easier to express complex constraints and requirements.
* **Kubernetes-native integration:** OPA Gatekeeper is purpose-built for Kubernetes and integrates seamlessly into the admission control workflow, enforcing policies during resource creation and modification.
* **Policy as code:** Policies defined using OPA Gatekeeper are treated as code, allowing for version control, collaboration, and auditability. They can be stored alongside other Kubernetes manifests and managed using standard Git workflows.
* **Dynamic policy evaluation:** OPA Gatekeeper enables dynamic policy evaluation based on real-time data and context. Policies can reference external data sources, making it possible to enforce policies that depend on information from external systems or services.

**Deploying OPA Gatekeeper**

Deploying OPA Gatekeeper on the OKE cluster requires setting up the following components:

* **OPA Gatekeeper constraint templates:** Constraint templates are used to define the types of policies that OPA Gatekeeper can enforce. These templates specify the structure and parameters of the policies. Organizations can use built-in templates provided by the OPA Gatekeeper project or create custom templates based on their specific needs.
* **Constraints:** Constraints are instances of constraint templates that define specific policy rules. They specify the desired state and properties that Kubernetes resources should adhere to. Constraints are used to evaluate resources against policies during admission control.
* **OPA Gatekeeper controller:** The OPA Gatekeeper controller is responsible for managing constraint templates, constraints, and the policy evaluation process. It communicates with the OPA server to enforce policies and validate resources during admission control.

You can deploy OPA Gatekeeper using the manifest with predefined settings:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0284-01a)

kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/

gatekeeper/master/deploy/gatekeeper.yaml

Alternately, you can deploy OPA Gatekeeper using the Helm chart, as demonstrated here:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0284-02a)

helm repo add gatekeeper https://open-policy-agent.github.io/gatekeeper/

charts

helm upgrade --install gatekeeper/gatekeeper --name-template=gatekeeper

--namespace gatekeeper-system --create-namespace

**Enforcing Policies with OPA Gatekeeper**

After you deploy OPA Gatekeeper and set up the necessary components, you can define and enforce policies using constraint templates and constraints. The policy enforcement workflow typically involves the following steps:

Step 1.**Defining constraint templates:** Administrators define constraint templates using the Rego language. These templates specify the structure, parameters, and conditions of the policies to be enforced. The Rego language provides flexibility to express complex policies using logical operators, pattern matching, and iteration.

Step 2.**Creating constraints:** Based on the defined constraint templates, administrators create constraints that apply specific policies to Kubernetes resources. Constraints define the desired state and properties that resources must conform to. They are associated with constraint templates and are evaluated during admission control.

Step 3.**Evaluating admission control:** When a resource creation or modification request is made to the Kubernetes API server, the OPA Gatekeeper controller intercepts the request as part of the admission control process. It evaluates the resource against the defined constraints using the OPA server. If the resource violates any policies, then the request is rejected and the resource is not created or modified.

Step 4.**Continuously enforcing policy:** OPA Gatekeeper provides continuous policy enforcement by ensuring that resources remain compliant even after they are admitted. It periodically reevaluates existing resources against the defined policies, detecting and alerting any policy violations or drifts.

**Open Web Application Security Project (OWASP)**

The Open Web Application Security Project (OWASP) is a widely recognized nonprofit organization that focuses on improving the security of software applications.

OWASP provides a wealth of knowledge and resources to help organizations identify and address common security vulnerabilities in web applications and APIs. Its flagship project, the OWASP Top Ten, highlights the most critical security risks applications face. The OWASP community actively develops tools, guides, and best practices to help organizations build secure software.

Kubernetes introduces unique security challenges that organizations must address to ensure their containerized applications’ integrity and confidentiality. In a cloud native environment powered by Kubernetes, containers themselves must be properly secured. Organizations should ensure that container images are free from vulnerabilities and adhere to secure coding practices. OWASP guides secure container development and the use of secure base images.

Properly configuring Kubernetes clusters is critical for maintaining security as well. Misconfigured clusters can lead to unauthorized access, data breaches, and other security incidents. Organizations should follow Kubernetes security best practices, such as implementing strong authentication and authorization mechanisms, enabling encryption, and restricting access to sensitive resources.

OWASP guidance on secure network design can be applied to Kubernetes environments as well. This is because Kubernetes networking introduces additional challenges in securing communication between pods, services, and external resources. OWASP provides guidance in using network policies, ingress controllers, and secure communication protocols, such as Transport Layer Security (TLS), that are recommended to protect network traffic.

Similarly, OWASP recommendations on secure key management can be applied to Kubernetes secrets and other sensitive data. Managing sensitive information such as API keys, passwords, and certificates is crucial in Kubernetes. Organizations should follow the best secure secret management practices, including proper encryption, restricted access, and secure storage mechanisms.

**Leveraging OWASP Best Practices on OKE**

Some of the best practices and resources available to leverage OWASP guidance for OKE clusters include the following:

* **OWASP Top Ten:** The OWASP Top Ten list of common vulnerabilities and corresponding mitigation techniques (<https://owasp.org/www-project-kubernetes-top-ten/>)[**12**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_12a) is a valuable resource in securing web applications. Although Kubernetes is not a web application, many security risks identified by OWASP, such as injection attacks, broken authentication, and insecure access control, are still relevant in Kubernetes deployments. Organizations should review the OWASP Top Ten and apply relevant mitigations to their Kubernetes environments.
* **OWASP Secure Coding Practices:** OWASP provides a comprehensive set of secure coding practices that can be applied to containerized applications running on OKE. These practices cover areas such as input validation, output encoding, access control, and error handling. Adhering to these practices helps reduce the risk of common application-level vulnerabilities. Secure Coding Practices extend to OCI SDK usage with access to the infrastructure services.
* **OWASP tools and libraries:** OWASP offers a range of security tools and libraries that can be integrated into the CI/CD pipeline and runtime environment of Kubernetes applications. For example, tools such as OWASP Zed Attack Proxy (ZAP) can be used to perform security testing on containerized applications during development and deployment.
* **Continuous security testing and monitoring:** In addition to applying OWASP best practices, organizations should implement continuous security testing and monitoring in their Kubernetes environments. This includes vulnerability scanning of container images, runtime monitoring of cluster activities, and regular security assessments. The OWASP toolset provides several resources for security testing, including dependency checkers, vulnerability scanners, and secure code analysis tools. This chapter covered many resources integrated into the OWASP toolset and practices.

**Supporting Tools**

This section describes tools that were tested on OKE and can support the security activities and your DevSecOps strategy.

**External Container Scanning Tools**

Several tools are available for Kubernetes container scanning. These are some popular open-source tools:

* **Trivy:** Trivy is an open-source vulnerability scanner for containers. It can scan container images for vulnerabilities and generate a report. Trivy can be integrated with Kubernetes and can scan containers in a Kubernetes cluster.
* **Anchore:** Anchore is an open-source container image analysis tool. It can scan container images for vulnerabilities, configuration issues, and policy violations. Anchore can be integrated with Kubernetes and can be used to scan containers in a Kubernetes cluster.
* **Clair:** Clair is an open-source vulnerability scanner for containers. It can scan container images for vulnerabilities and generate a report. Clair can be integrated with Kubernetes and can be used to scan containers in a Kubernetes cluster.
* **Aqua Security:** Aqua Security provides a comprehensive container security platform, including container scanning, runtime protection, and compliance monitoring. Aqua Security can be integrated with Kubernetes and can be used to scan containers in a Kubernetes cluster.

**CIS-CAT Pro Assessor**

CIS-CAT Pro Assessor is a tool developed by the Center for Internet Security that automates assessing the security configurations of various technologies, including Kubernetes clusters. It uses CIS Benchmarks, consensus-based guidelines that provide recommendations for secure system configurations. Using CIS-CAT Pro Assessor, organizations can identify vulnerabilities, misconfigurations, and security weaknesses within their Kubernetes environments.

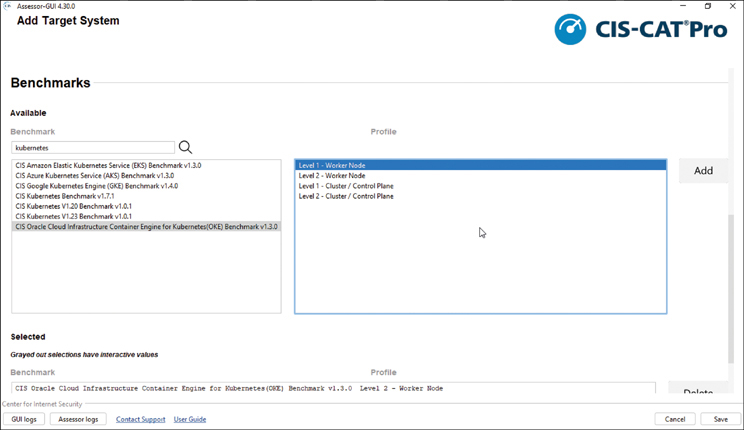
To leverage the advanced capabilities of CIS-CAT Pro Assessor with Kubernetes Benchmarks, follow these steps:

Step 1.**Download CIS-CAT Pro Assessor:** CIS-CAT Pro Assessor is a commercial tool that can be obtained from the Center for Internet Security. After acquiring the tool, follow the installation instructions provided by CIS.

Step 2.**Obtain the Kubernetes Benchmarks:** Download the CIS Kubernetes Benchmarks from the CIS website. The benchmarks are available in various formats, including YAML and JSON. Make sure you obtain the Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) Benchmark.

Step 3.**Configure CIS-CAT Pro Assessor:** When CIS-CAT Pro Assessor is installed, you need to configure it to work with Kubernetes Benchmarks. This involves specifying the location of the benchmark files and setting up any required parameters or options. The configuration file should contain information about your Kubernetes deployment, such as the API server URL, authentication credentials, and the location of the Kubernetes configuration file. OKE kubeconfig needs the oci-cli to be installed and configured for the same tenancy of the cluster, or using the --profile parameter if more than one profile is on the same machine.

Step 4.**Run the assessment:** Launch CIS-CAT Pro Assessor and initiate the assessment process by providing the necessary inputs, such as the target Kubernetes cluster information. The tool automatically evaluates the cluster’s security configurations against the CIS Kubernetes Benchmark. You can run the assessment using the command line or the graphical user interface (GUI), as shown in [Figure 6-23](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig23).



**Figure 6-23** CIS-CAT Pro Assessor GUI

[Listing 6-11](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_11) provides an example of running the assessment using the CIS-CAT CLI using OKE Benchmark.

**Listing 6-11** CIS-CAT CLI: Using the OKE Benchmark

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0289-01a)

./Assessor-CLI.sh -b benchmarks/CIS\_Oracle\_Cloud\_Infrastructure\_Container\_Engine\_

for\_Kubernetes(OKE)\_Benchmark\_v1.3.0-xccdf.xml

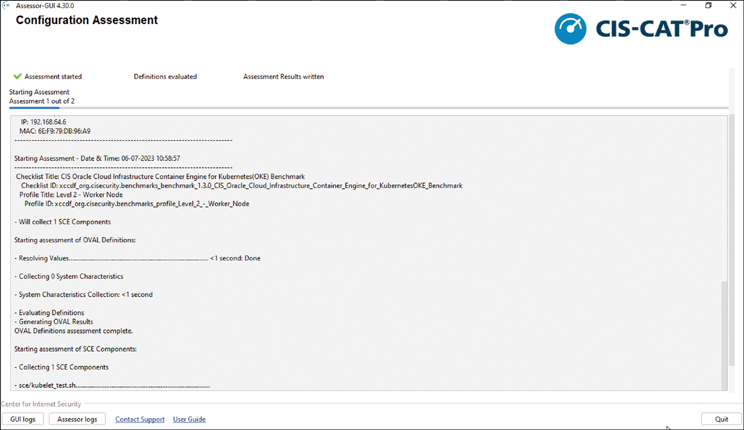
[Listing 6-12](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#list6_12) provides an example of running the assessment using the CIS-CAT CLI using a custom configuration file.

**Listing 6-12** CIS-CAT: Using Custom Configuration File

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0289-02a)

./Assessor-CLI.sh --config-xml /CIS/kubernetes\_assessment-configuration.xml

Step 5.**Review the assessment results:** CIS-CAT Pro Assessor generates a comprehensive report highlighting deviations from the recommended configurations outlined in the Kubernetes Benchmark (see [Figure 6-24](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig24)). Carefully review the report to identify areas that require attention or remediation.

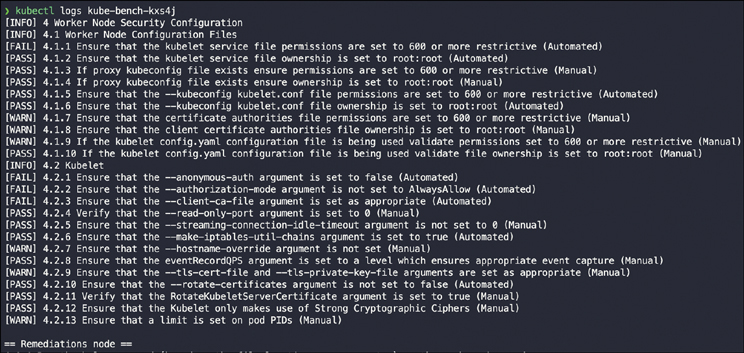


**Figure 6-24** CIS-CAT Pro Assessor Report

**kube-bench**

kube-bench is a security auditing tool developed by Aqua Security. It automates checking Kubernetes configurations against the CIS Kubernetes Benchmark. This benchmark provides a comprehensive set of best practices for securing Kubernetes deployments.

After running all the checks, kube-bench gives you a formatted output of FAIL, WARN, and PASS benchmarks. It supports various output formats, including console, JSON, and JUnit XML, making it easy to integrate with other tools and systems. [Figure 6-25](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig25) illustrates the log report of the Kubernetes job executed by kube-bench against the OKE.



**Figure 6-25** kube-bench Log Report from the Kubernetes Job Execution

Using kube-bench with OKE requires you to have cluster-admin access; you also must have deployed the provided job.yaml or installed the kube-bench CLI.

To deploy as a job using the generic Kubernetes Benchmark, run the following commands:

kubectl apply -f job.yaml

You can monitor the job’s progress:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0290-01a)

kubectl logs -f job/kube-bench

Additionally, you can retrieve the job’s results:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0290-02a)

kubectl logs job/kube-bench > kube-bench-results.txt

Running kube-bench using the CLI is simple as well:

kube-bench run --targets node

You can find the configurations for OKE in the book’s GitHub repository.

-- oke-1.4.0

|-- config.yaml

|-- controlplane.yaml

|-- managedservices.yaml

|-- master.yaml

|-- node.yaml

'-- policies.yaml

To include an OKE-specific benchmark, download the OKE benchmark from the book’s GitHub repository, copy it to the kube-bench/cfg directory, and run the following command:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0291-01a)

kube-bench run --benchmark oke-1.4.0

kube-bench provides detailed results indicating the compliance status of each check in the CIS Kubernetes Benchmark. It highlights failed or skipped checks, enabling you to identify potential security vulnerabilities or misconfigurations. Analyze the results carefully, taking appropriate actions to remediate any security issues discovered.

**AppArmor**

AppArmor is an alternative Linux Security Module (LSM) to SELinux. Support for it has been incorporated in the Linux kernel since 2006. It has been used by Oracle Linux, SUSE, Ubuntu, and other distributions. AppArmor supplements the traditional UNIX discretionary access control (DAC) model by providing mandatory access control (MAC). In addition to manually specifying profiles, AppArmor includes a learning mode, in which violations of the profile are logged but not prevented. This log can then be turned into a profile, based on the program’s typical behavior.

Consider some AppArmor facts:

* It allows administrators to associate a security profile to a program that restricts its capabilities
* It is sometimes considered easier to use than SELinux
* It is considered filesystem neutral (no security labels required)

For an AppArmor profile to be used by a pod, it must be available on the node where it is assigned. There is no native process for Kubernetes to load policies. As a result, you need to ensure that policies are loaded on every node where AppArmor-required pods are scheduled and that the scheduler is unaware of which nodes have profiles. Adding profiles can be done during node installation with a tool such as Ansible or Puppet, at least for some of the nodes. If only some nodes will have profiles installed, you can use a NodeSelector or a taint to ensure that the scheduler chooses the appropriate node. Another solution is to deploy a DaemonSet and allow the pod to modify the host and add profiles. This would assign the responsibility to the cluster administrators, if they are different from the administrators responsible for operating system configuration and security. If you need to disable AppArmor on the entire cluster (perhaps to troubleshoot), you pass the --feature-gates=AppArmor=false option.

AppArmor has several administrative utilities for monitoring and control (see [Table 6-1](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06tab01)).

**Table 6-1** AppArmor Administrative Utilities

| **Program** | **Use** |
| --- | --- |
| apparmor\_status | Shows the status of all profiles and processes with profiles. |
| apparmor\_notify | Shows a summary for AppArmor log messages. |
| complain | Sets a specified profile to complain mode. |
| enforce | Sets a specified profile to enforce mode. |
| disable | Unloads a specified profile from the current kernel and prevents it from being loaded upon system startup. |
| logprof | Scans the log file and suggests modifications to augment the existing profile if there are new AppArmor events that are not covered by it. If AppArmor is running, the updated profiles are reloaded and processes are set to run under their proper profiles. |
| easyprof | Helps set up a basic AppArmor profile for a program. |

On an Oracle Linux system, AppArmor has the following monitoring and controlling utilities:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0292-01a)

$ rpm -qil apparmor-utils | grep bin

/usr/bin/aa-easyprof

/usr/sbin/aa-audit

/usr/sbin/aa-autodep

/usr/sbin/aa-cleanprof

/usr/sbin/aa-complain

/usr/sbin/aa-decode

/usr/sbin/aa-disable

/usr/sbin/aa-enforce

/usr/sbin/aa-genprof

/usr/sbin/aa-logprof

/usr/sbin/aa-notify

/usr/sbin/aa-remove-unknown

/usr/sbin/aa-status

...

/usr/sbin/complain

/usr/sbin/decode

/usr/sbin/disable

/usr/sbin/enforce

Note that many of these utilities can be invoked with either their short or long names:

[Click here to view code image](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06_images.xhtml#f0292-02a)

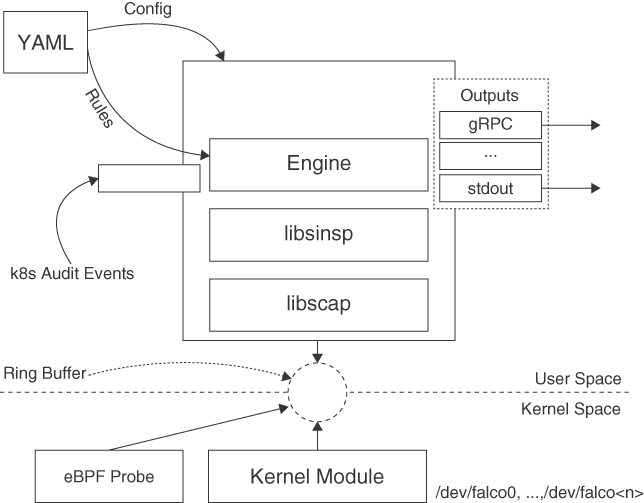
opc:/etc/apparmor.d > ls -l /usr/sbin/\*complain

-rwxr-xr-x 1 root root 1442 Jan 25 07:37 /usr/sbin/aa-complain\*

lrwxrwxrwx 1 root root 11 Feb 11 13:02 /usr/sbin/complain -> aa-complain\*

**Falco**

Falco is a container runtime security tool that deploys to your cluster and starts gathering data for analysis in real time. Falco runs a set of rules against this constant stream of data and raises alerts when the rules detect an event. Falco is deployed to a Kubernetes cluster as a DaemonSet, ensuring that it runs on all nodes. From here, it can start instrumenting system calls, Kubernetes audit logs, and other data sources. Falco can correlate the data from these multiple sources to create a more complete picture of the events in your environment. The capability to instrument system calls is what makes Falco interesting; the Linux system calls power basically everything from opening a file to sending a packet of data to a network device. This means that malicious attackers have to use system calls even if they manage to circumvent other security precautions. Instrumenting system calls can be challenging because they are so pervasive in the normal functioning of the OS; adding instrumentation can negatively affect the system performance. To instrument system calls efficiently, Falco offers a kernel module that is very performant or an eBPF probe for systems that can use eBPF. [Figure 6-26](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ch06fig26) illustrates the high-level architecture for Falco.



**Figure 6-26** Architecture Diagram for Falco

**Tracee**

Tracee is a tool that allows for the real-time monitoring of system calls and kernel events. Although all actions are traced, you can grep through the output to narrow it to a particular pod. The information shown has a precise time stamp, uts\_name, UID, PID, return code, event, and arguments. For Tracee to run, you need to provide at least three volume locations with the -v option for the kernel information to be pulled (such as /lib/modules/ and /usr/src), as well as ephemeral information such as /tmp/tracee. Many options are available for working with the traced information, such as capturing data that a container writes to disk or memory, for further investigation, as well as extracting binaries from a container. All of these features allow Tracee to provide in-depth tracing of an entire container or pod. Although Clair uses alpine-secdb (which covers back-ported fixes), it is not complete—it might have half as many issues as are currently discovered.

**Trivy**

Trivy is a simple and comprehensive vulnerability scanner for containers. Each time Trivy is run, it retrieves a more complete list of vulnerabilities (vuln-list) to analyze. Because this list is downloaded from Alpine Linux, it is most complete when analyzing Alpine and RHEL/CentOS. In a large environment, you might want to set up your own server of the vuln-list, and then use Trivy in client mode and pass the address and port of the server. This does not download the database; instead, it references the common one on the server. This approach also helps in an air-gapped environment, where you can download the list on an external system, check the contents, and manually install the list on a protected server. Another reason Trivy might be a more accurate scanner in finding issues is that it checks the middle layers of an image to find the version of the static linking library. Several continuous integration (CI) tools have easy-to-use integration files for using Trivy in a CI/CD pipeline.

**National Institute of Standards and Technology (NIST) Kubernetes Benchmarks**

The National Institute of Standards and Technology (NIST) has developed Kubernetes Benchmarks, a comprehensive set of guidelines and best practices to ensure the secure deployment and operation of Kubernetes clusters. This section explores the key components of the NIST Kubernetes Benchmarks and discusses how organizations can leverage them to enhance their Kubernetes security posture.

Staying informed about security threats requires reading numerous publications. To start, check out the Federal Information Processing Standard (FIPS) and visit the Computer Security Resource Center Publications web page for standards, special publications, research reports, and more.[**13**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_13a)

The vast amount of information can be overwhelming, so consider using the Cybersecurity Framework (CSF) to organize security into five activities: identify, protect, detect, respond, and recover.

By breaking down cybersecurity into manageable activities and categories, you can focus on the most important information for your current role and expand your knowledge over time.

**NIST Kubernetes Benchmarks**

The Kubernetes Benchmarks provided by NIST are a valuable resource for organizations looking to secure their Kubernetes deployments. They offer a systematic approach to assess and validate the security configurations and controls within a Kubernetes environment. The benchmarks cover various aspects of Kubernetes security, including authentication, authorization, network policies, and logging. By adhering to these benchmarks, organizations can reduce the risk of security incidents, enhance their ability to detect and respond to threats, and maintain compliance with relevant industry standards and regulations.

* **Configuration and hardening guidelines:** One of the fundamental aspects of Kubernetes security involves ensuring the proper configuration and hardening of the cluster components. The NIST Kubernetes Benchmarks provide detailed recommendations for securing the Kubernetes control plane, worker nodes, and associated resources. These guidelines include recommendations for securing the Kubernetes API server, restricting privileged access, enabling secure communication channels, configuring pod security policies, and implementing network segmentation.
* **Authentication and authorization controls:** Authentication and authorization mechanisms ensure that only authorized entities can access and operate within a Kubernetes cluster. The Kubernetes Benchmarks outline best practices for implementing strong authentication mechanisms, such as mutual TLS authentication, integration with external identity providers, and the enforcement of secure password policies. Additionally, the benchmarks provide guidance on implementing fine-grained authorization controls using Kubernetes role-based access control (RBAC) and other relevant mechanisms.
* **Network policies and segmentation:** Effective network policies and segmentation are essential for isolating workloads, preventing lateral movement within a cluster, and protecting sensitive data. The Kubernetes Benchmarks provide recommendations for defining network policies that restrict traffic flows between pods and namespaces, implementing network segmentation through the use of network plug-ins, and enabling encrypted communication between components using Transport Layer Security (TLS).
* **Logging and monitoring:** Comprehensive logging and monitoring are crucial for detecting and investigating security incidents within a Kubernetes cluster. The benchmarks highlight the importance of configuring centralized logging for Kubernetes components, capturing relevant log events, and retaining logs for an appropriate duration. Additionally, the benchmarks emphasize the implementation of robust monitoring solutions that can provide real-time visibility into the cluster’s security posture and facilitate timely incident response.
* **Implementing Kubernetes Benchmarks:** Organizations should adopt a systematic approach to effectively implement the NIST Kubernetes Benchmarks. This involves performing a security assessment of the Kubernetes cluster against the benchmark recommendations, identifying areas of noncompliance or potential vulnerabilities, and remediating them accordingly. Organizations can leverage various tools and technologies to automate the assessment process and continuously monitor their Kubernetes environment for compliance deviations or security incidents.

**National Checklist Program Repository**

The National Checklist Program (NCP), defined by the NIST SP 800-70 Rev. 4, is the U.S. government repository of publicly available security checklists (or benchmarks) that provide detailed low-level guidance on setting the security configuration of operating systems and applications.[**14**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_14a) NCP provides guidance on secure configuration and vulnerability assessments for various operating systems and applications, including Kubernetes and operating systems used by the containers.

**National Vulnerability Database**

The National Vulnerability Database (NVD) is a resource offered by the National Institute of Standards and Technology (NIST), a physical sciences laboratory run by the U.S. government. NIST also manages the Computer Security Resource Center (CSRC), which provides access to documents such as Federal Information Processing Standards (FIPS) and Special Publications (SP). The NVD contains a wealth of information, including checklists for compliance, known issues, and specialized vulnerabilities.[**15**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_15a)

**NIST SP 800-190 Application Container Security Guide**

The NIST SP 800-190 Application Container Security Guide provides an overview of containers and their security challenges. It also provides recommendations for securing container-based applications throughout the entire application lifecycle.[**16**](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_16a)

**Summary**

Security is such a pervasive and multifaceted topic that entire books have been dedicated to it. This chapter merely scratched the surface of security practices in a cloud native environment. To start, you looked at the 4Cs of cloud native security and examined the various security controls, features, and best practices for securing your workloads and infrastructure. You explored several security controls services and features offered by Oracle Cloud Infrastructure. You worked your way through the various layers, including cluster-level security controls, containers, and container supply chain security, and you looked at best practices and tools to ensure secure coding. You also examined several third-party and open-source tools that are popular in the cloud native community for implementing security best practices and ensuring a good security posture.

**References**

[1](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_1) Secrets Store CSI Driver Provider for OCI Vault: <https://github.com/oracle/oci-secrets-store-csi-driver-provider/blob/main/GettingStarted.md>

[2](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_2) Master Encryption Keys in Other Tenancies: <https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengencryptingdata.htm#contengencryptingdata_topic_Encryption_keys_in_other_tenancies>

[3](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_3) Audit events: <https://cloud.oracle.com/audit/events>

[4](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_4) Audit logs: <https://cloud.oracle.com/logging/audit>

[5](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_5) LoadBalancer: <https://cloud.oracle.com/load-balancer/load-balancers>

[6](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_6) WAF edge policies: <https://cloud.oracle.com/waf/policies>

[7](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_7) Snyk.io (<https://snyk.io/>): <https://github.com/oracle-quickstart/oke-snyk>

[8](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_8) Sysdig.com (<https://sysdig.com/>)8: <https://github.com/oracle-quickstart/oke-sysdig>

[9](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_9) Kubernetes CIS Benchmark: <https://www.cisecurity.org/benchmark/kubernetes>

[10](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_10) securityContext Options: <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#security-context-1>

[11](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_11) Supported admission controllers: <https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengadmissioncontrollers.htm>

[12](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_12) OWASP Top Ten reference: <https://owasp.org/www-project-kubernetes-top-ten/>

[13](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_13) NIST Computer Security Resource Center: <https://csrc.nist.gov/publications>

[14](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_14) National Checklist Program (NCP): <https://nvd.nist.gov/ncp/repository>

[15](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_15) National Vulnerability Database (NVD): <https://nvd.nist.gov/>

[16](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9780137902835/ch06.xhtml#ref6_16) NIST SP 800-190 Application Container Security Guide: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>