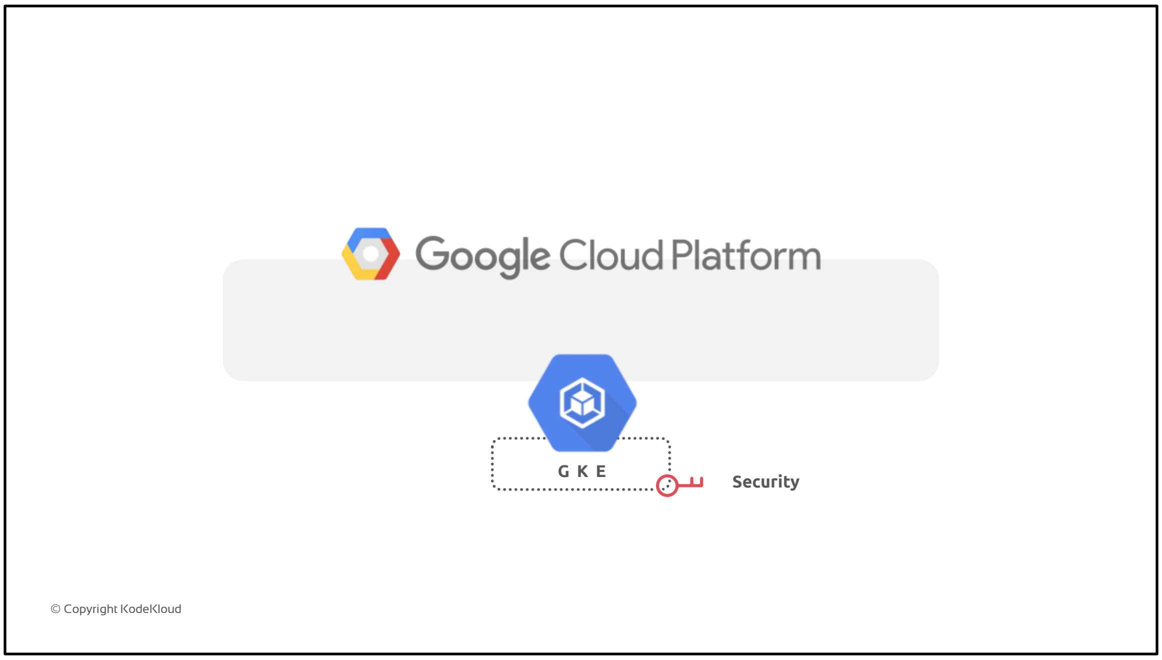
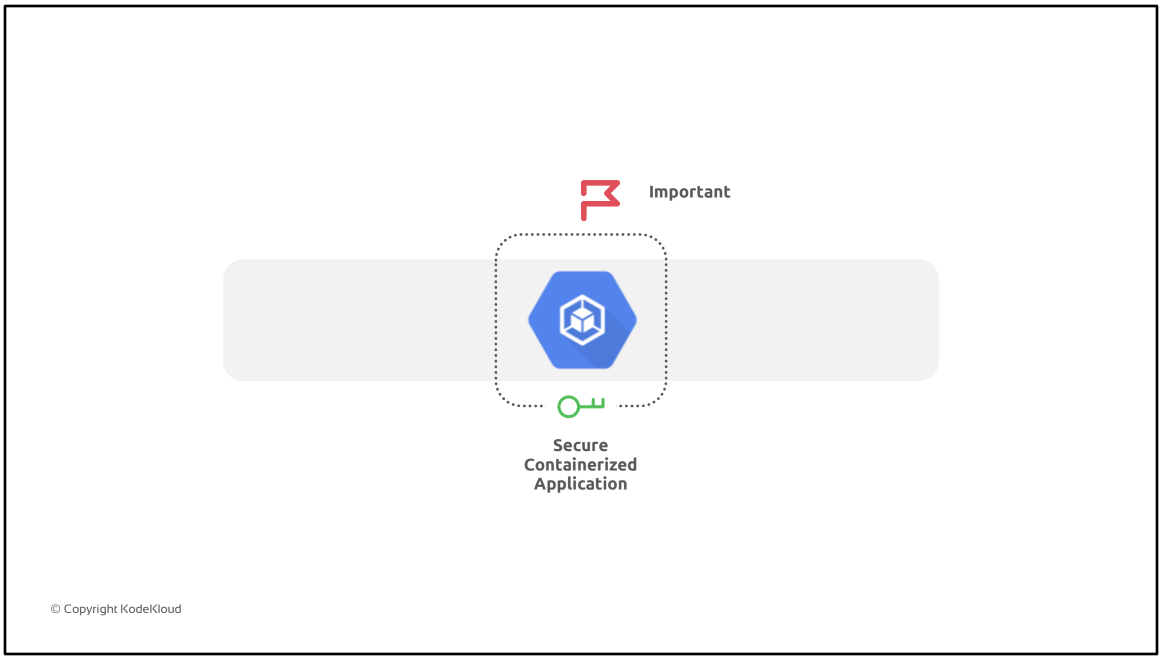
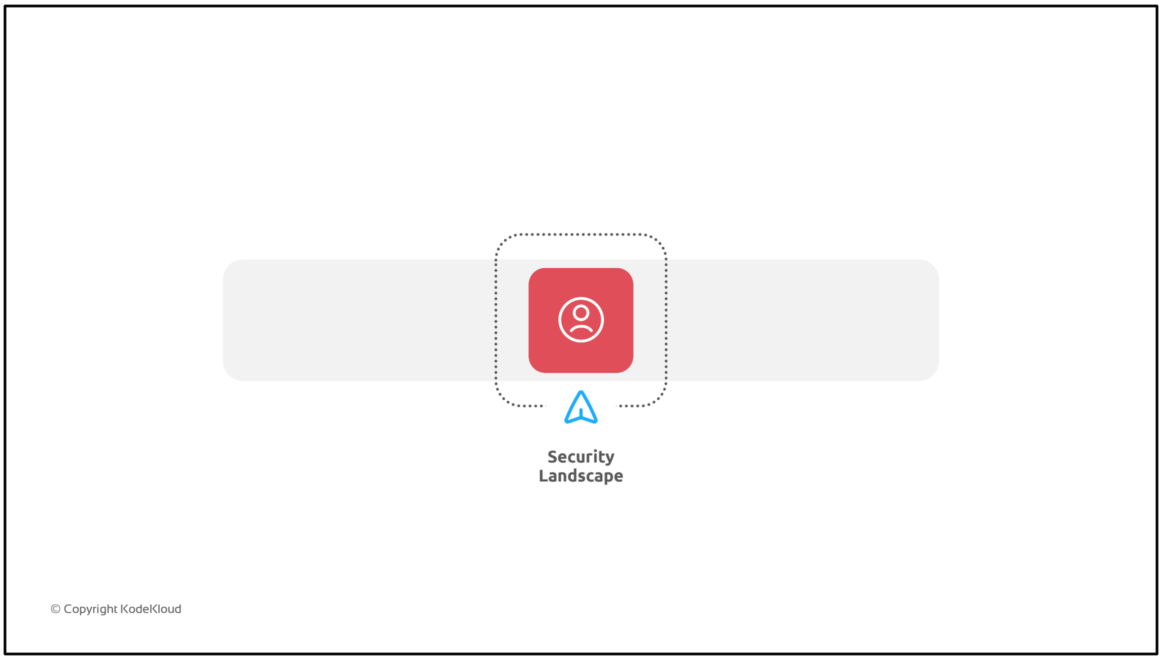
Section Introduction – Managing Security Aspects



Hello fellow Google Cloud Enthusiasts, welcome to the Google Kubernetes Engine security module.

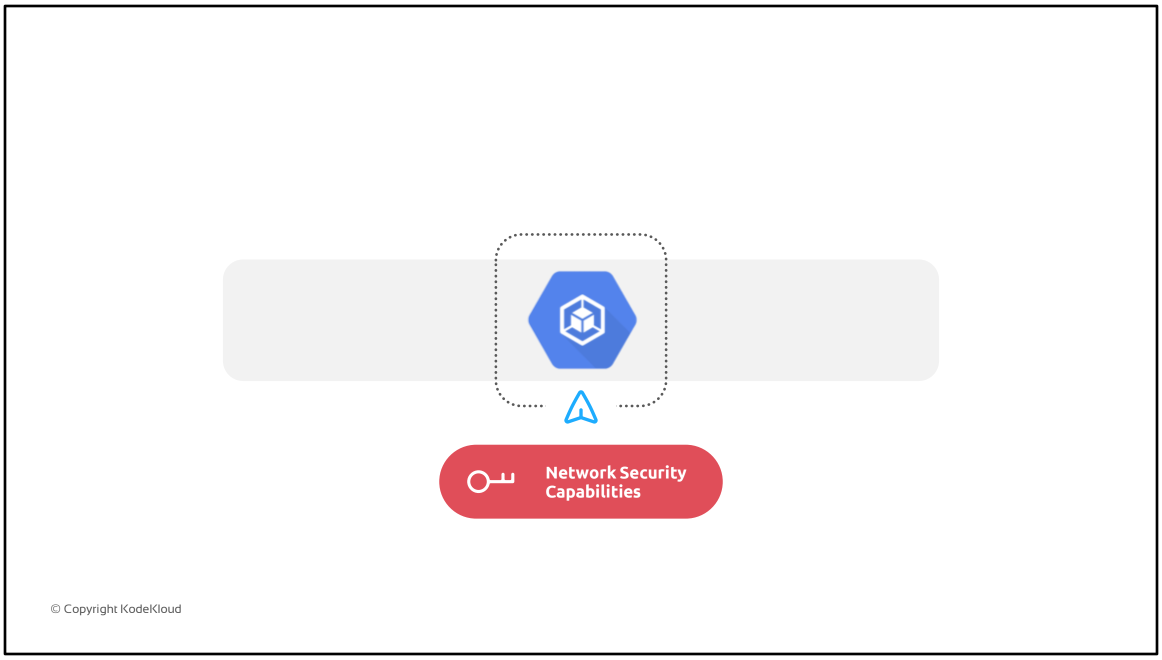


In today's digital landscape, ensuring the security of your containerized applications is paramount. As organisations embrace the power of Google Kubernetes Engine (GKE) for orchestrating their workloads, it becomes crucial to navigate the intricacies of securing your GKE clusters effectively. In this section, we'll delve into the comprehensive domain of GKE security.

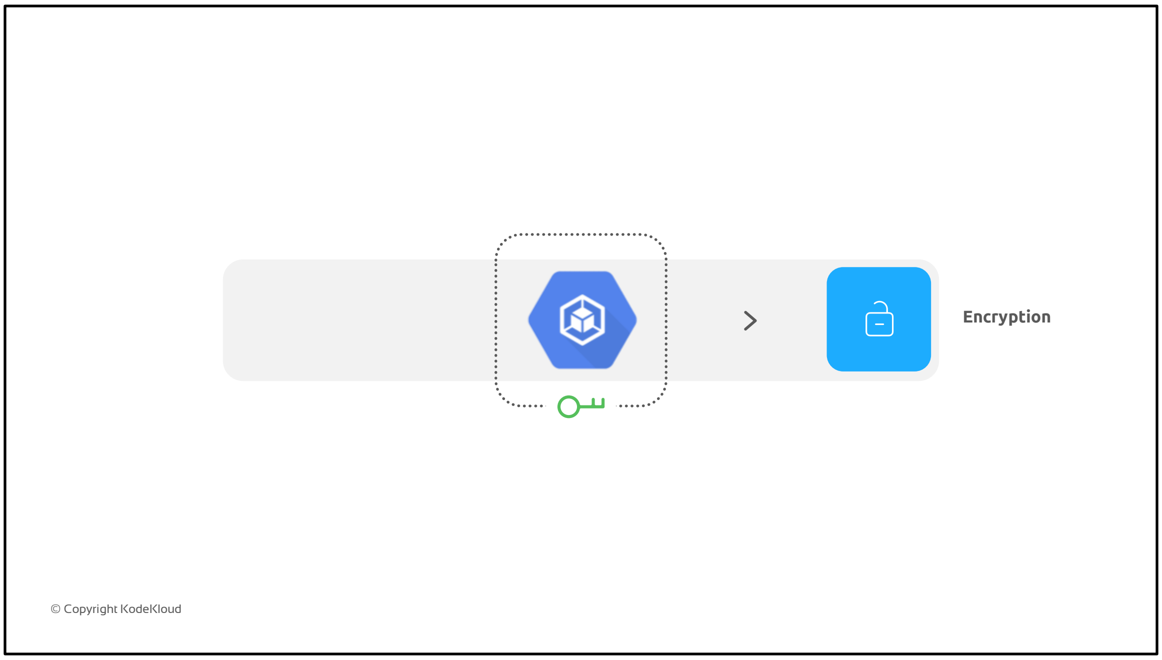


We’ll start by discussing the **shared responsibility** model that delineates Google's role from the customer's, and exploring an array of **robust security mechanisms** at our disposal. You'll gain insights into the authentication and authorization mechanisms that form the bedrock of GKE's security architecture, encompassing GKE ***Role-Based Access Control (RBAC), Google Cloud Identity and Access Management (IAM),*** and the synergy between Kubernetes **Service accounts** and GKE service accounts.

We’ll then move onto the **security landscape** of the control plane, nodes, and credentials, equipping you with the knowledge to safeguard these critical facets.

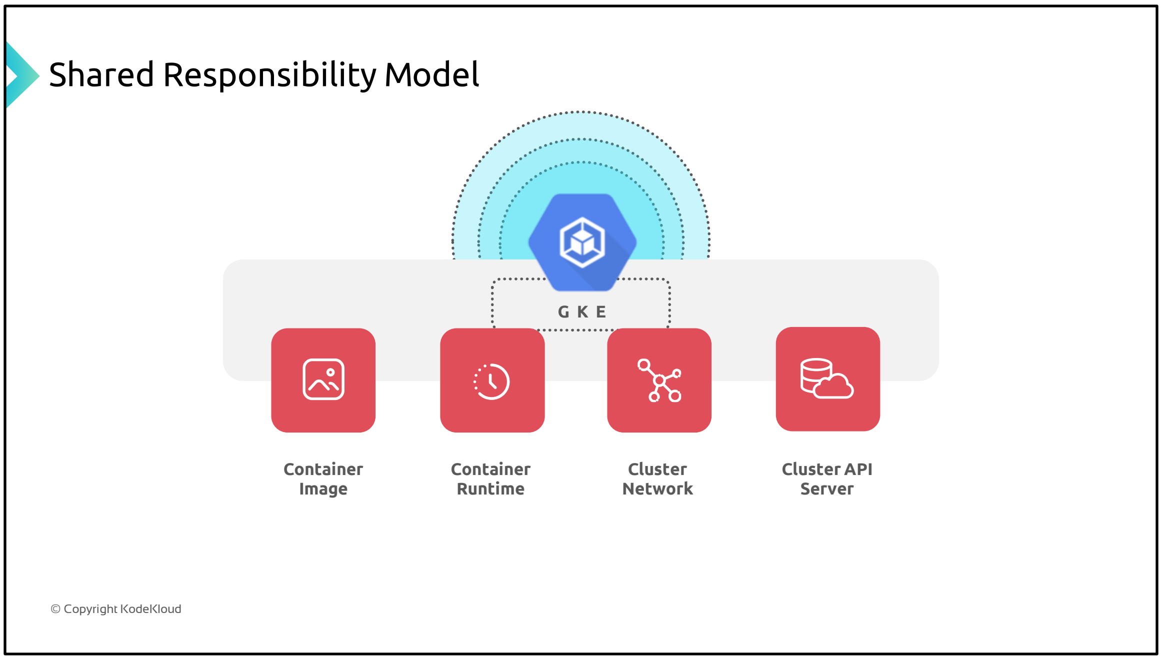


We'll delve into GKE's network security capabilities, spanning Virtual Private Cloud (VPC) **integration** and network **policies**, while also emphasising the significance of audit logging for maintaining visibility and compliance.



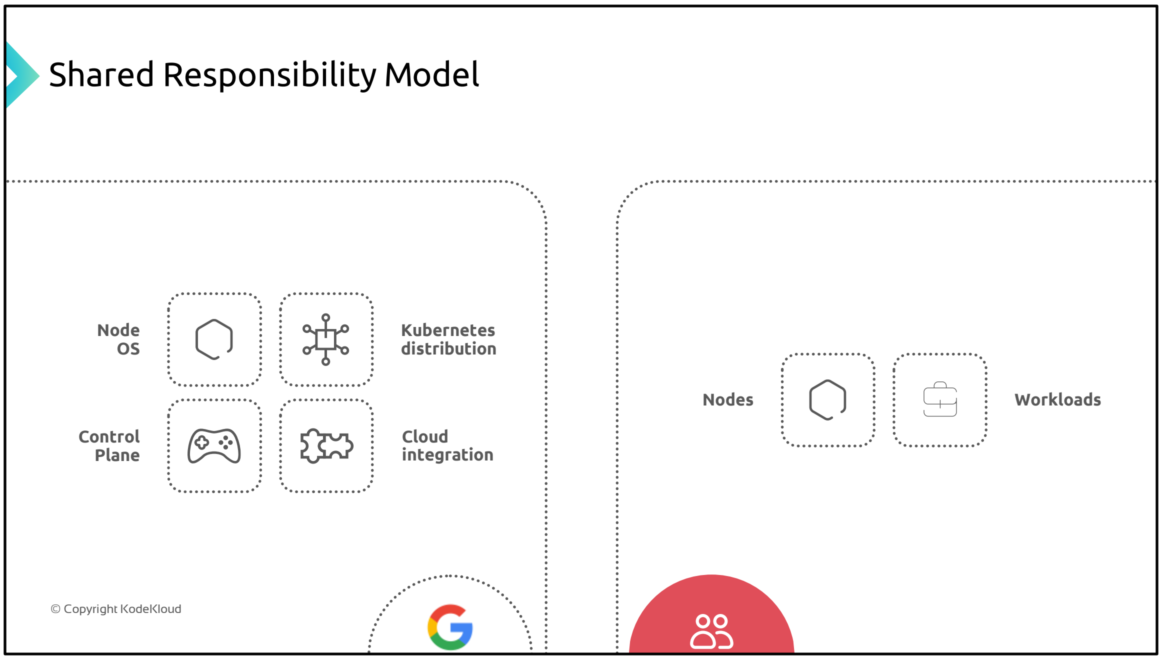
The protection of sensitive data is a paramount concern, and GKE provides a suite of tools to tackle this challenge. To finish this section we'll dive into the concepts around controlling the encryption of sensitive data on GKE, leveraging Google Cloud's encryption solutions for data at rest, in transit and in use.

GKE Shared Responsibility Model

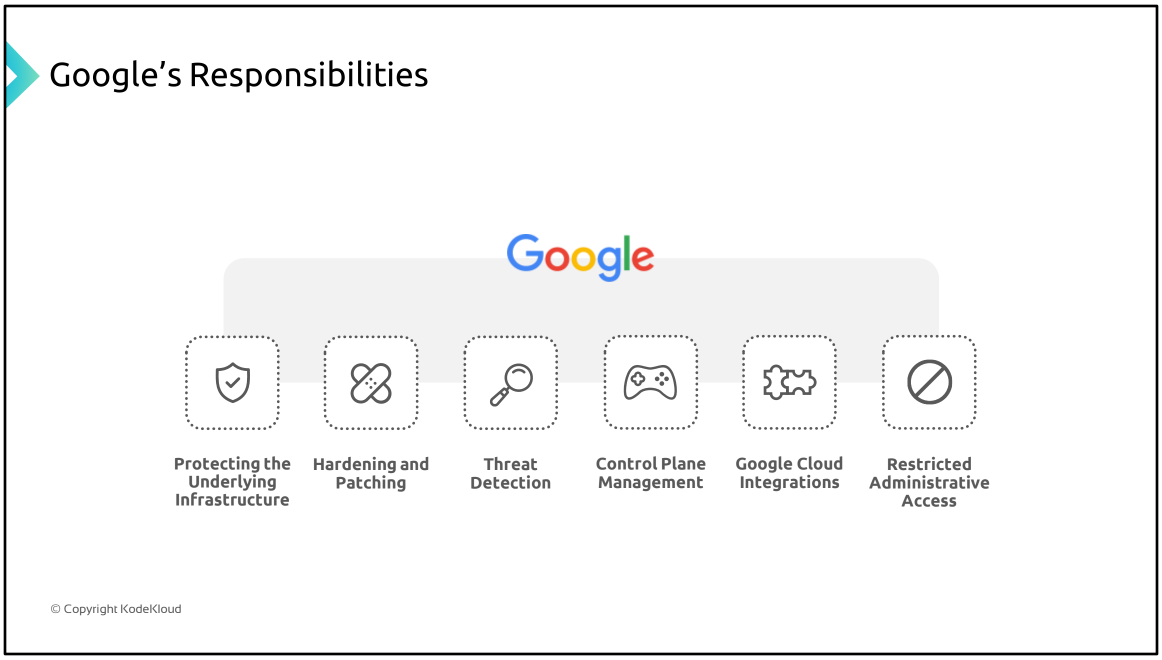


Shared Responsibility Model

Securing workloads in GKE requires comprehensive protection across multiple layers of the stack, encompassing the container image, container runtime, cluster network, and access to the cluster API server. To ensure effective security, it is recommended to adopt a layered approach that considers each of these components. Implementing the principle of least privilege is crucial when granting access to users and applications in your GKE clusters.



Similar to most of the other cloud services, Google adopts a shared responsibility model when it comes down to managing security aspects of GKE clusters and the application workloads running on it. The shared responsibility model for running applications on Google Kubernetes Engine (GKE) involves distinct responsibilities for both Google and the google Kubernetes engine users. While Google is responsible for securing the underlying infrastructure components for a GKE cluster, customers too need to make sure that the customisable aspects of application workloads are secured properly.



Google's Responsibilities

Google
Protecting the Underlying Infrastructure
Hardening and Patching
Threat Detection
Control Plane Management
Google Cloud Integrations
Restricted Administrative Access

Google's Responsibilities:

1. Protecting the Underlying Infrastructure: Google is responsible for safeguarding the underlying infrastructure, including hardware, firmware, kernel, operating system, storage, and network. This involves encryption of data at rest, data encryption in transit, custom-designed hardware, physical security measures, protection against modification using Shielded Nodes, and adherence to secure software development practices.

2. Hardening and Patching: Google ensures the hardening and patching of the nodes' operating system (e.g., Container-Optimized OS or Ubuntu), Kubernetes node components, and the GKE control plane. This includes making patches and updates available promptly, automatic deployment of updates if auto-upgrade is enabled, and implementing security measures following CIS benchmarks.

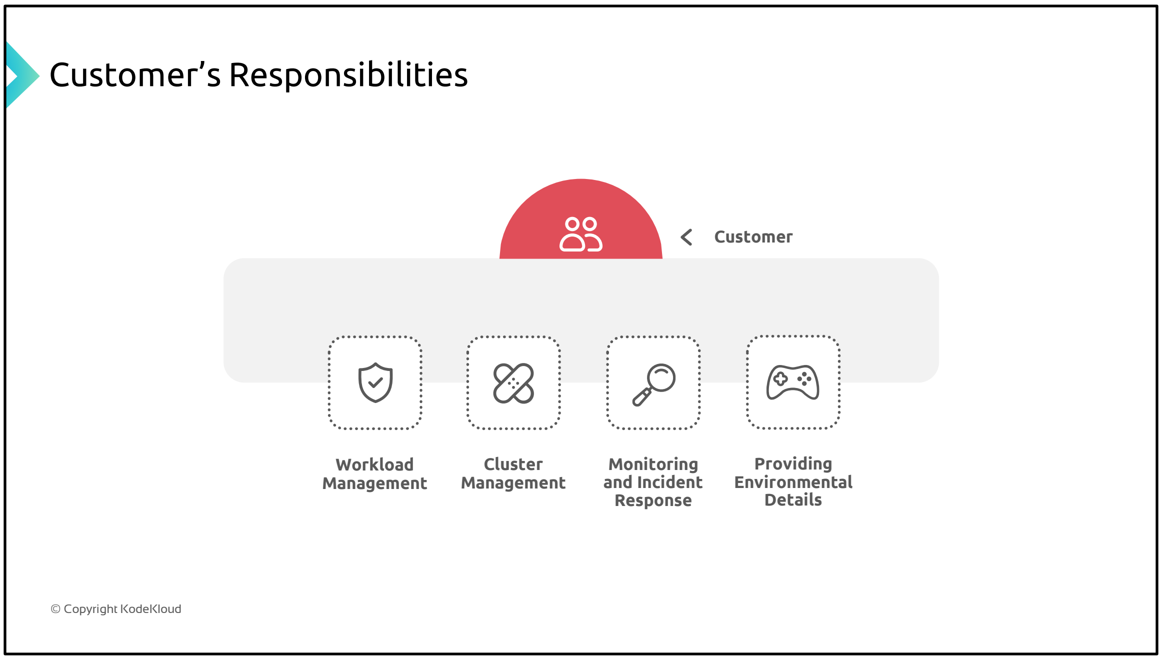
3. **Threat Detection:** Google incorporates container-specific threat detection into the kernel with Container Threat Detection (offered separately with Security Command Center). This helps identify and mitigate container-related security risks and threats.

4. **Control Plane Management:** Google hardens and patches the control plane components, which include the control plane VM, API server, scheduler, controller manager, cluster CA, TLS certificate management, secrets encryption, audit logging, etcd, and other controllers. The control plane components run on Google-operated Compute Engine instances and are single tenant, ensuring isolation for each customer.

5. **Google Cloud Integrations:** Google provides integrations with various Google Cloud services, including IAM, Cloud Audit Logs, operations suite, Key Management Service, Security Command Center, and more. These integrations enhance the

security, management, and compliance capabilities of GKE.

6. **Restricted Administrative Access:** Google also restricts and logs administrative access to customer clusters for support purposes, ensuring transparency and accountability with Access Transparency.



Customer's Responsibilities

Customer
Workload Management
Cluster Management
Monitoring and Incident Response
Providing Environmental Details

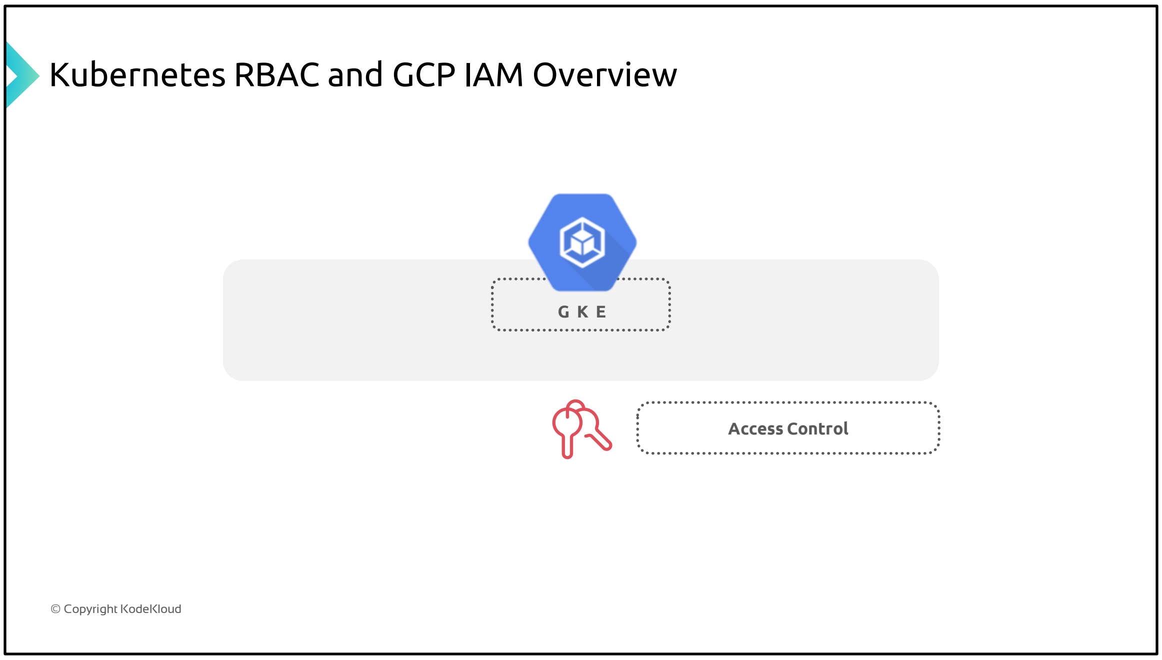
1. **Workload Management:** Customers are responsible for maintaining their workloads, including application code, build files, container images, data, RBAC or IAM policies, and the containers and pods they are running. This includes ensuring the security, availability, and performance of their applications.
2. **Cluster Management:** Customers need to enrol their GKE clusters in auto-upgrade (default) or manually upgrade them to supported versions. Keeping the clusters up to date helps benefit from the latest features, bug fixes, and security

enhancements provided by Google.

3. **Monitoring and Incident Response:** Customers are responsible for monitoring their GKE clusters and applications. They should proactively respond to alerts and incidents, leveraging tools such as the security posture dashboard and Google Cloud's operations suite. This helps maintain the health, performance, and security of their applications.

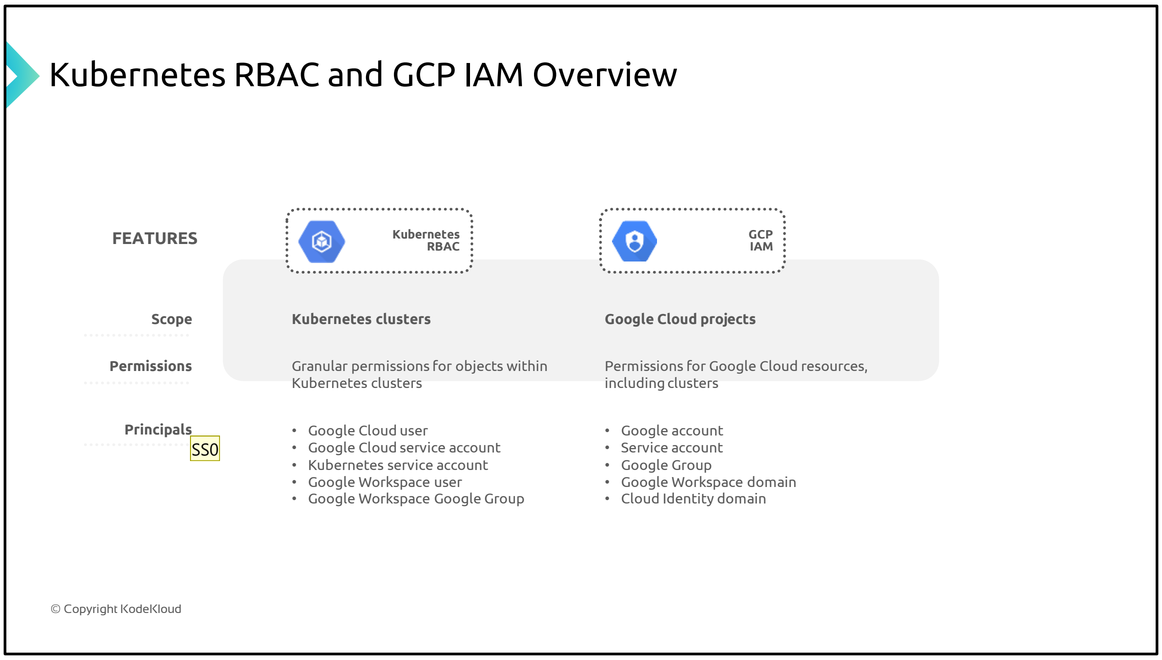
4. **Providing Environmental Details:** When requested by Google for troubleshooting purposes, customers need to provide relevant environmental details related to their GKE clusters. This information assists Google in diagnosing and resolving any issues effectively.

Authentication and Authorization in GKE



Kubernetes RBAC and GCP IAM Overview

Access control in Google Kubernetes Engine (GKE) is an important aspect of managing and securing your GKE resources. In this lesson, we will explore how access control works in GKE and the mechanisms available for managing access.



Kubernetes RBAC and GCP IAM Overview

FEATURES	Kubernetes RBAC	GCP IAM
Scope	Kubernetes clusters	Google Cloud projects
Permissions	Granular permissions for objects within Kubernetes clusters	Permissions for Google Cloud resources, including clusters
Principals SS0	• Google Cloud user • Google Cloud service account • Kubernetes service account • Google Workspace user • Google Workspace Google Group	• Google account • Service account • Google Group • Google Workspace domain • Cloud Identity domain

There are two main mechanisms for managing access control in GKE: GCP IAM and Kubernetes RBAC

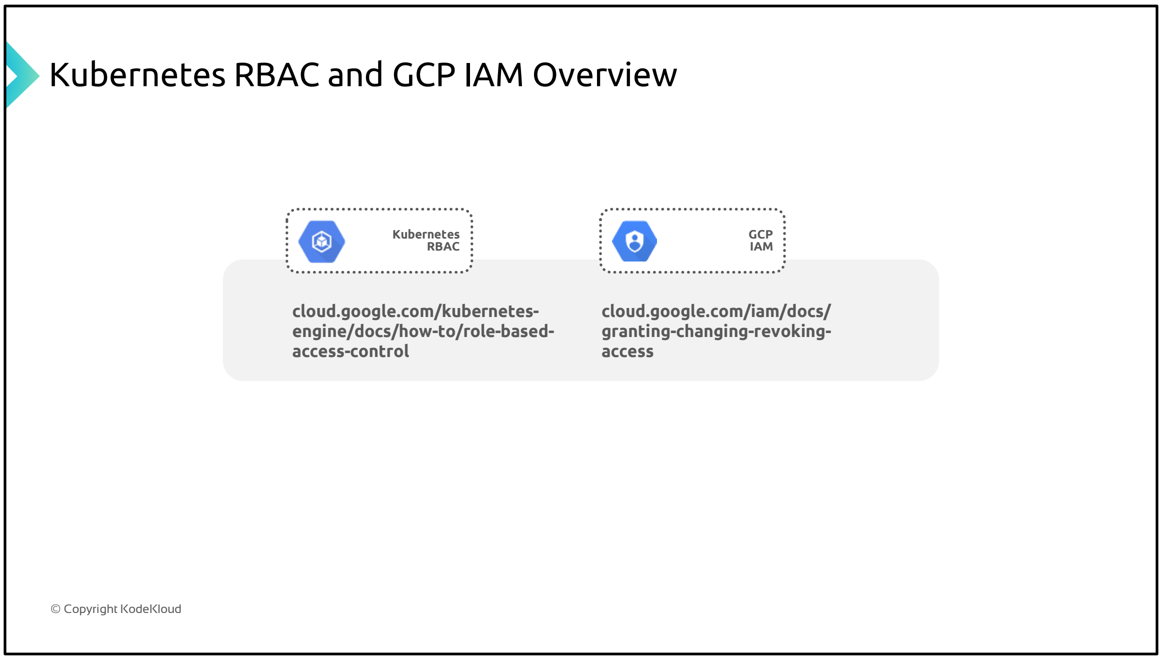
Which type of access control to use depends on the specific needs. If fine-grained permissions for every object and operation is needed within a cluster, Kubernetes RBAC is the best choice. However If multiple Google Cloud components are being used and there is no need to manage granular Kubernetes-specific permissions, IAM is a good choice.

Here is a table that summarizes the differences between Kubernetes RBAC and GCP IAM.

Identity and Access Management (IAM): IAM is a central component of Google Cloud Platform (GCP) that manages access to various resources, including GKE clusters. With IAM, you can assign roles and permissions to users, service accounts, and groups at a project level in GCP.
IAM roles define a set of permissions that determine what actions can be performed on GKE clusters and other GCP resources. For example, you can assign a user the role of "Kubernetes Engine Developer," which grants permissions to create and manage

GKE clusters.
It's important to note that IAM manages access at the project level and applies to all GKE clusters within the project. IAM roles provide broader control and are suitable if you use multiple GCP components and don't require fine-grained Kubernetes-specific permissions.

Kubernetes Role-Based Access Control (RBAC): RBAC, on the other hand, is a built-in feature of Kubernetes, the underlying container orchestration system used by GKE. RBAC grants granular permissions to objects within Kubernetes clusters. Permissions are defined as ClusterRole or Role objects.
ClusterRole defines a set of permissions that can be applied across the entire cluster, while Role is specific to a namespace within the cluster. For example, you can create a ClusterRole that allows read-only access to all resources in the cluster.
RoleBinding objects are used to associate Roles or ClusterRoles with Kubernetes users, service accounts, Google Cloud users, or Google Groups. This allows you to grant specific permissions to individual users or groups within your GKE clusters.



Kubernetes RBAC and GCP IAM Overview

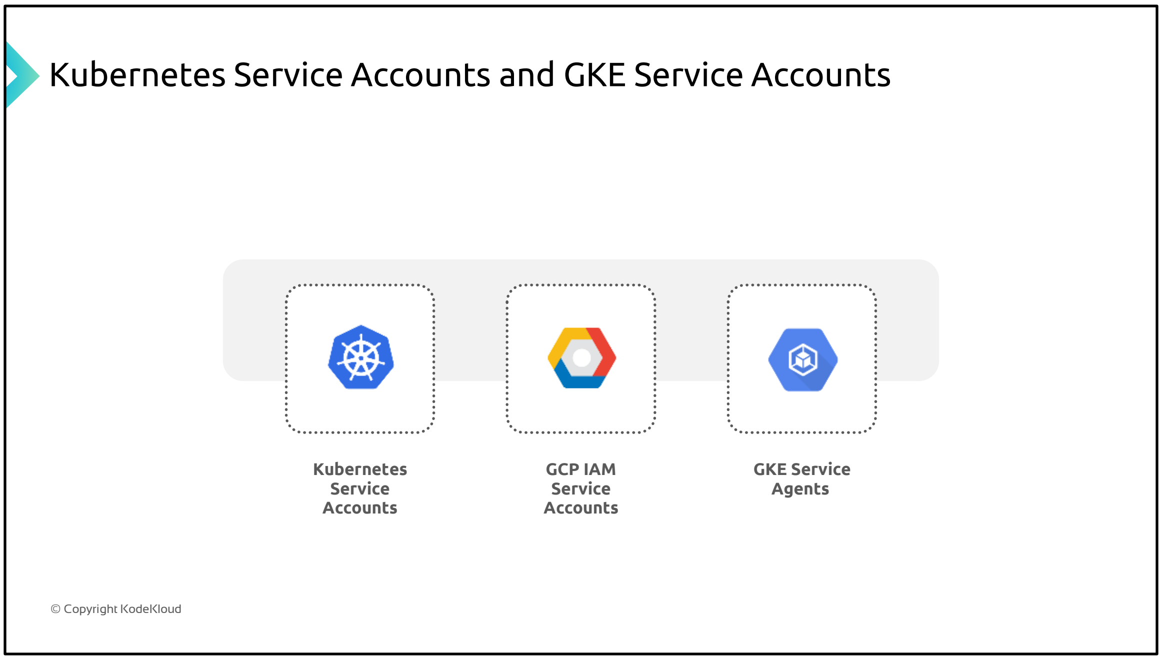
Kubernetes RBAC
GCP IAM

cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control

cloud.google.com/iam/docs/granting-changing-revoking-access

Some additional resources that you may find helpful:

- GKE RBAC documentation: https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control
- GCP IAM documentation: https://cloud.google.com/iam/docs/granting-changing-revoking-access



Kubernetes Service Accounts and GKE Service Accounts

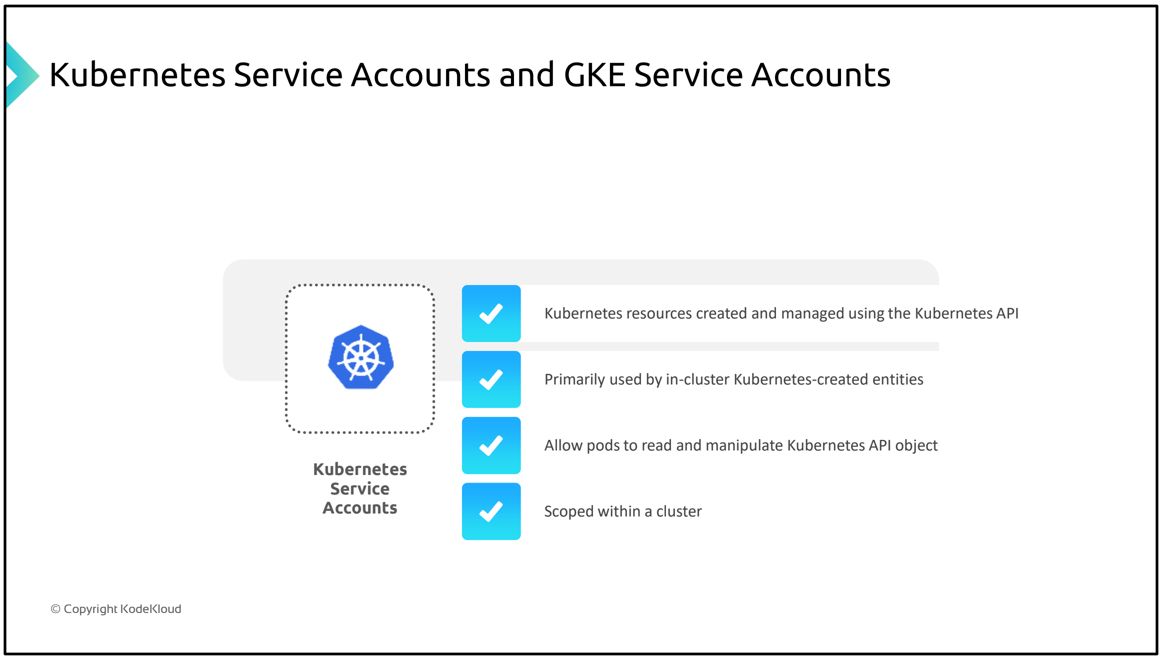
There are three different ways in which a GKE cluster is accessed programmatically.

Kubernetes Service Accounts
GCP IAM Service Accounts
GKE Service Agents

These are different entities with distinct purposes within the context of Google Kubernetes Engine (GKE) and Google Cloud Platform (GCP).

They provide an identity for applications and workloads running in Pods. Pods can use

Kubernetes service accounts to authenticate to the Kubernetes API server.



Kubernetes Service Accounts and GKE Service Accounts

Kubernetes Service Accounts

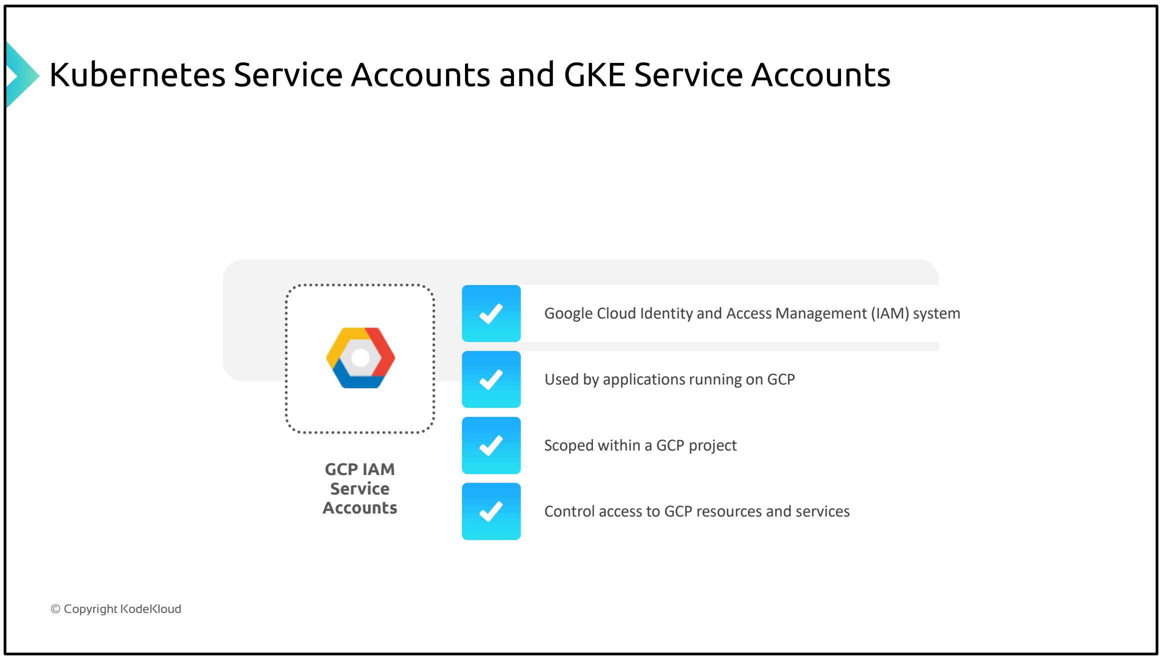
- ✓ Kubernetes resources created and managed using the Kubernetes API
- ✓ Primarily used by in-cluster Kubernetes-created entities
- ✓ Allow pods to read and manipulate Kubernetes API object
- ✓ Scoped within a cluster

Kubernetes Service Accounts:

- **Kubernetes resources created and managed using the Kubernetes API:** Kubernetes service accounts are Kubernetes resources that are created and managed using the Kubernetes API.
- **Primarily used by in-cluster Kubernetes-created entities:** They are primarily used by in-cluster Kubernetes-created entities, such as Pods, to authenticate themselves to the Kubernetes API server or external services.
- **Allow Pods to read and manipulate**

Kubernetes API objects: Kubernetes Service Accounts allow Pods to read and manipulate Kubernetes API objects, enabling operations like deploying applications to the cluster or interacting with other cluster resources.

• **Scoped within a cluster:** Kubernetes Service Accounts are scoped within a cluster and provide an identity for applications and workloads running in Pods.



Kubernetes Service Accounts and GKE Service Accounts

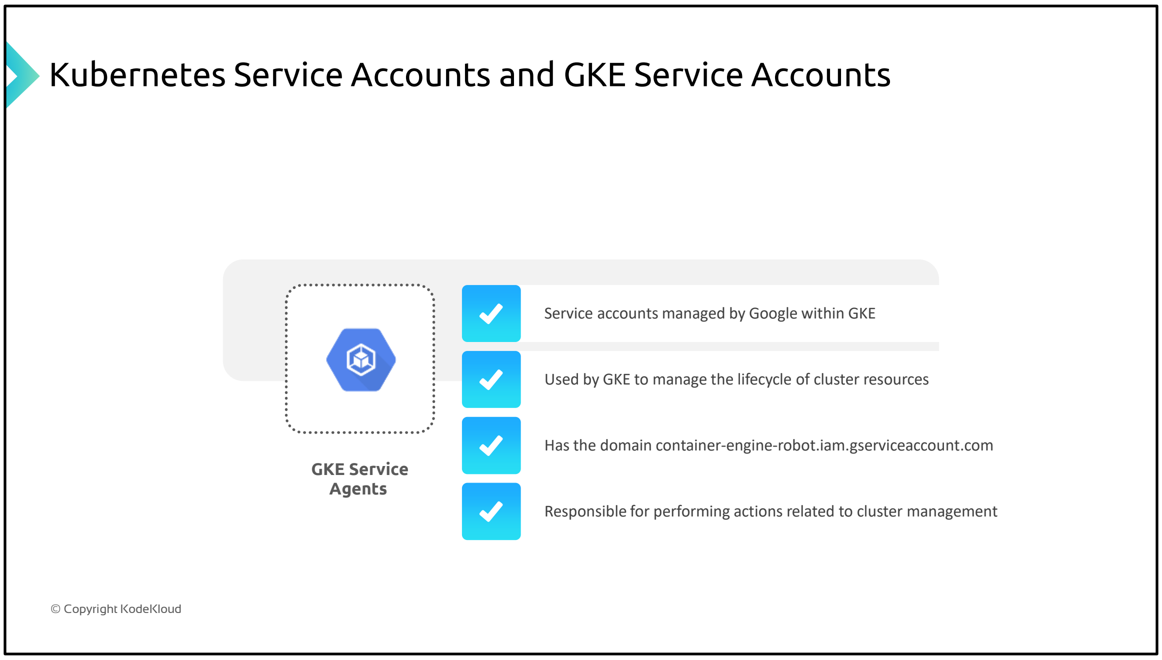
GCP IAM Service Accounts

- ✓ Google Cloud Identity and Access Management (IAM) system
- ✓ Used by applications running on GCP
- ✓ Scoped within a GCP project
- ✓ Control access to GCP resources and services

GCP IAM Service Accounts:

- GCP IAM Service Accounts are special accounts managed by the **Google Cloud Identity and Access Management (IAM) system**.
- They are intended **to be used by applications running on GCP** rather than individuals directly.
- IAM Service Accounts are **scoped within a GCP project** and are used to make authenticated calls to Google Cloud APIs.
- They can be assigned specific roles and

permissions to **control access to GCP resources and services.**



Kubernetes Service Accounts and GKE Service Accounts

GKE Service Agents

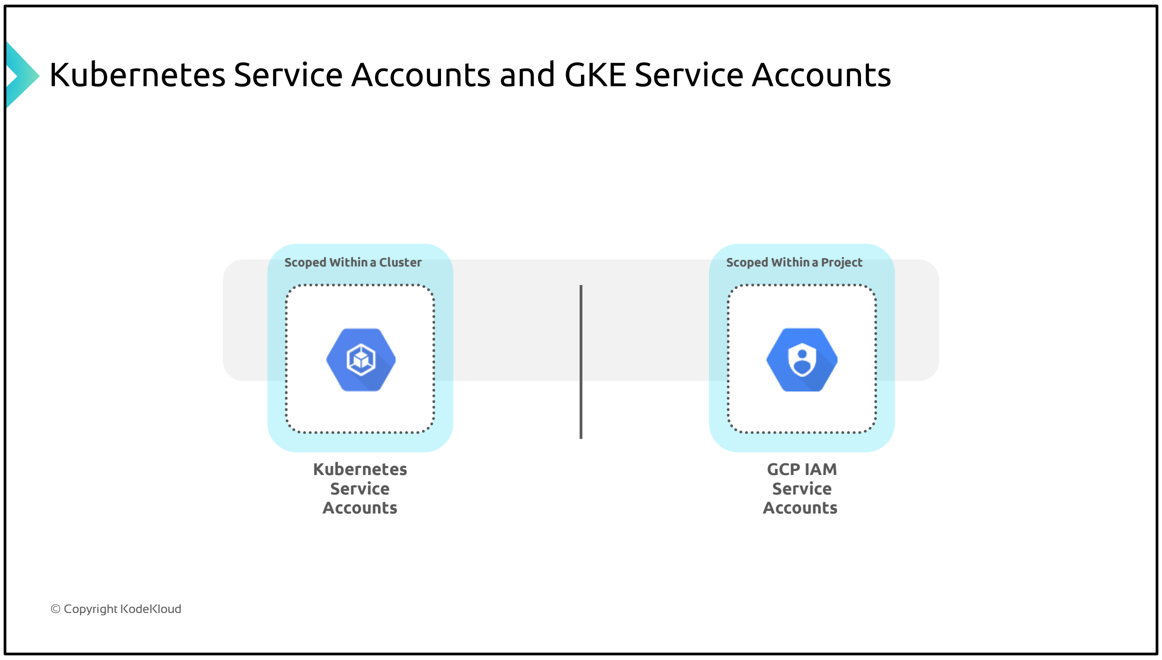
- ✓ Service accounts managed by Google within GKE
- ✓ Used by GKE to manage the lifecycle of cluster resources
- ✓ Has the domain container-engine-robot.iam.gserviceaccount.com
- ✓ Responsible for performing actions related to cluster management

GKE Service Agents:

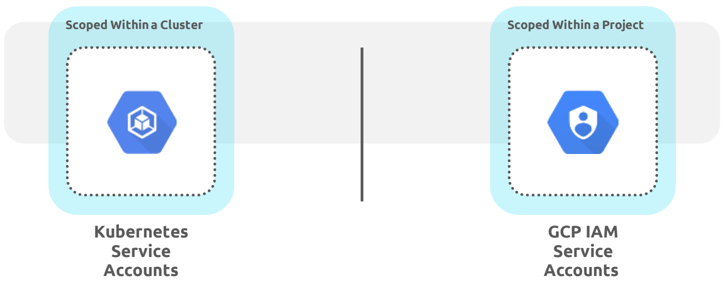
• GKE Service Agents are **service accounts managed by Google within GKE.**
• They are specifically **used by GKE to manage the lifecycle of cluster resources** on behalf of users, such as nodes, disks, and load balancers.
• The **GKE Service Agent has the domain container-engine-robot.iam.gserviceaccount.com** and is granted the Kubernetes Engine Service Agent role on the project as soon as the GKE API is

enabled.

• The GKE Service Agent is **responsible for performing actions related to cluster management** and resource provisioning.

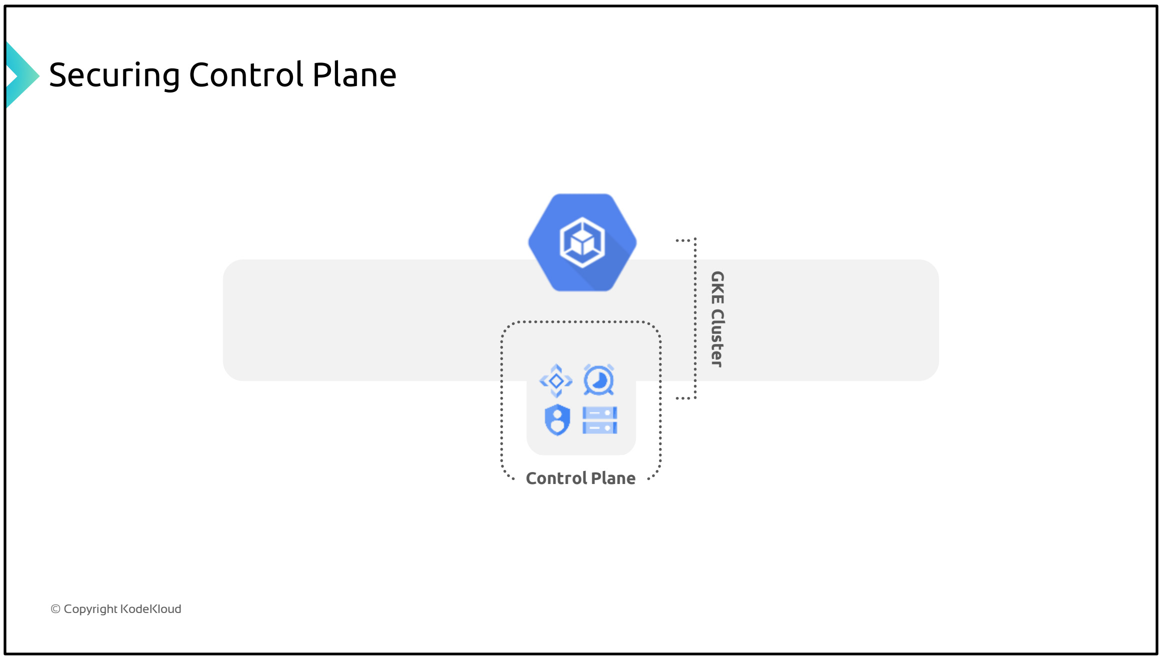


Kubernetes Service Accounts and GKE Service Accounts



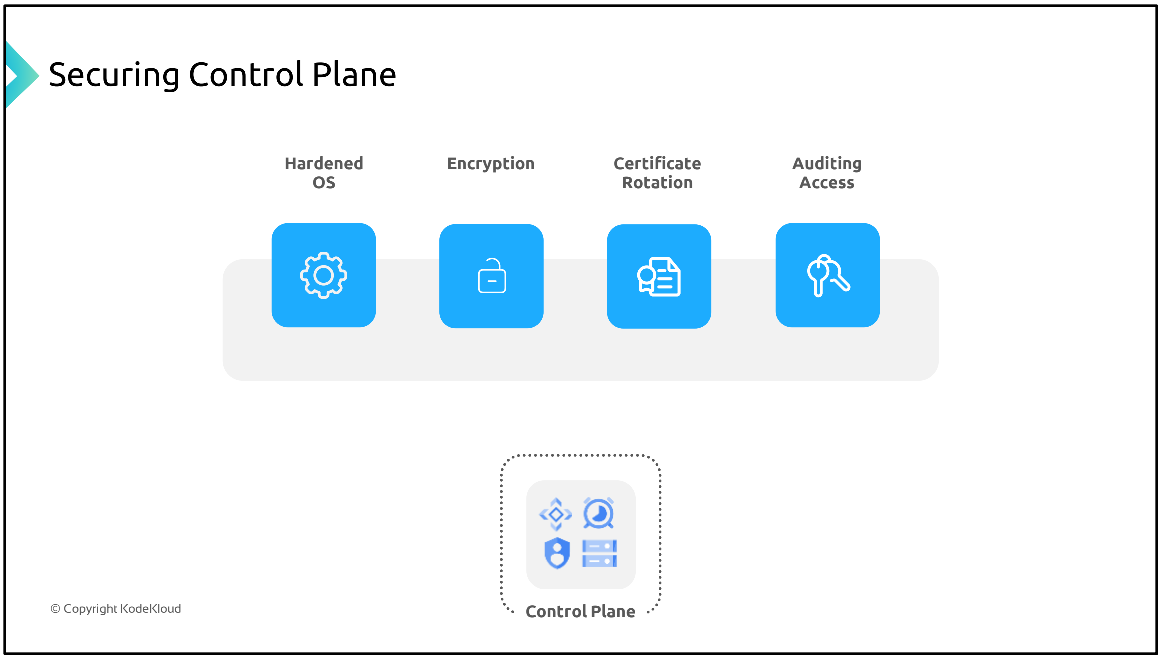
The main difference between Kubernetes service accounts and IAM service accounts is that Kubernetes service accounts are scoped within a cluster, while IAM service accounts are scoped within a project. This means that Kubernetes service accounts can only be used to access resources within the cluster in which they were created, while IAM service accounts can be used to access resources in any project that they have permissions to access.

GKE Security – Overview



Securing Control Plane

As we have seen in previous lectures, the control plane is the core of a GKE cluster and is responsible for managing the cluster's resources, such as nodes, pods, and services.



Securing Control Plane

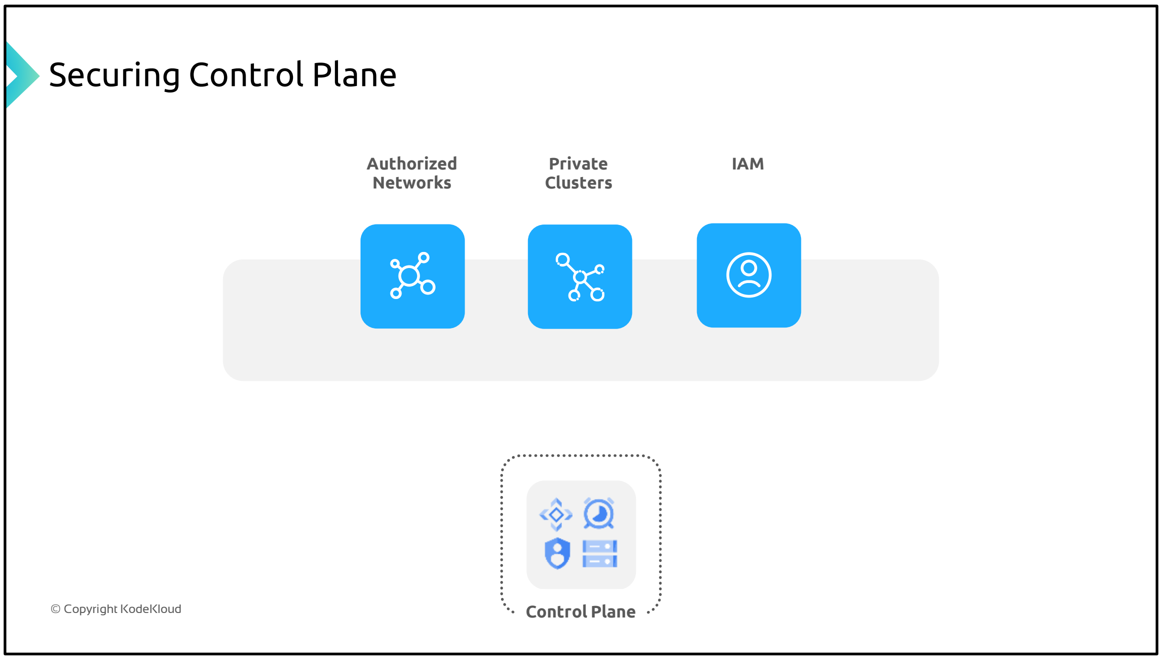
Google takes a number of steps to secure the control plane, including:

•Running the control plane on a hardened operating system: The control plane runs on Container-Optimized OS, which is a security-hardened operating system designed by Google. This operating system includes a number of security features, such as SELinux, AppArmor, and kernel hardening.

•Encrypting the control plane's communication: The control plane's communication is encrypted using TLS. This means that all traffic between the control plane and other components of the cluster is encrypted, making it difficult for attackers to eavesdrop on or tamper with the traffic.

•Rotating the control plane's certificates on a regular basis: The control plane's certificates are rotated on a regular basis. This helps to protect the control plane from attacks that exploit expired or compromised certificates.

•Auditing access to the control plane: Access to the control plane is audited. This means that you can track who has accessed the control plane and what they have done. This information can be used to identify and investigate security incidents.



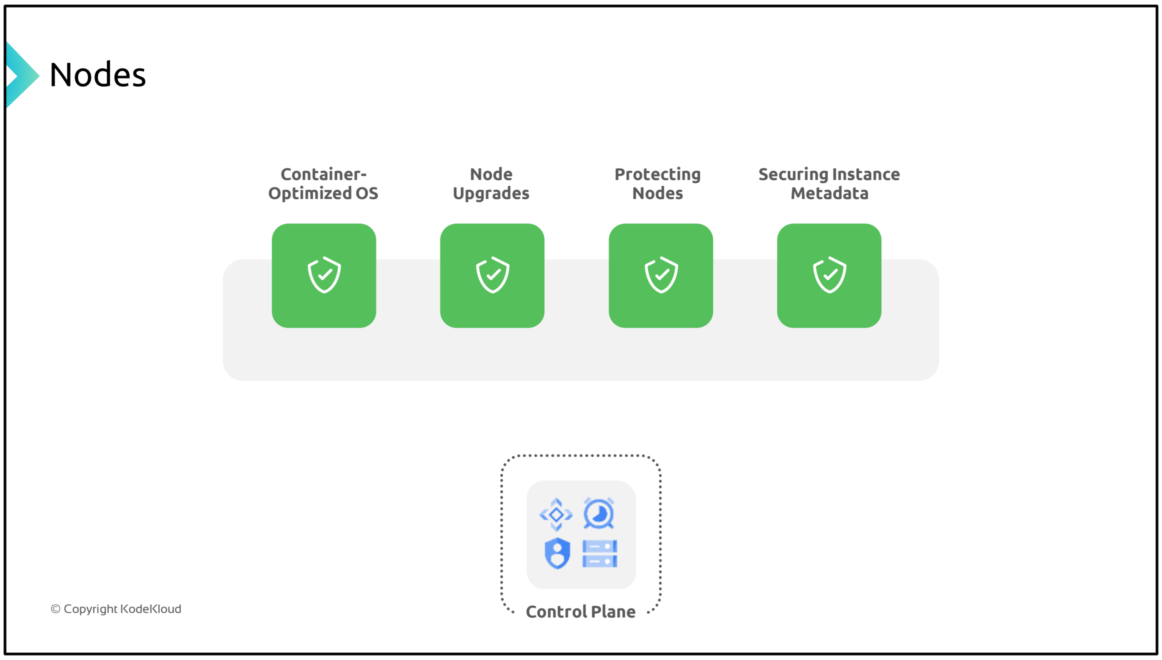
Securing Control Plane

In addition, Google also provides a number of features that help in securing the control plane, such as:

•Authorized networks: You can use authorized networks to restrict access to the control plane to specific IP addresses or ranges.

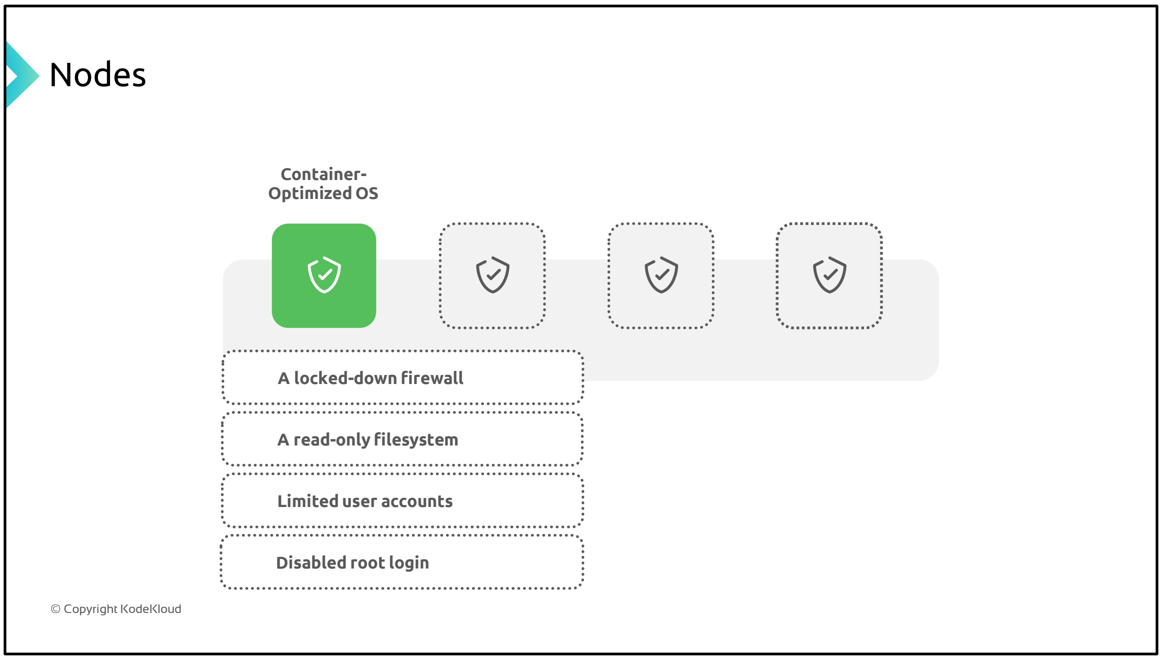
•Private clusters: You can create private clusters that are not accessible from the public internet.

•IAM: You can use IAM to control who has access to the control plane and what permissions they have.



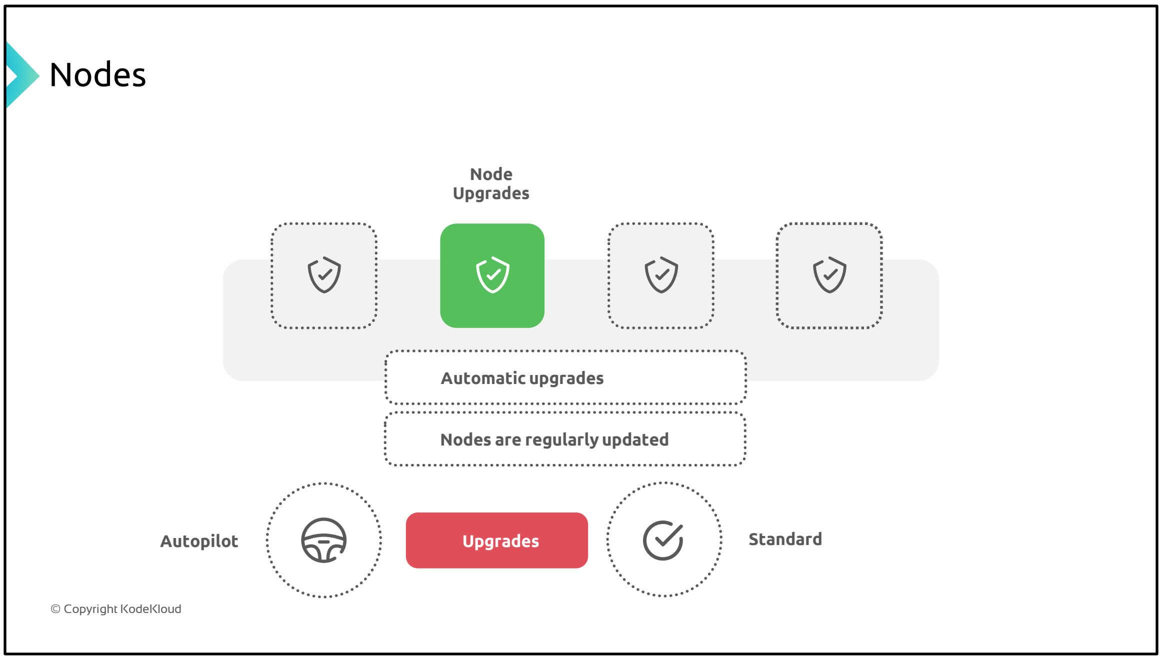
Nodes

Now, let's take a look at the factors that we can consider for securing worker nodes of a GKE cluster.



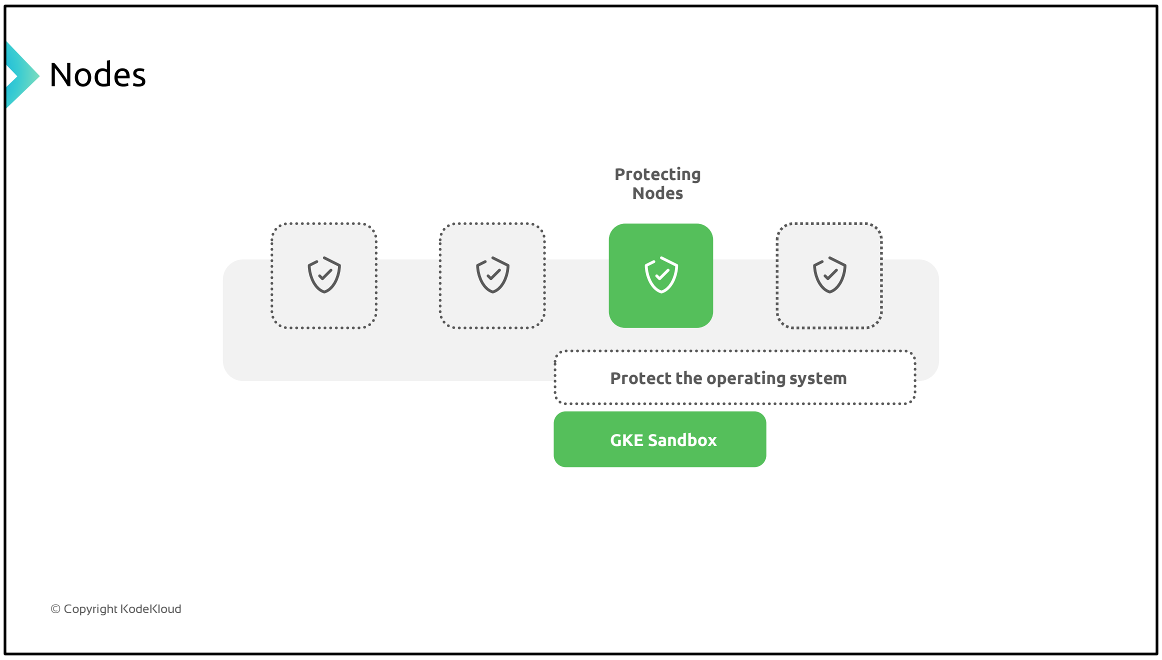
Nodes

Container-Optimized OS (COS):
By default, GKE nodes use Container-Optimized OS (COS) as the operating system. COS is a security-hardened operating system that includes a number of features to help protect your nodes, such as:
•A locked-down firewall
•A read-only filesystem
•Limited user accounts
•Disabled root login
It's worth noting that GKE Autopilot nodes always use Container-Optimized OS as the operating system, ensuring the highest level of security.

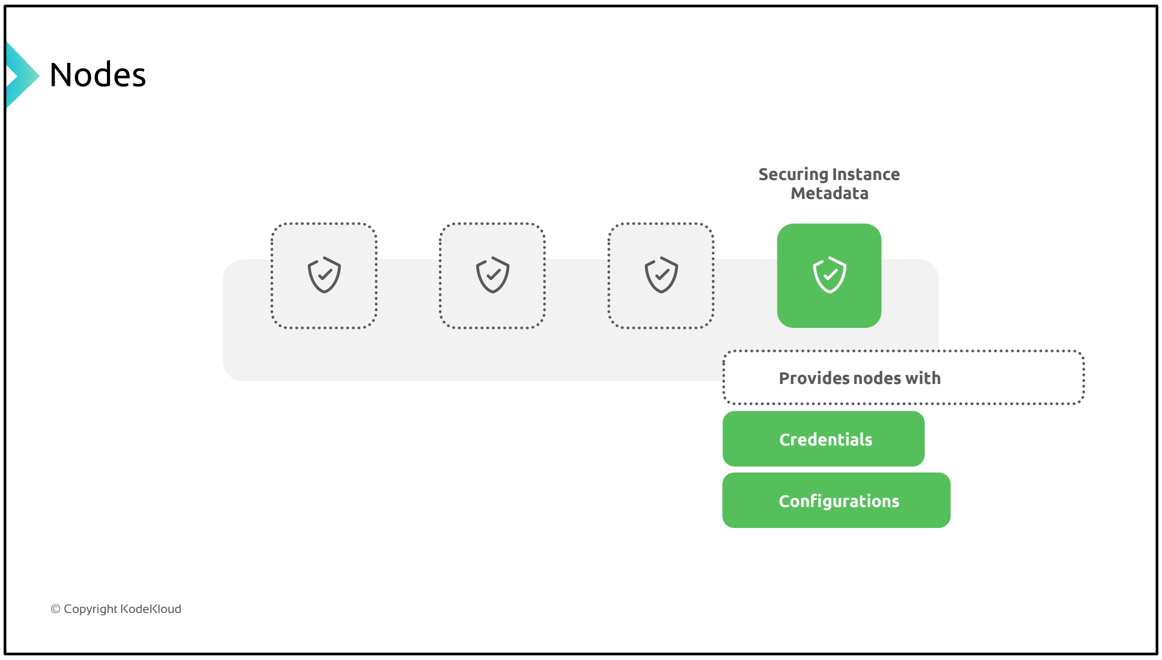


Node Upgrades

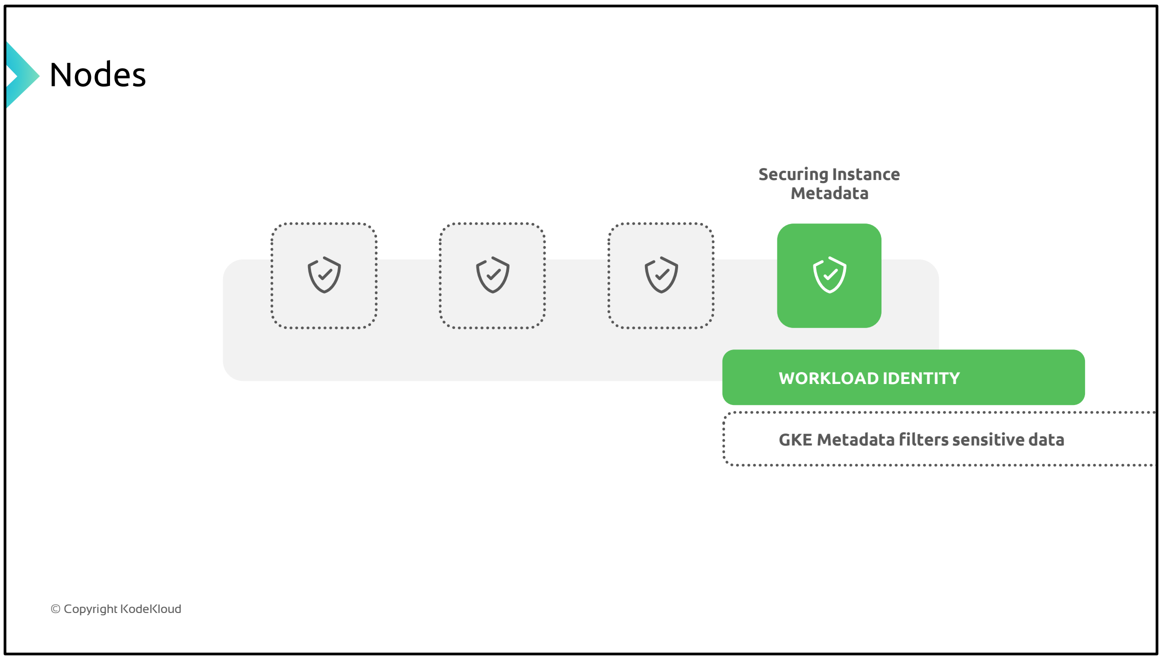
GKE clusters support **automatic upgrades**, ensuring that the **nodes of our GKE cluster are regularly updated** with the latest software versions. In Autopilot clusters, automatic upgrades are always enabled. Additionally, in Standard clusters, you can manually initiate upgrades to keep your nodes secure.


Nodes

Protecting Nodes from Untrusted Workloads
When running clusters that host unknown or untrusted workloads, it is crucial to **protect the operating system** on the node from potential threats introduced by these workloads. For instance, in multi-tenant clusters like those used by software-as-a-service (SaaS) providers or in security research environments, isolating untrusted workloads from the underlying node is essential.


Nodes

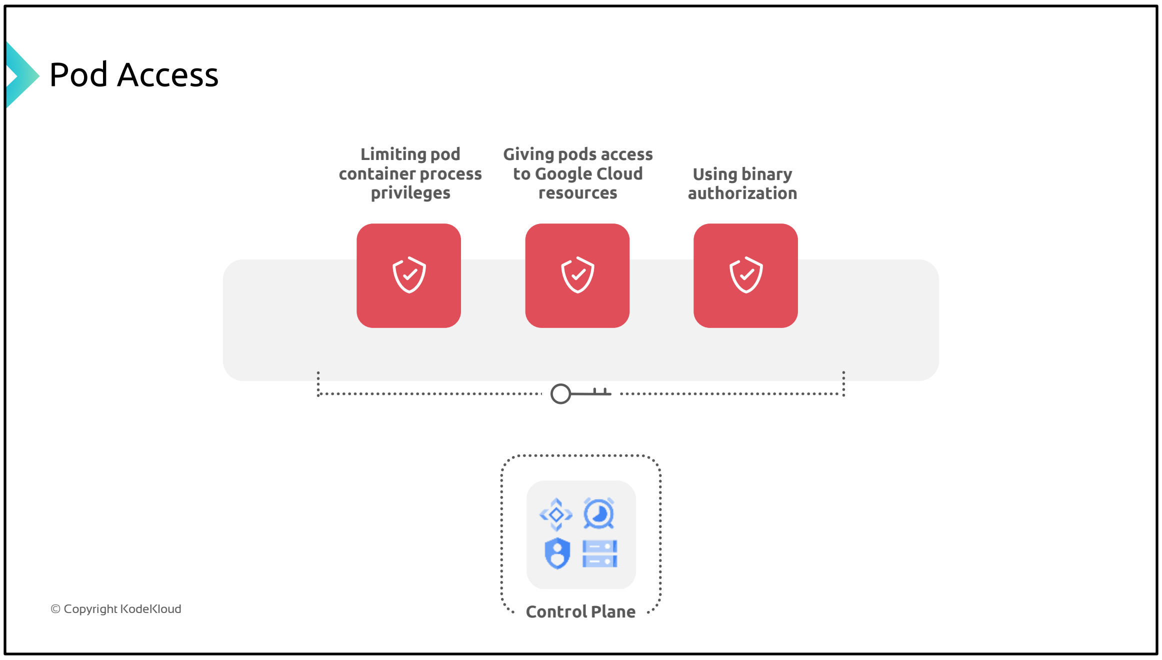
Securing Instance Metadata
GKE utilizes instance metadata from the underlying Compute Engine instances to **provide nodes with necessary credentials and configurations** for bootstrapping and connecting to the control plane. However, this metadata contains sensitive information that is not required by the Pods running on the node, such as the node's service account key.



Nodes

Securing Instance Metadata
WORKLOAD IDENTITY
GKE Metadata filters sensitive data

To ensure the security of instance metadata, **Workload Identity can be employed**. By enabling Workload Identity, the GKE metadata server in the **cluster filters requests to sensitive metadata paths**. This way, access to sensitive information that Pods don't need can be restricted, reducing the attack surface and enhancing overall security.

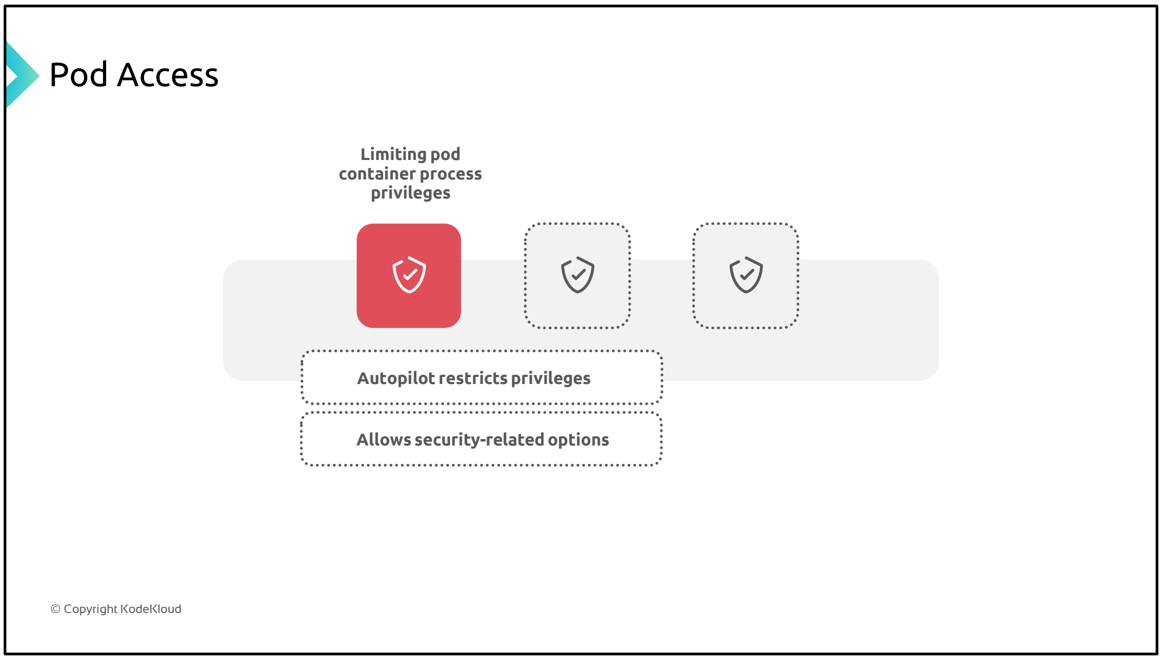


Pod Access

Limiting pod container process privileges
Giving pods access to Google Cloud resources
Using binary authorization
Control Plane

In a GKE cluster, managing the security credentials for workloads is crucial to ensure the overall security of the containerized applications. Some of the key aspects that should be considered while managing security aspects for production workload using access privileges and credentials are:

Limiting Pod Container Process Privileges
Giving Pods Access to Google Cloud Resources
Using Binary Authorization

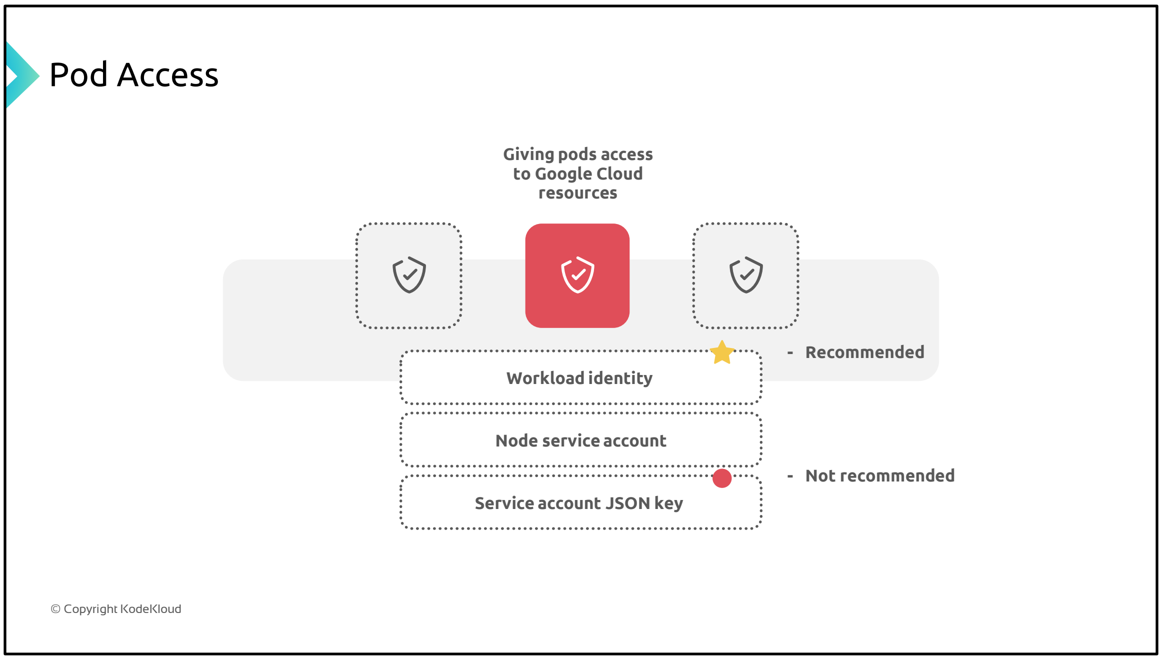


Pod Access

Limiting pod container process privileges
Autopilot restricts privileges
Allows security-related options

Limiting Pod Container Process Privileges:

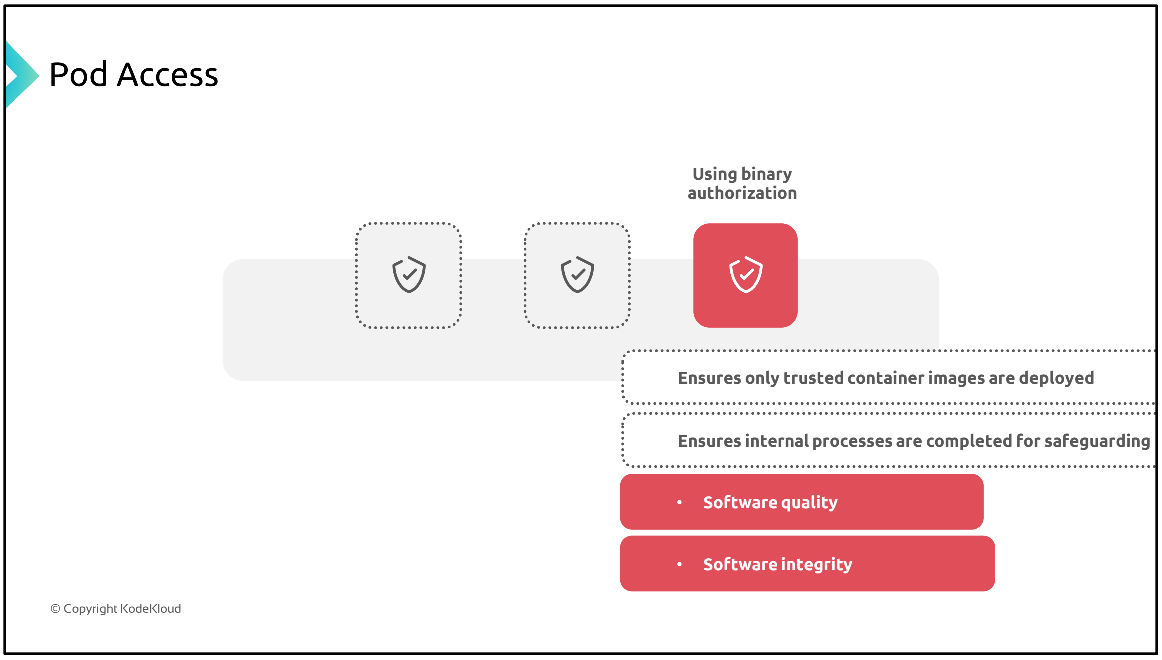
- GKE Autopilot clusters always restrict specific privileges automatically.
- GKE allows you to set security-related options via the Security Context on Pods and containers. These settings include specifying the user and group to run as, defining available Linux capabilities, and controlling the ability to escalate privileges.



Pod Access

Giving Pods Access to Google Cloud Resources:

- **Workload Identity (recommended):** The most secure way to authorize Pods to access Google Cloud resources is through Workload Identity. It allows a Kubernetes service account to run as an IAM service account, granting the Pods permissions based on the IAM service account.
- **Node Service Account:** In Standard clusters, Pods can authenticate to Google Cloud using the credentials of the service account associated with the node's Compute Engine virtual machine (VM). However, this method is not compatible with GKE Sandbox.
- **Service Account JSON Key (not recommended):** This approach involves granting credentials for Google Cloud resources to applications using a service account key. However, it is strongly discouraged due to the difficulty of securely managing account keys and should be avoided whenever possible.



Pod Access

Using binary authorization

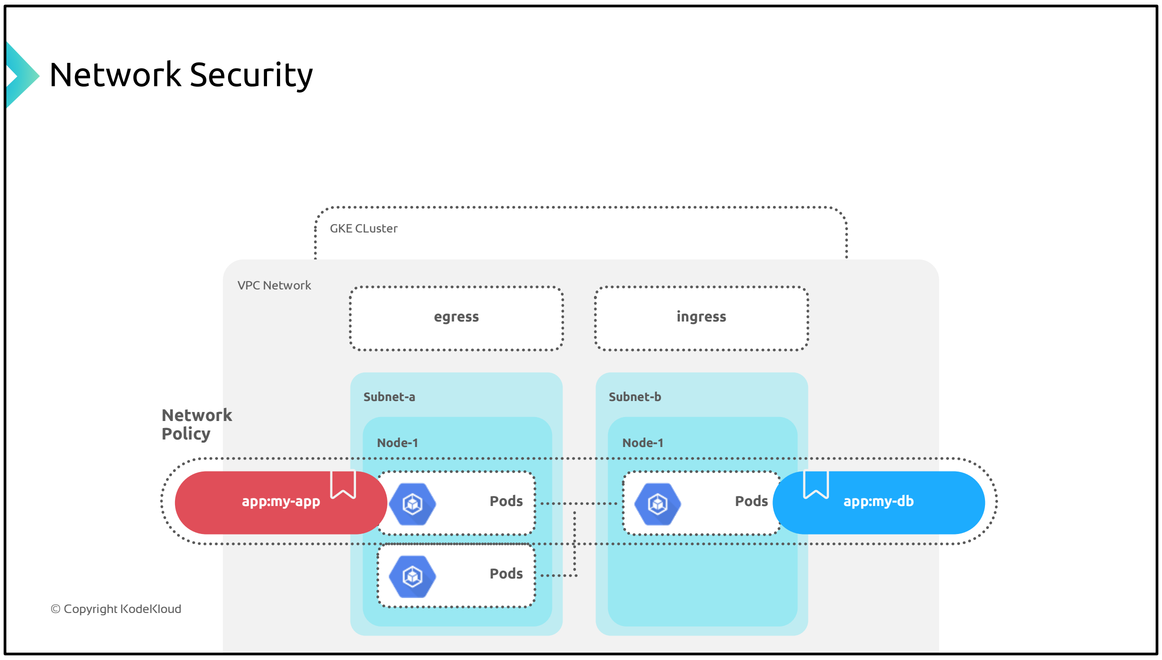
Ensures only trusted container images are deployed

Ensures internal processes are completed for safeguarding

- Software quality
- Software integrity

Using Binary Authorization:

- Binary Authorization is a Google Cloud service that **ensures only trusted container images are deployed** on Google Kubernetes Engine (GKE).
- It works with container images deployed to GKE from Artifact Registry or another container image registry.
- With Binary Authorization, you **can ensure that internal processes for safeguarding software quality and integrity are completed before deploying** applications to the production environment.



Network Security

In GKE, network security is a crucial aspect to control and secure the communication between workloads running in the cluster. Workloads often need to communicate with other services both within and outside the cluster. It's crucial to control and restrict the traffic to prevent unauthorized access and ensure the overall security of your applications and data.

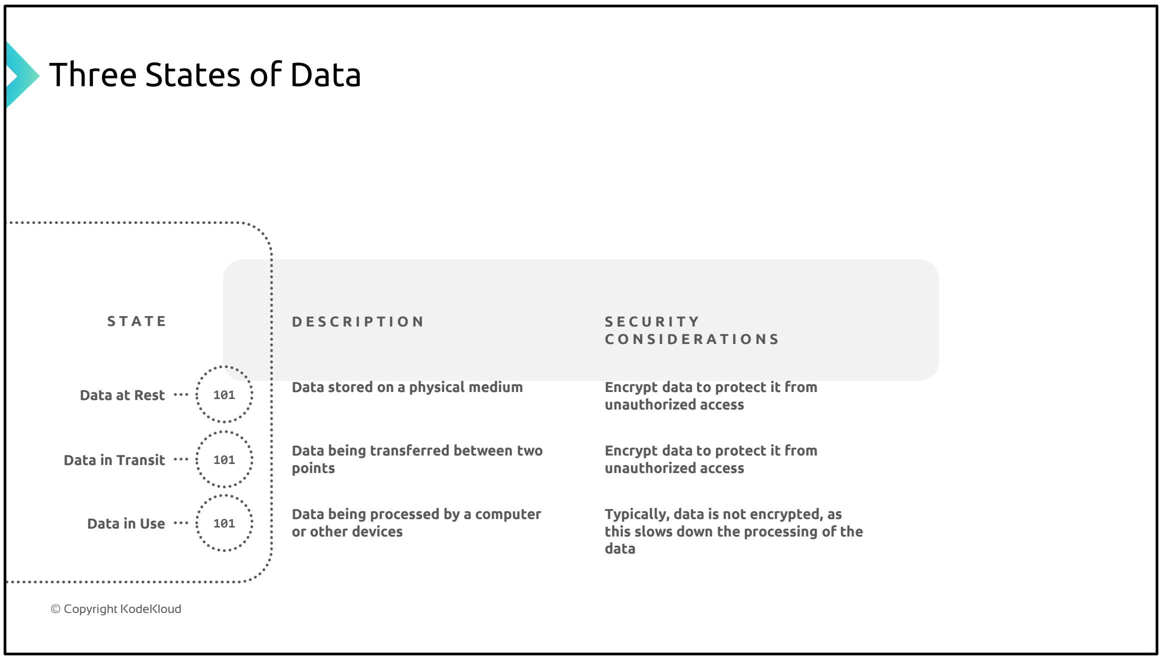
By default, all Pods in a GKE cluster can communicate with each other over the network via their Pod IP addresses. Additionally, egress traffic allows outbound connections to any address within the Virtual Private Cloud (VPC) where the cluster is deployed. To enhance security, GKE provides network policies that allows us to control ingress and egress connections to and from Pods in a cluster. Network policies define rules using labels to specify the traffic flow between Pods. When no network policies are defined, all ingress and egress traffic is allowed by default.

Let's take a look at an example of how network policies work. Let's say we have three pods inside our cluster running in a VPC. By default all the pods can communicate with each other inside that VPC. We want to restrict traffic to only allow left Pod in node-1, which is running our main app, to be able to access to pod on node-2 that has access to the data.

We can achieve this by assigning a label as "app=my-app" to the pod on node-1 and another label as "db=my-db” to the pod on node-2. We then define a network policy with these labels, and any traffic that doesn't match those labels will be dropped.

When no network policies are defined, all ingress and egress traffic is allowed by default. To ensure that all new workloads added to the cluster must explicitly authorize the traffic they require, we can apply a default deny traffic policy. This means that unless a Pod explicitly allows traffic, all traffic will be blocked by default. This adds an extra layer of security, reducing the chances of unintended access.

Control Encryption of Sensitive Data on GKE



Three States of Data

STATE	DESCRIPTION	SECURITY CONSIDERATIONS
Data at Rest 101	Data stored on a physical medium	Encrypt data to protect it from unauthorized access
Data in Transit 101	Data being transferred between two points	Encrypt data to protect it from unauthorized access
Data in Use 101	Data being processed by a computer or other devices	Typically, data is not encrypted, as this slows down the processing of the data

There are three states of data for any workload and they refer to different phases or conditions in which data can exist.

Here is a table that summarizes these three states.

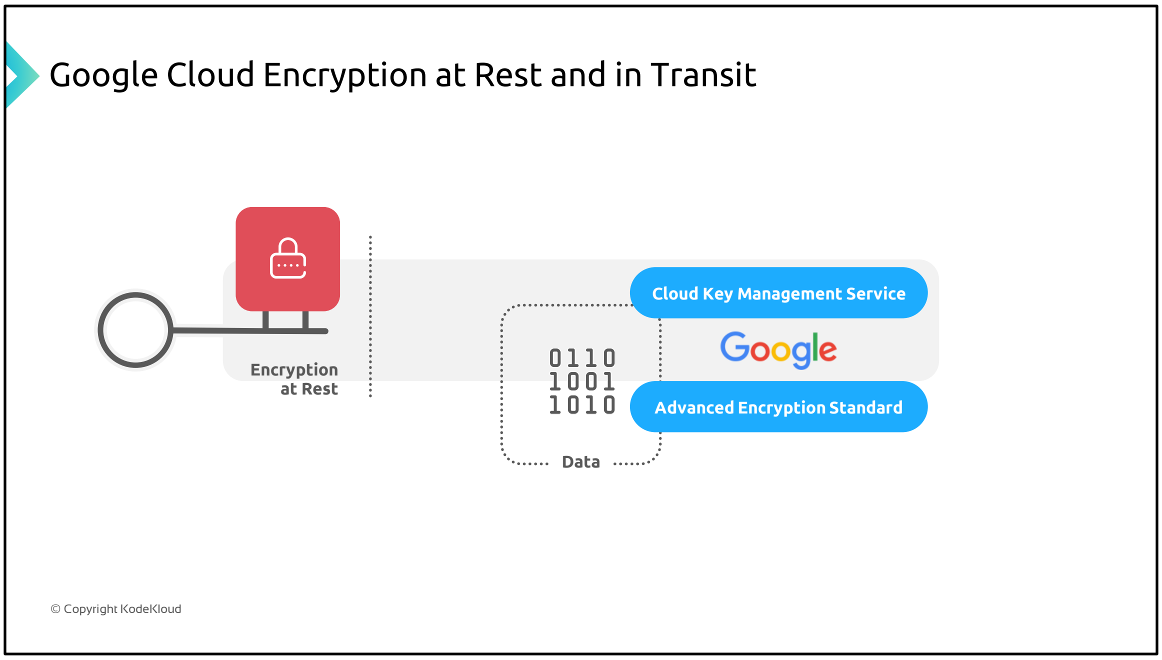
1. Data at Rest: Data at rest refers to data that is stored or persisted on a storage medium, such as a hard drive, solid-state drive, or database. This includes data stored on local disks, network-attached storage, or cloud storage services. Data at rest is typically in a dormant state and not actively

being accessed or transmitted.

2. Data in Transit: Data in transit refers to data that is being transmitted or transferred between systems or across networks. This includes data sent over the internet, private networks, or other communication channels. Examples of data in transit include emails, file transfers, web browsing, and network communications between servers.

3. Data in Use: Data in use refers to data that is actively being processed or accessed by an application, system, or user. This includes data being manipulated, analyzed, or utilized by software applications or individuals. Data in use is typically present in the memory or cache of a system during processing.

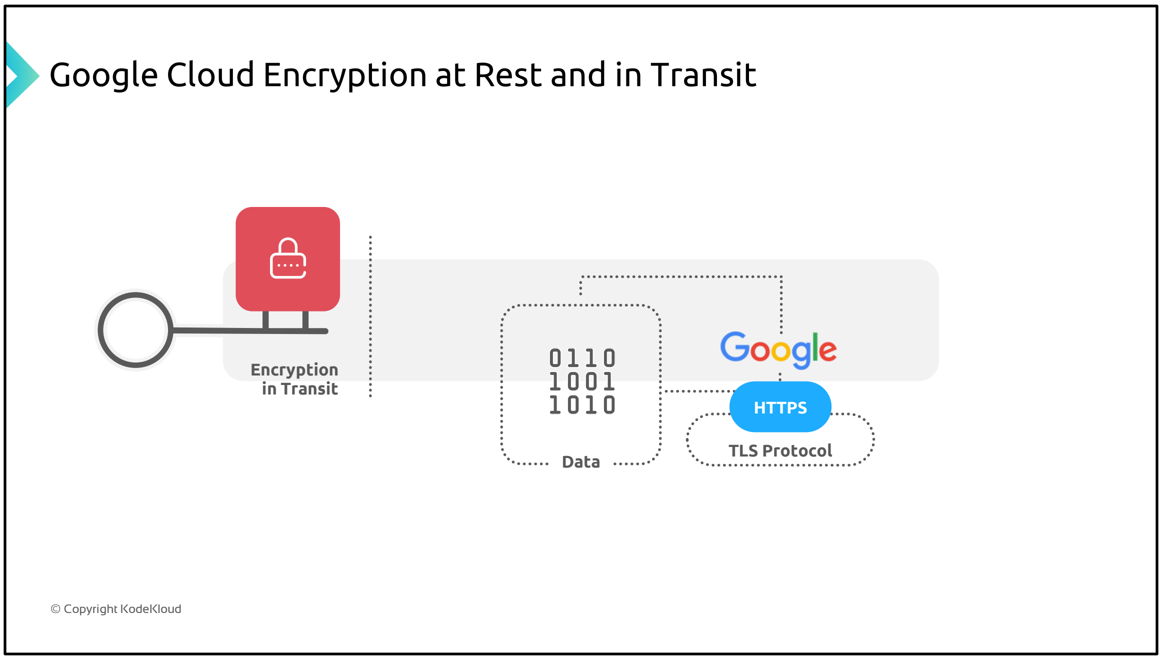
One of the key security aspects in any cloud platform is how secure the data is. One way this is achieved is through encryption. Encryption in Google Kubernetes Engine (GKE) involves encryption at rest, encryption in transit, and encryption in use, which are managed by Google to protect customer data.



Google Cloud Encryption at Rest and in Transit

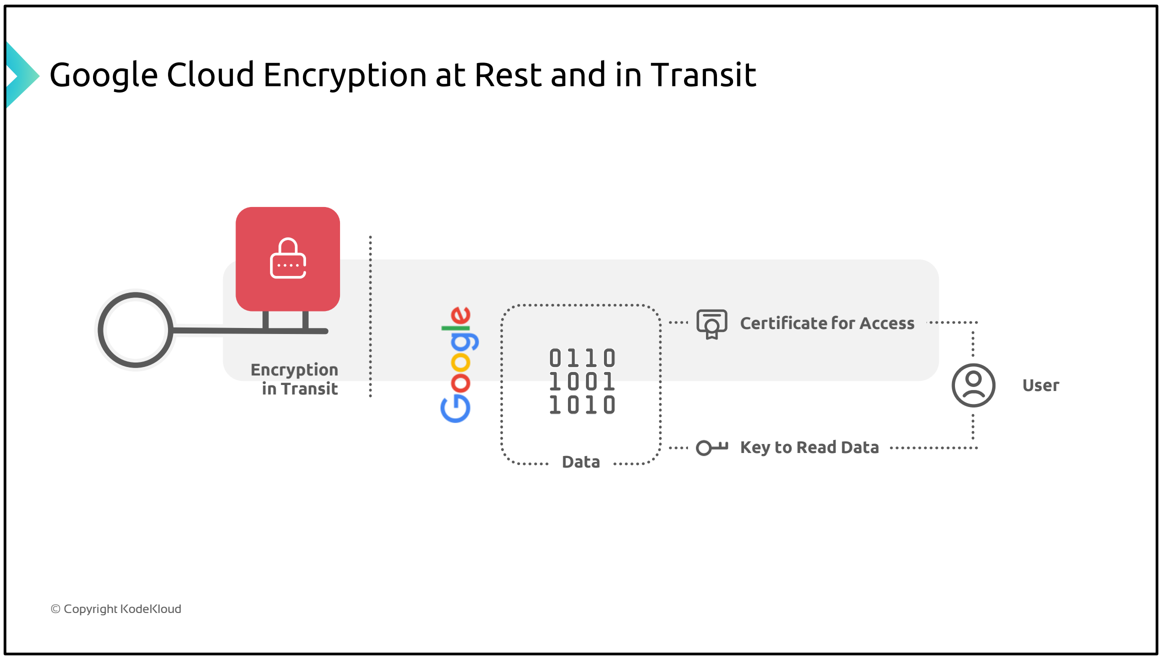
Encryption at Rest: Encryption at rest ensures that data stored on disks or backup media is protected. Google automatically encrypts all customer content at rest using one or more encryption mechanisms. In GKE, all data stored by Google is encrypted at the storage layer using the Advanced Encryption Standard (AES) algorithm with a key size of 256 bits (AES-256).

For additional controls, Google Cloud platform also offers the Cloud Key Management Service (Cloud KMS). Cloud KMS allows users to create their own encryption keys and add envelope encryption to their data. With Cloud KMS, you can create, rotate, track, and delete keys, providing an extra layer of control and security for the data in GKE.



Google Cloud Encryption at Rest and in Transit

Encryption in Transit (User to Google Front End): Encryption in transit is used to secure the data as it travels between the user and Google services. When a user sends a request to a Google Cloud service, the data is protected using HTTPS, which leverages the Transport Layer Security (TLS) protocol. HTTPS ensures the authenticity, integrity, and privacy of requests and responses.



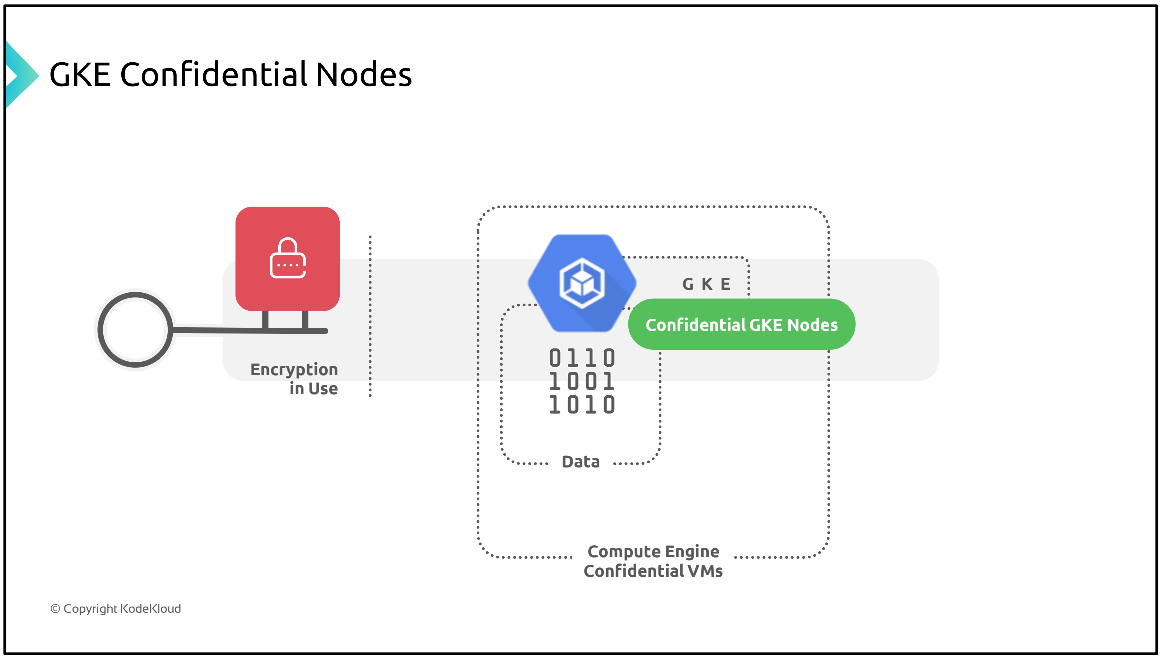
Google Cloud Encryption at Rest and in Transit

To establish HTTPS connections, the receiver requires a public-private key pair and an X.509 certificate for server authentication from a Certificate Authority (CA). Google utilizes TLS and BoringSSL, an open-source TLS library, to encrypt the data in transit. The Google Front End (GFE) servers negotiate the appropriate encryption protocol with the client based on its capabilities, prioritizing more modern encryption protocols whenever possible.

The TLS encryption implemented by Google is not limited to end-user interactions but also applies to API interactions with Google over TLS, including

Google Cloud services.

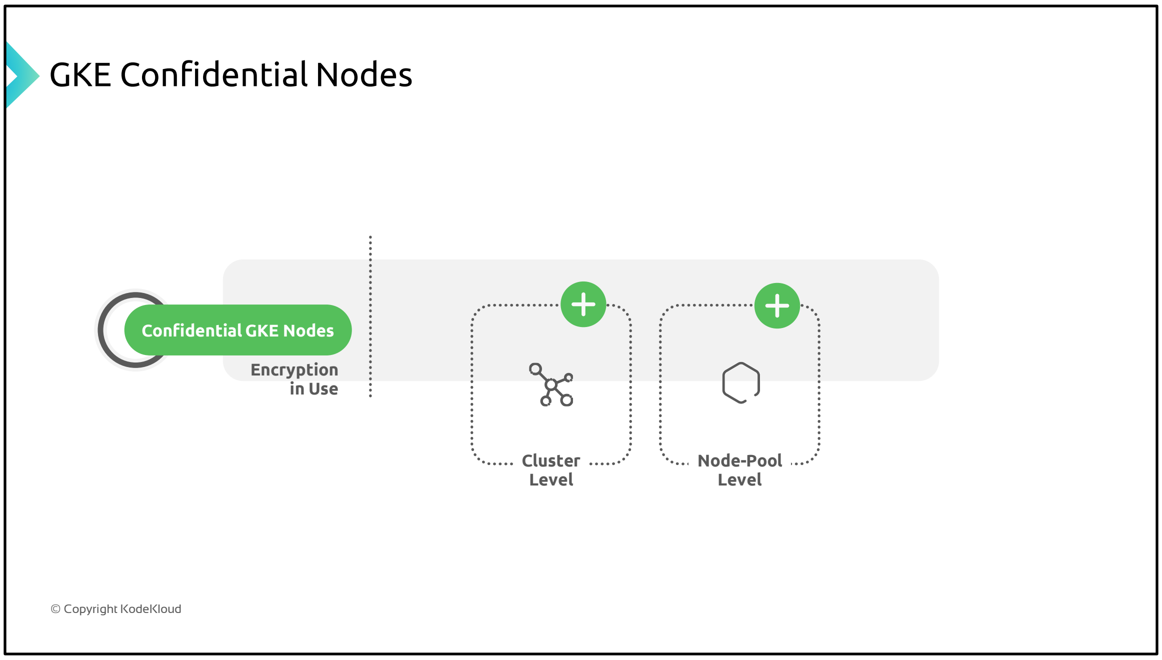
Many security features of TLS are enabled by default, such as forward secrecy. Forward secrecy ensures that the encryption key protecting a connection is not persisted, preventing an attacker who intercepts and reads one message from decrypting previous messages.



GKE Confidential Nodes

Encryption-in-use is one of the three states of end-to-end encryption, ensuring that the data is protected even while it is being processed and not just while it's at rest or in transit.

Confidential Google Kubernetes Engine (GKE) Nodes provide an additional layer of security by encrypting data in-use within your nodes and workloads. This encryption-in-use is achieved using Compute Engine Confidential VMs, which encrypt the memory contents of the virtual machines (VMs) running your workloads.

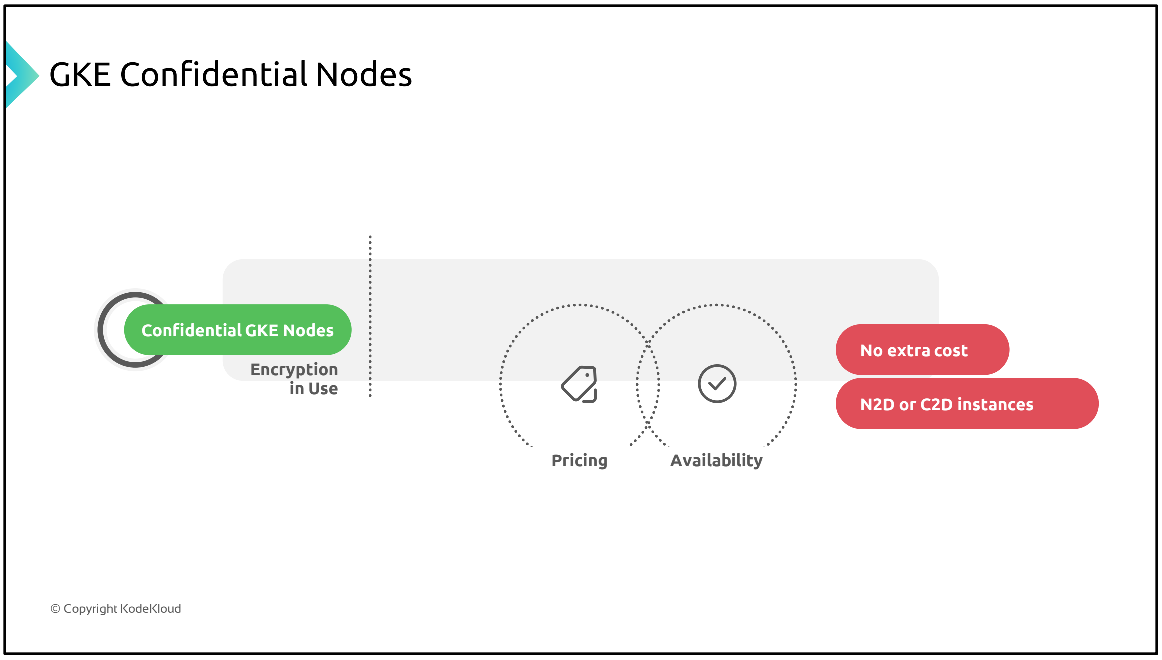


GKE Confidential Nodes

Enabling Confidential GKE Nodes: Confidential GKE Nodes can be enabled in two ways:

1. Cluster-level: When creating a new cluster, you can configure it to use Confidential GKE Nodes. This setting applies to all nodes in the cluster across all node pools. However, it's important to note that once enabled at the cluster level, you cannot disable Confidential GKE Nodes for a new or existing node pool or disable it for the entire cluster.

2. Node pool level: You can enable Confidential GKE Nodes by creating a new node pool or updating an existing node pool within a cluster. However, this option is only available if Confidential GKE Nodes are disabled at the cluster level. When enabled at the node pool level, only the nodes within that specific node pool will use Confidential GKE Nodes.



GKE Confidential Nodes

Pricing and Availability: Enabling Confidential GKE Nodes does not incur any additional cost other than the standard cost of Compute Engine Confidential VMs. However, it's worth noting that Confidential GKE Nodes may generate slightly more log data during startup compared to standard nodes.

Confidential GKE Nodes are available in zones and regions where N2D instances or C2D instances are available.