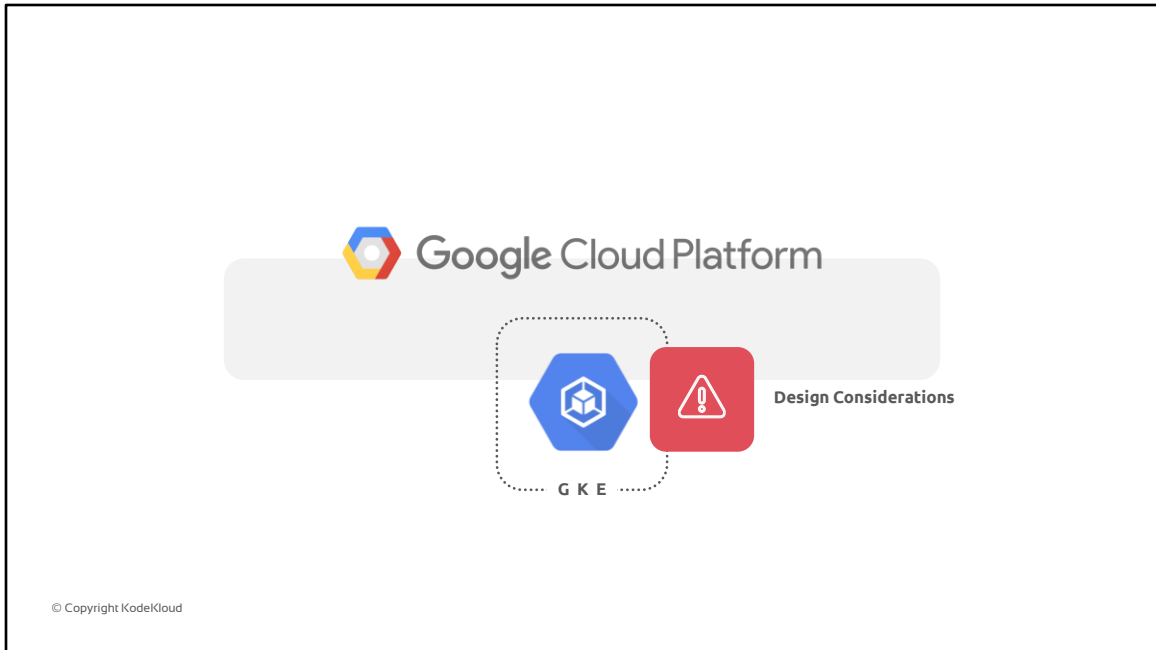


Section Introduction – GKE Design Considerations

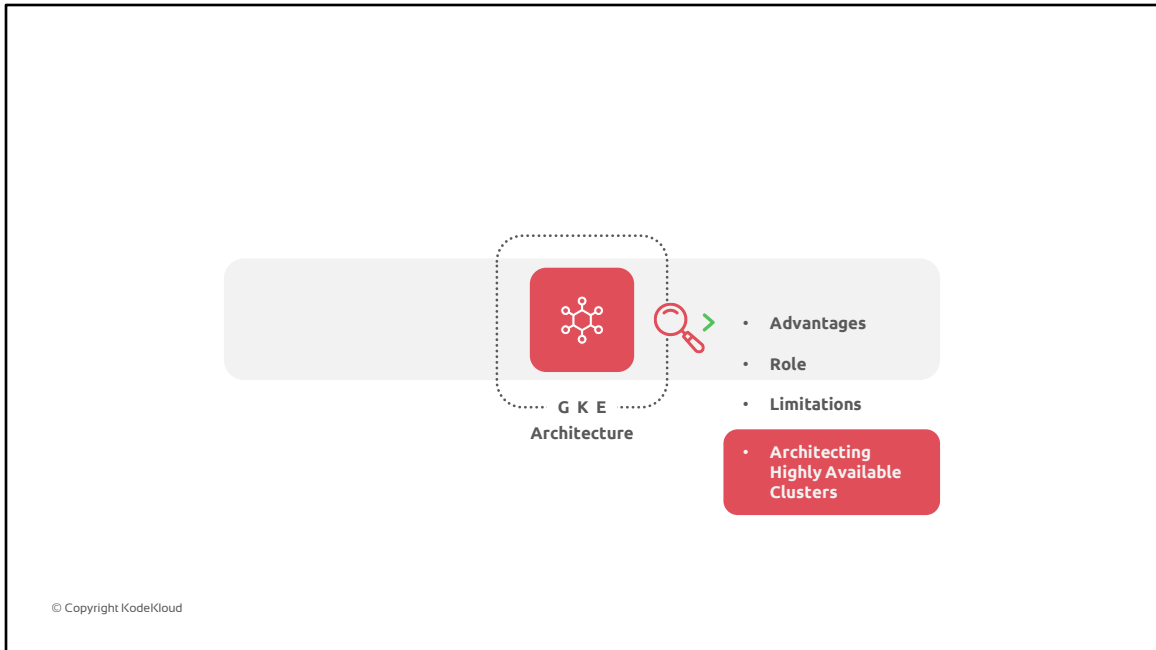
© Copyright KodeKloud



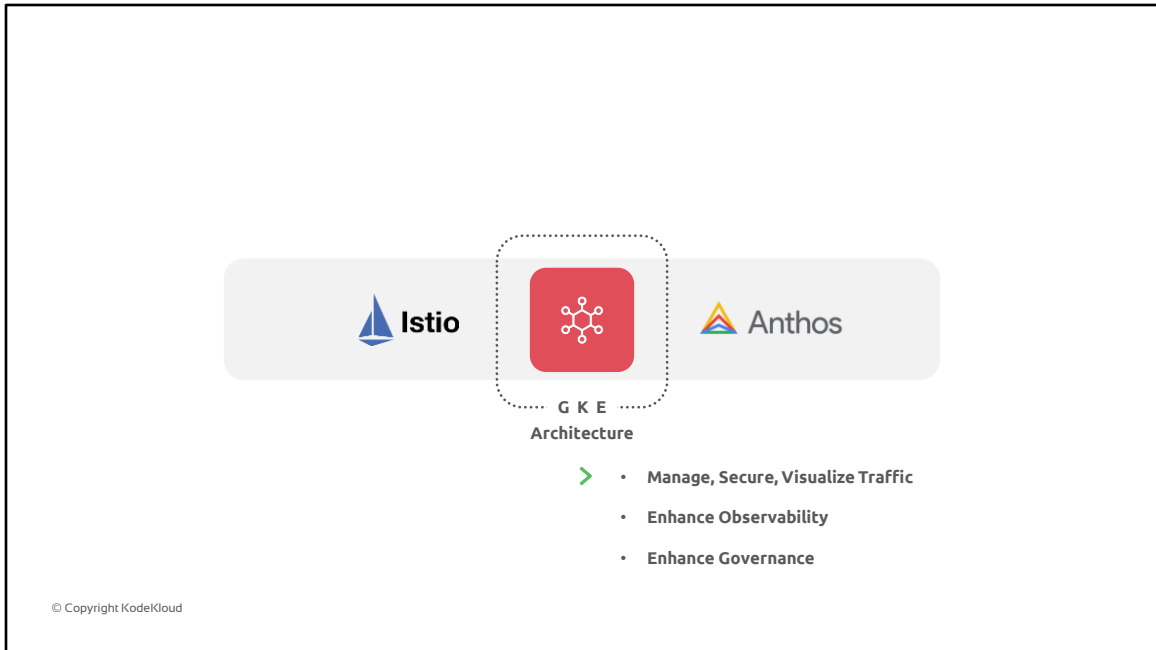
Hello fellow Google Cloud Enthusiasts, welcome to this section on GKE design considerations.



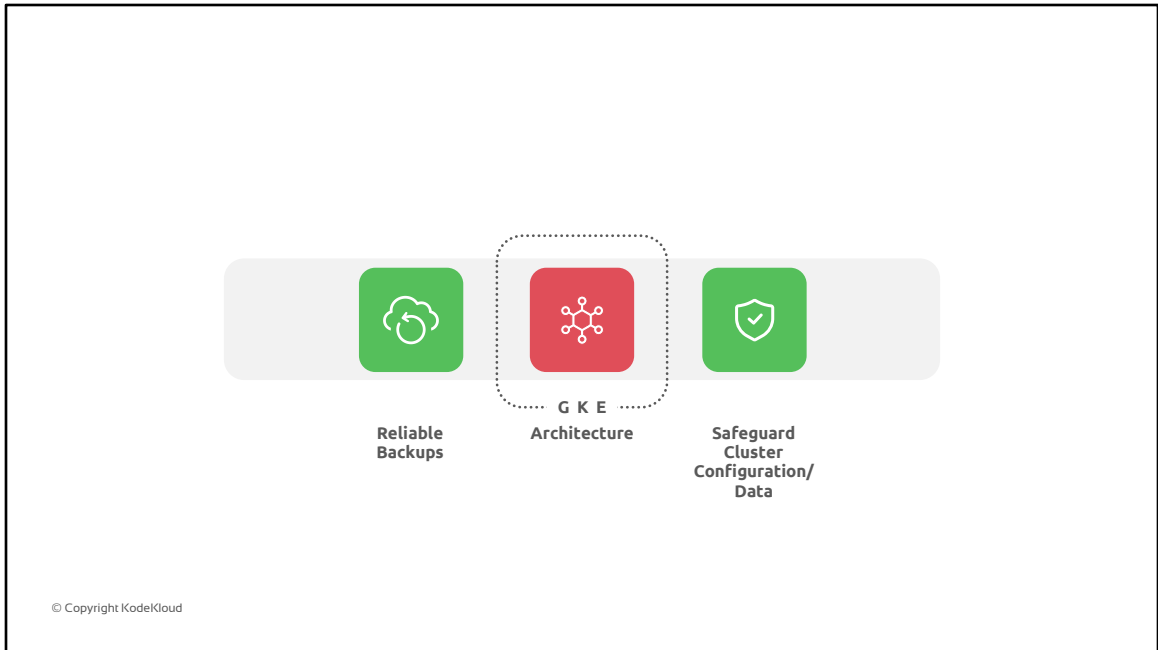
Crafting a resilient and efficient architecture for your applications on Google Kubernetes Engine (GKE) requires thorough planning and thoughtful design. This section covers some key considerations while designing a GKE environment, helping you make informed decisions to ensure optimal performance, scalability, and reliability.



We begin our exploration by delving into the advantages of regional GKE clusters, their role in enhancing application reliability, and the essential insights into their limitations. We'll also look at the ways for architecting highly available clusters, where we'll discuss crucial design considerations that encompass fault tolerance, redundancy, and effective load distribution.



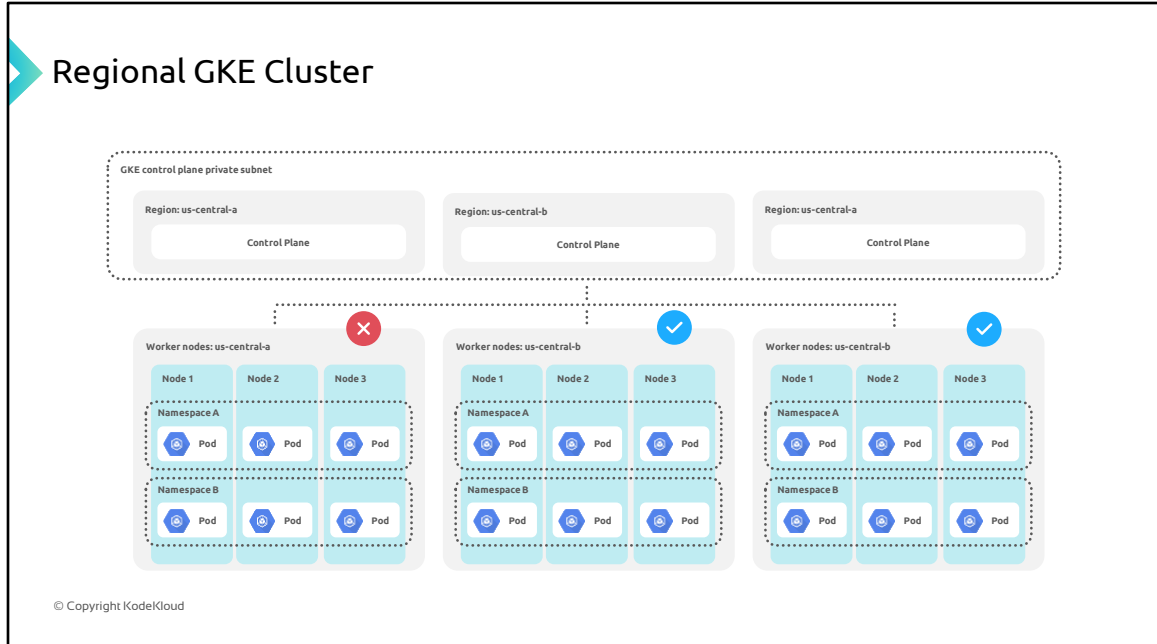
We'll also introduce some advanced networking capabilities like Istio on GKE and Anthos Service Mesh. Understand how these powerful tools empower you to manage, secure, and visualise the flow of traffic between your microservices, enhancing observability and governance in complex architectures.



We'll wrap up this section by briefly exploring strategies and best practices to create reliable backups, safeguarding your cluster's configurations and data against unforeseen events.

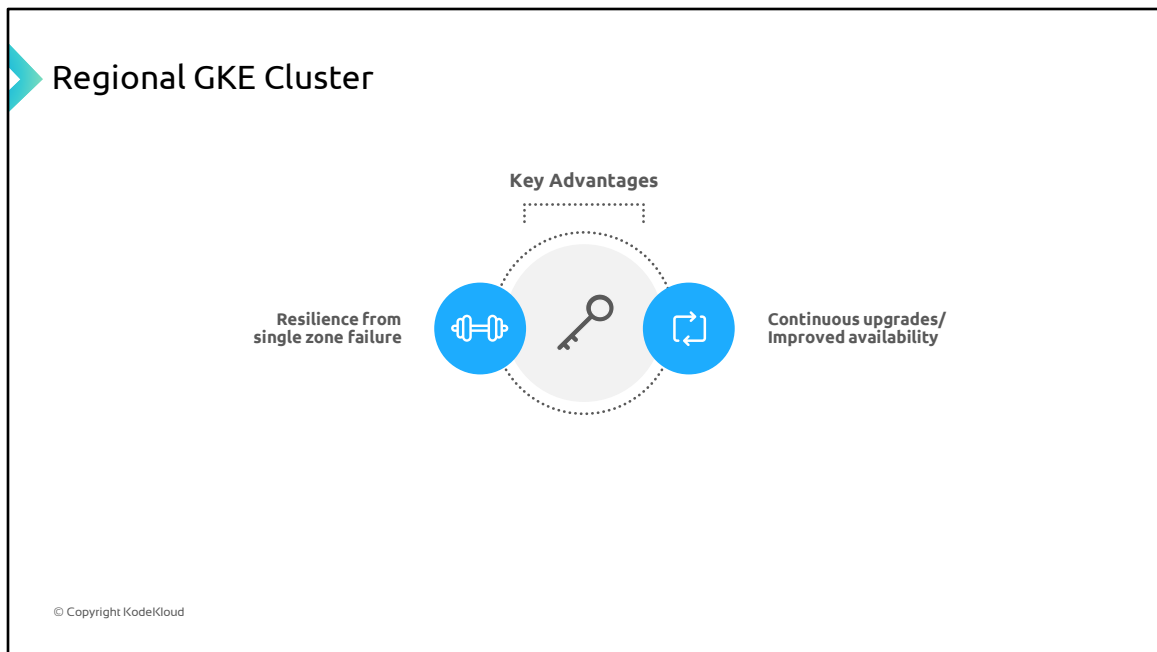
High Availability Clusters

© Copyright KodeKloud



Regional clusters in Google Kubernetes Engine (GKE) offer increased availability by replicating both the control plane and nodes across multiple zones within a specific region.

The control plane is automatically replicated across three zones within a region, ensuring that if one zone experiences an issue or becomes unavailable, the cluster's functionality remains unaffected.



Some of the key advantages that Regional clusters offer are:

- **Resilience from single zone failure:** Regional clusters are distributed across multiple zones within a region. This means that even if one zone fails, the cluster can continue operating seamlessly since there are redundant replicas of the control plane and resources in other zones.
- **Continuous control plane upgrades and improved availability:** Regional clusters enable continuous control plane upgrades and resizing without causing downtime. With multiple replicas

of the control plane, the Kubernetes API remains highly available during upgrades, ensuring uninterrupted access to the cluster.

It's recommended to use regional clusters for running production workloads due to their higher availability compared to zonal clusters.



When choosing regional clusters in Google Kubernetes Engine (GKE), there are several important factors to consider:

Default node pool configuration: The default configuration for regional Standard clusters includes nine nodes (three per zone) evenly spread across three zones in a region. This consumes nine IP addresses. If needed, this can be reduced to one node per zone.

Zones for Standard mode: For Standard mode node pools, the zones must be within the same region as the cluster's control plane. If necessary, you can change a cluster's zones, which will affect both new and existing nodes.

Regional cluster costs: Because of the higher number of nodes, Regional clusters in Standard mode require more of your project's regional quotas compared to zonal or multi-zonal clusters. Understanding the quotas and Standard pricing before using regional clusters is highly recommended to analyse the extra cost as compared to zonal clusters.

Node-to-node traffic cost: In regional clusters, if workloads in different zones

need to communicate with each other, the cross-zone traffic incurs additional cost. Review the Compute Engine pricing page for more details on egress between zones within the same region.



Overprovisioning scaling limits: To ensure availability during zonal failures, you can allow GKE to overprovision scaling limits. By specifying a higher maximum number of nodes per zone, the cluster can maintain capacity even if some zones become unavailable. For example, overprovisioning a three-zone cluster to 150% means allowing a maximum of six nodes per zone. So, if one zone fails, the cluster scales up to 12 nodes in the remaining zones.

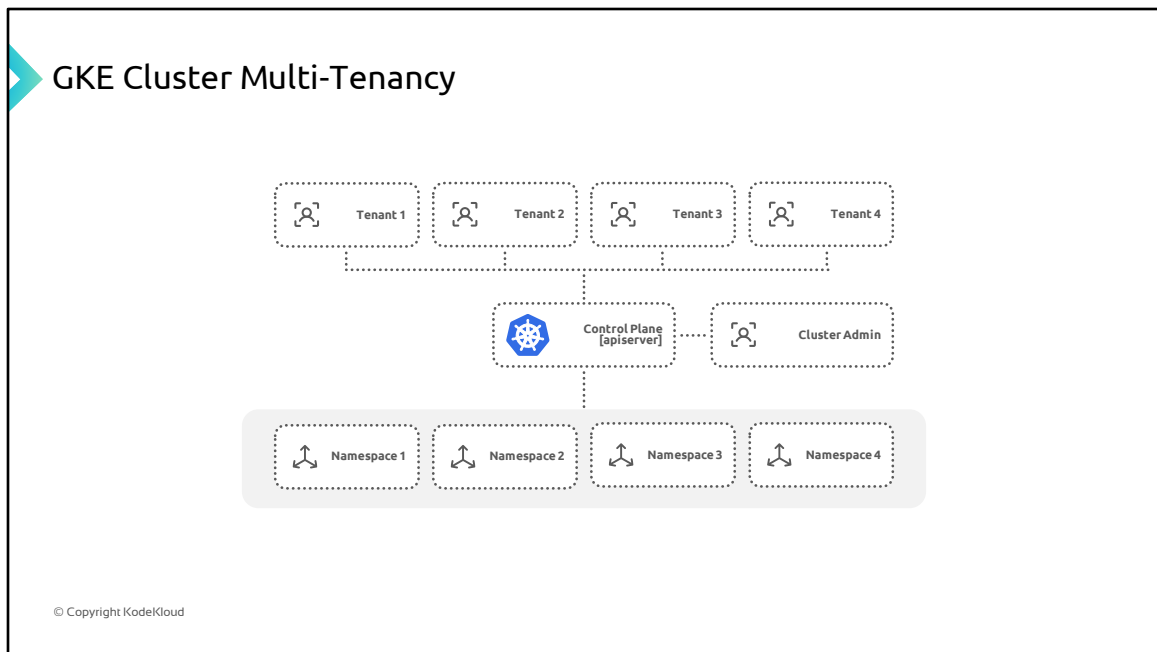
For more information on how to design a highly available workload environment, refer to the resource provided in the description:

[Best practices for creating a highly available GKE cluster | Google Cloud Blog](#)

[Design for scale and high availability | Architecture Framework | Google Cloud](#)

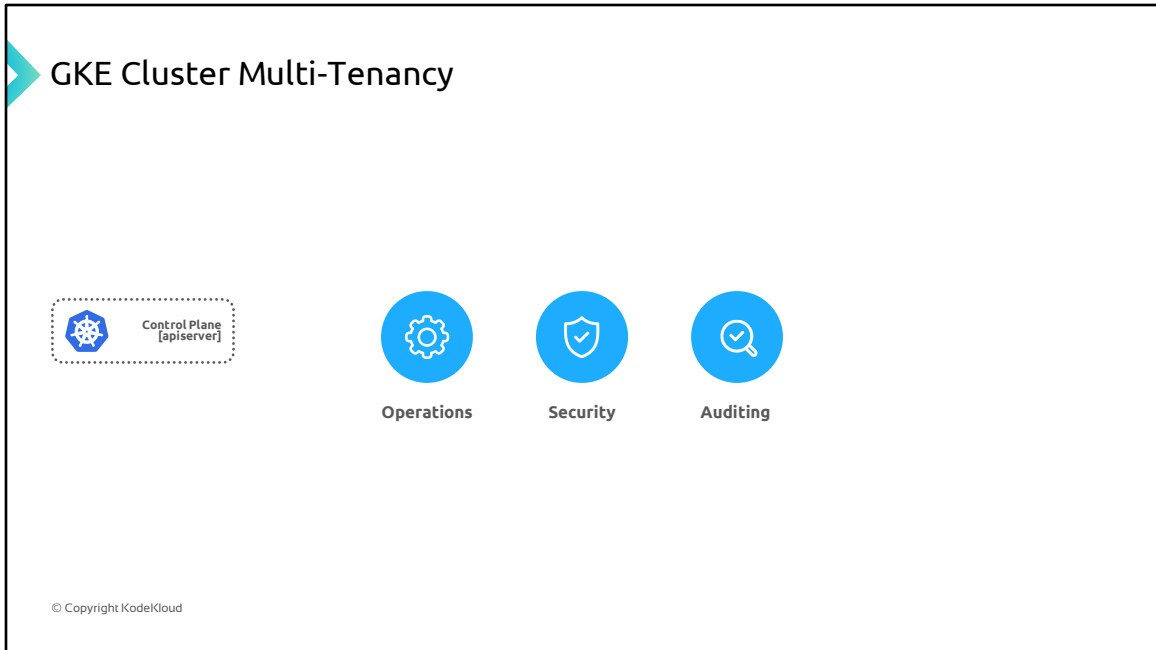
Cluster Multi-Tenancy

© Copyright KodeKloud

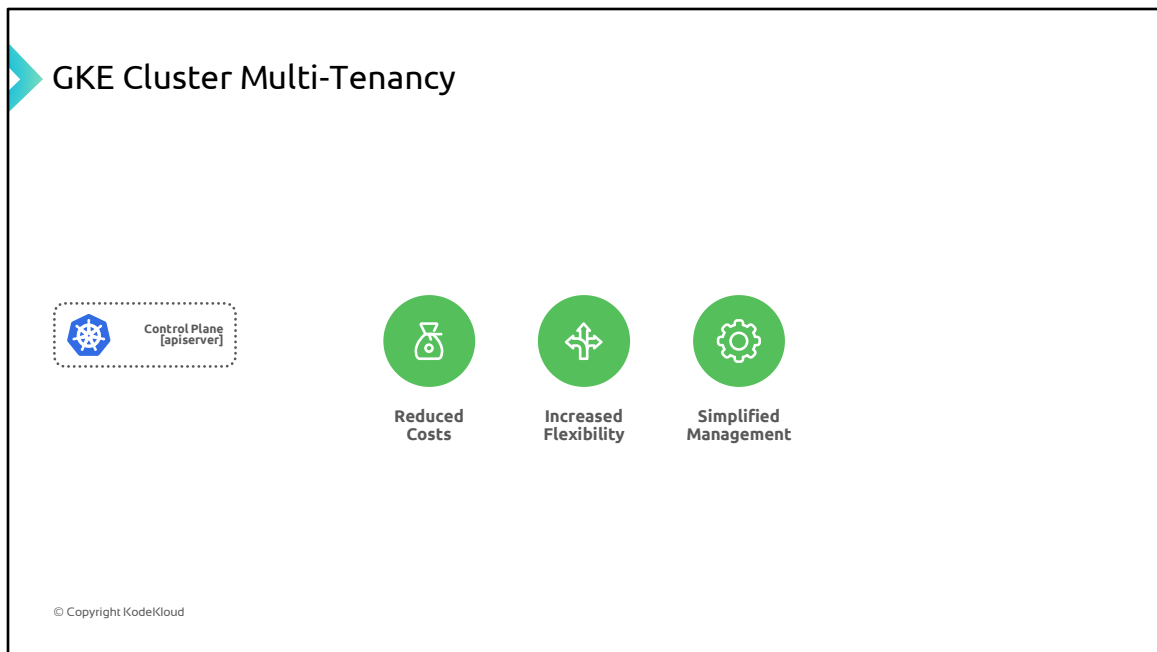


Multi-tenancy is a deployment model where a single Kubernetes cluster is shared by multiple users and/or workloads. Each user or workload is called a "tenant". GKE cluster multi-tenancy is a feature of Google Kubernetes Engine (GKE) that allows you to create and manage multi-tenant clusters.

The tenants of a multi-tenant cluster share some of the resources of GKE cluster like extensions, controllers, and add-ons.

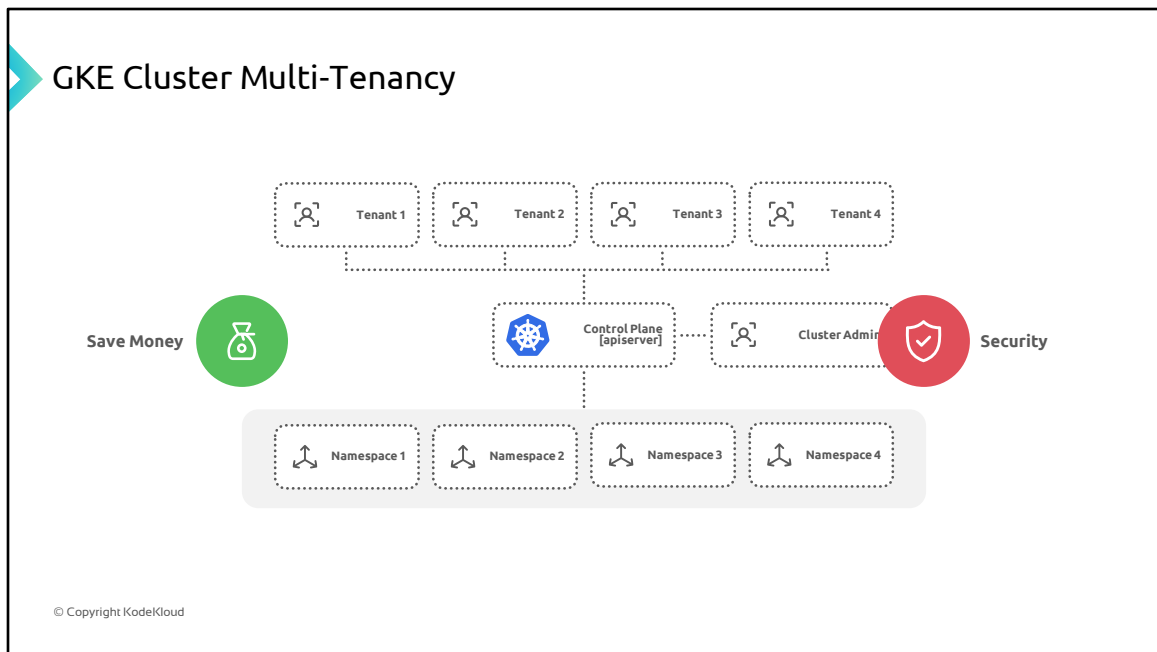


The cluster control plane of the cluster is also shared, centralising the cluster operations, security, and auditing.



Some factors why a multi-tenant cluster can be considered are:

- Reduced costs:** You can save money by sharing resources across tenants.
- Increased flexibility:** You can easily add or remove tenants as needed.
- Simplified management:** You can manage all of your tenants from a single console.



while considering using a GKE cluster multi-tenancy, the organization's overall needs and requirements should be carefully considered. Multi-tenancy can be a good option if you need to save money on your Kubernetes deployments. However, it is important to understand the security implications of multi-tenancy and to implement appropriate controls to protect your data.

When designing a multi-tenant architecture, it is important to consider different layers of resource isolation within Kubernetes, such as cluster,


namespace, node, Pod, and container. Each layer provides a level of isolation and security for tenants. For example, you can use namespaces to separate tenants and their resources, and enforce policies to control access, resource usage, and container behaviour within each namespace.

▶ Best Practices for GKE Cluster Multi-Tenancy

Best Practice

Best practices for enterprise multi-tenancy
cloud.google.com/kubernetes-engine/docs/best-practices/enterprise-multitenancy

© Copyright KodeKloud

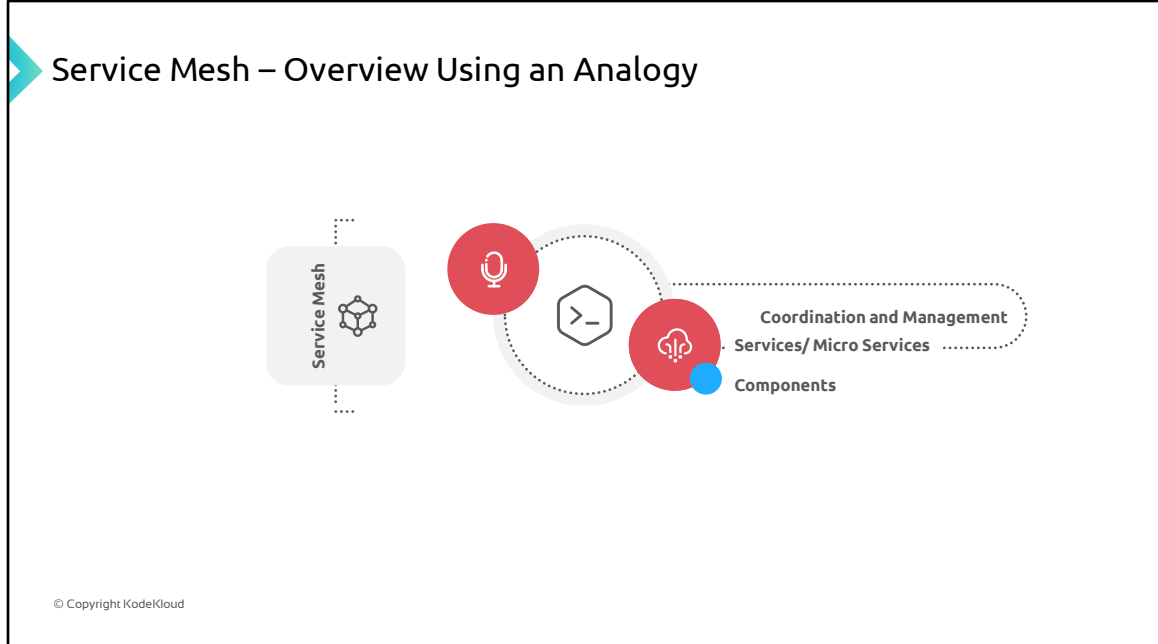


For additional guidance on setting up multi-tenant clusters in an enterprise organization, refer to the "Best practices for enterprise multi-tenancy" documentation, which provides further insights and recommendations tailored specifically for enterprise use cases.

<https://cloud.google.com/kubernetes-engine/docs/best-practices/enterprise-multitenancy>

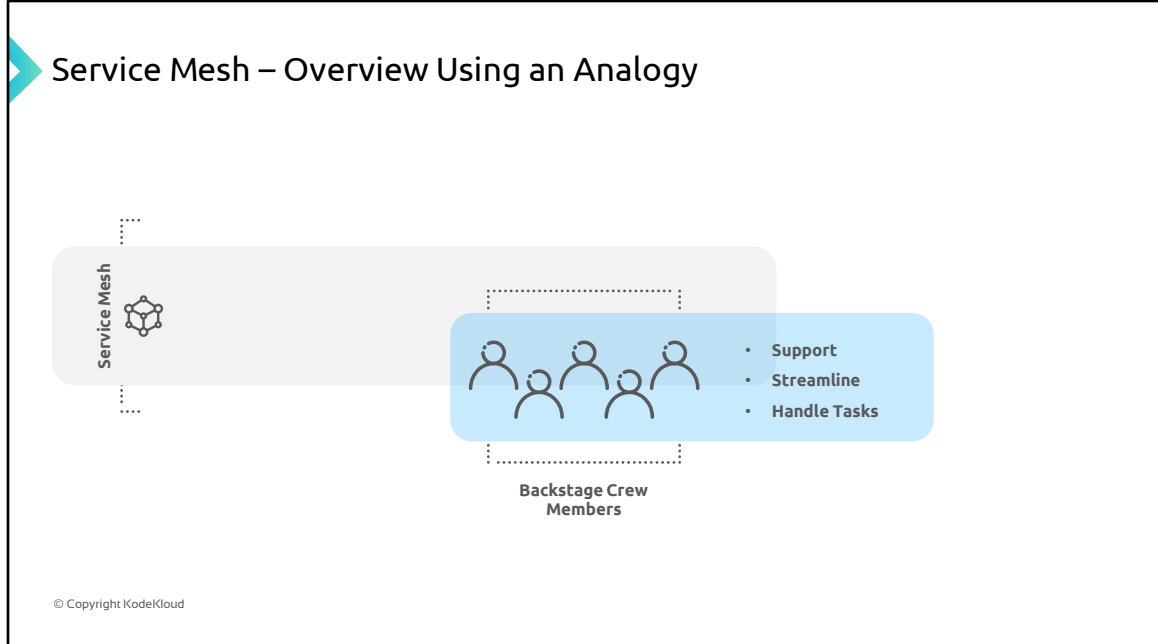
Service Mesh

© Copyright KodeKloud

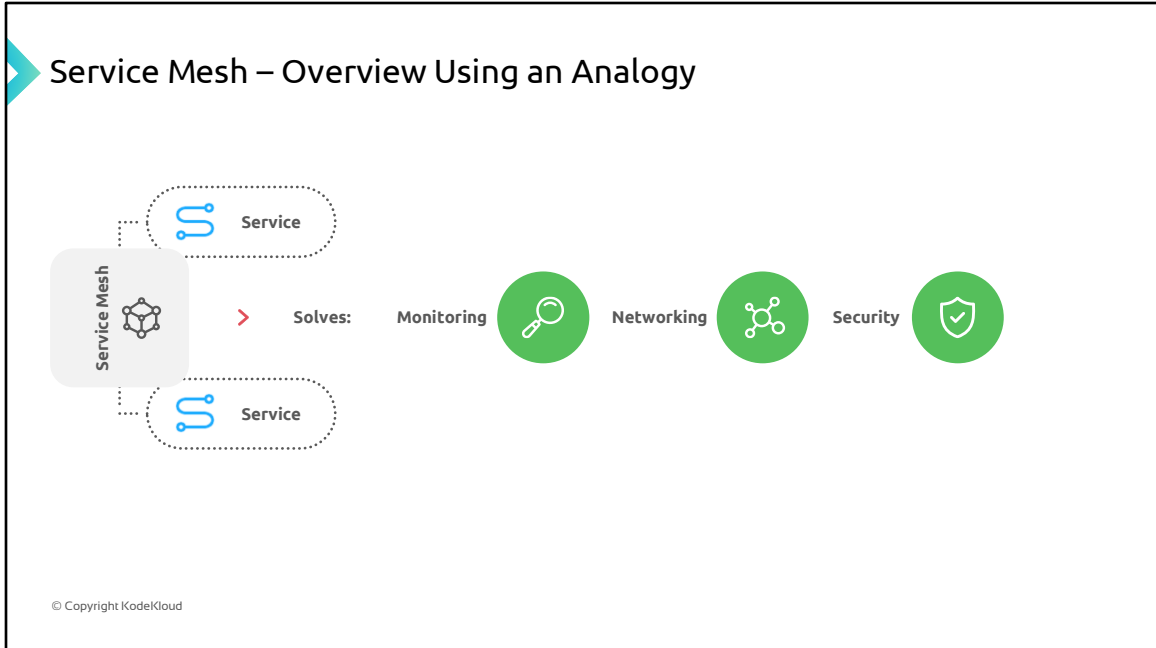


Imagine you're running a large-scale event, like a music festival, with multiple stages and performers. Organising such an event requires careful coordination and management to ensure everything runs smoothly. In this analogy, the event is your application, the stages are the different services or microservices that make up your application, and the performers are the individual components within each service.

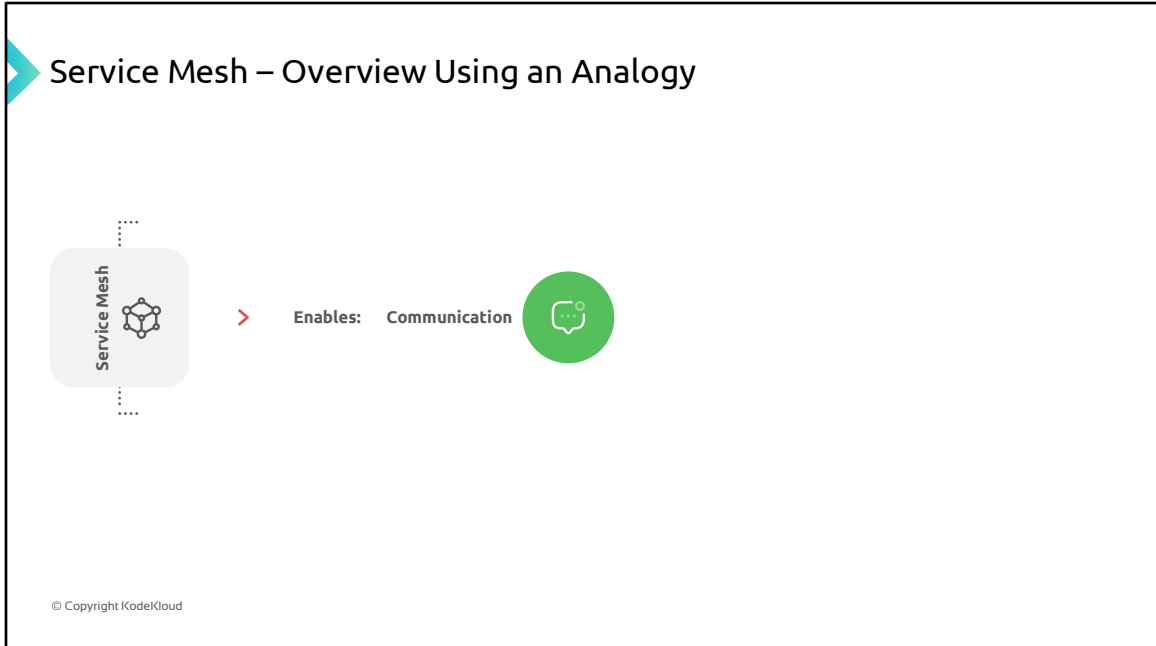
Now, let's introduce the concept of a service mesh.



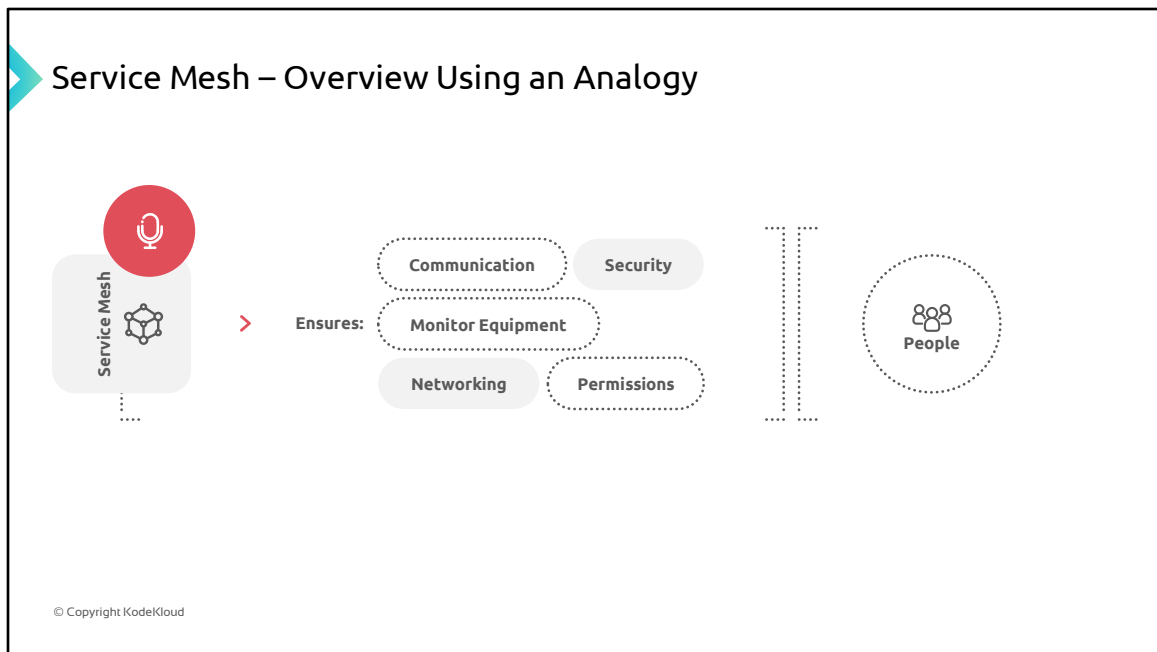
Think of a service mesh as a dedicated team of backstage crew members who work behind the scenes to support and streamline the entire event. They handle various tasks that are common to all stages and performers, allowing them to focus on their specific roles without worrying about the logistics and technicalities.



Similarly, a service mesh acts as a dedicated infrastructure layer that sits alongside your services and takes care of common concerns, such as monitoring, networking, and security.



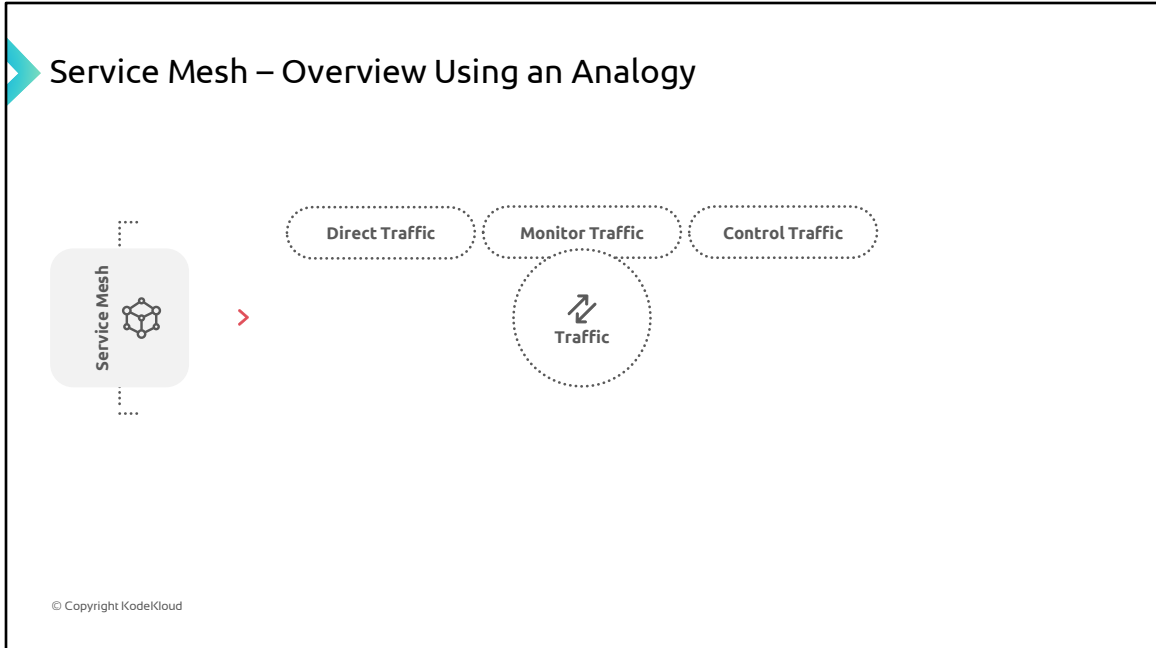
It provides a set of tools and features that enable developers and operators to easily manage and control communication between services.



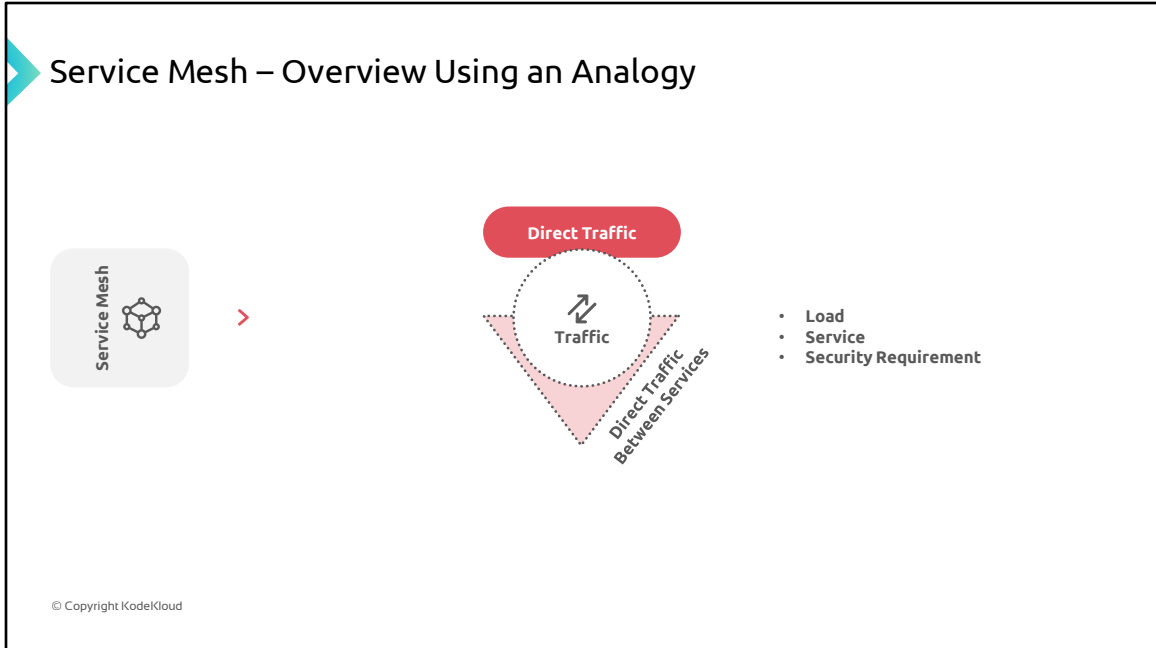
In our festival analogy, the service mesh team would ensure that all stages have proper **monitoring equipment** to track the performance metrics of each artist. They would handle the **networking** aspect, ensuring smooth **communication** between stages, like coordinating sound systems and managing traffic flow. They would also ensure the **security** of the event, monitoring access points and **managing permissions** for backstage areas.

By decoupling these common concerns from the individual services, just as the service mesh team

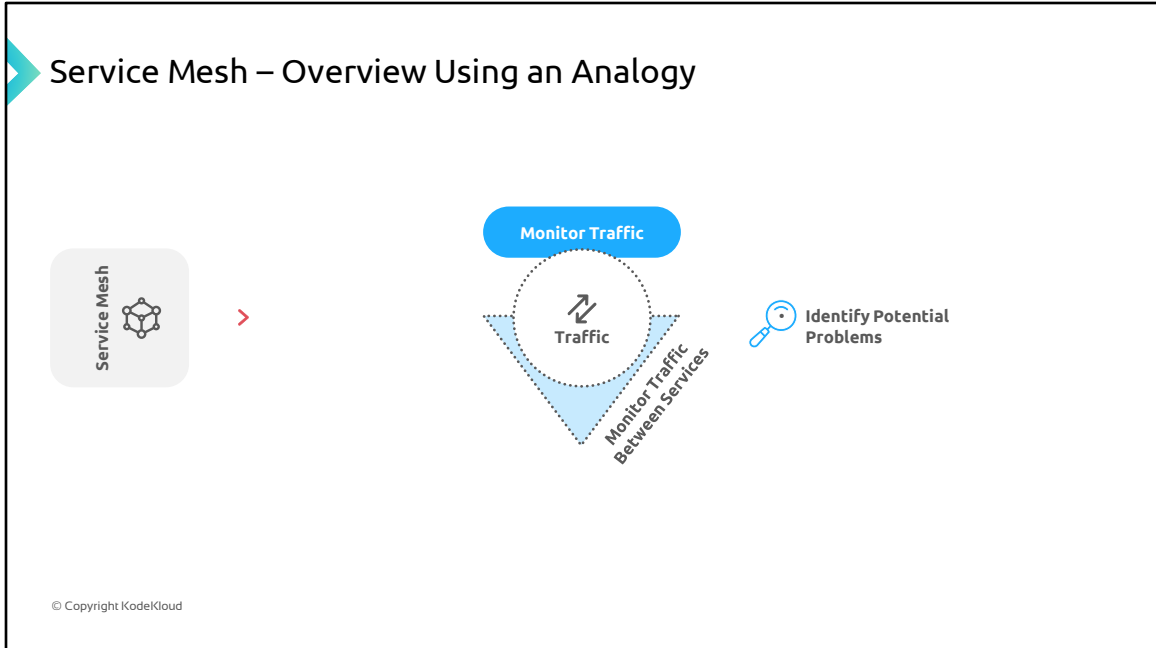
handles them separately from the performers, developers and operators can focus on creating and managing great applications. They don't need to worry about reinventing the wheel for each service, but instead leverage the consistent and powerful tools provided by the service mesh.



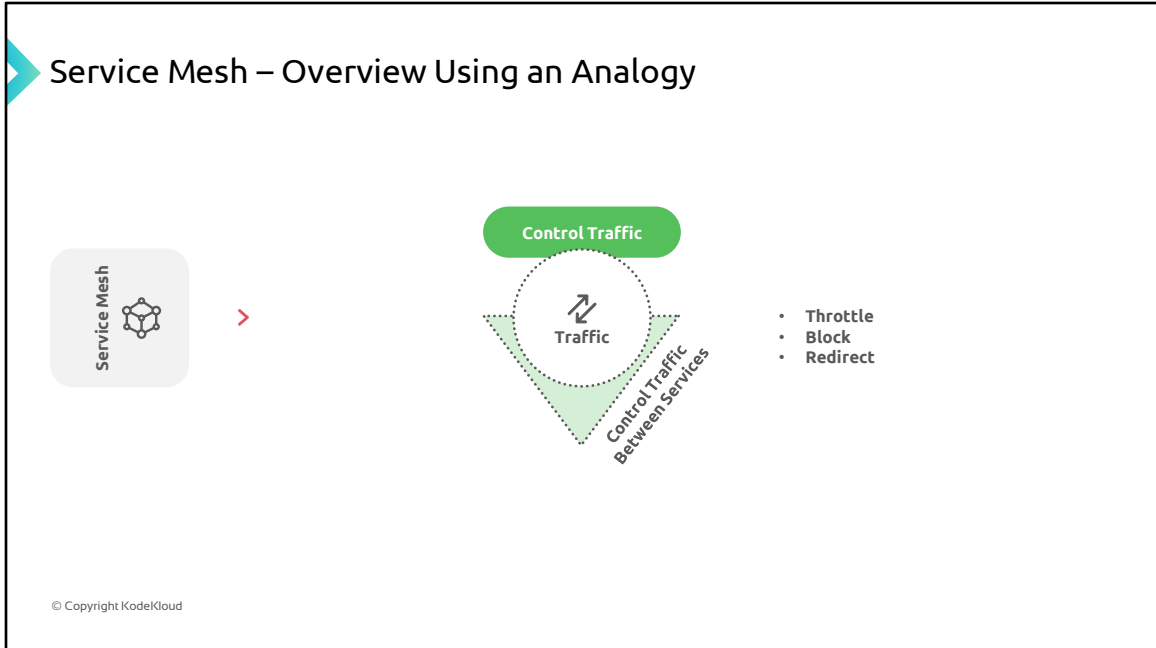
Just like the backstage crew, a service mesh can:



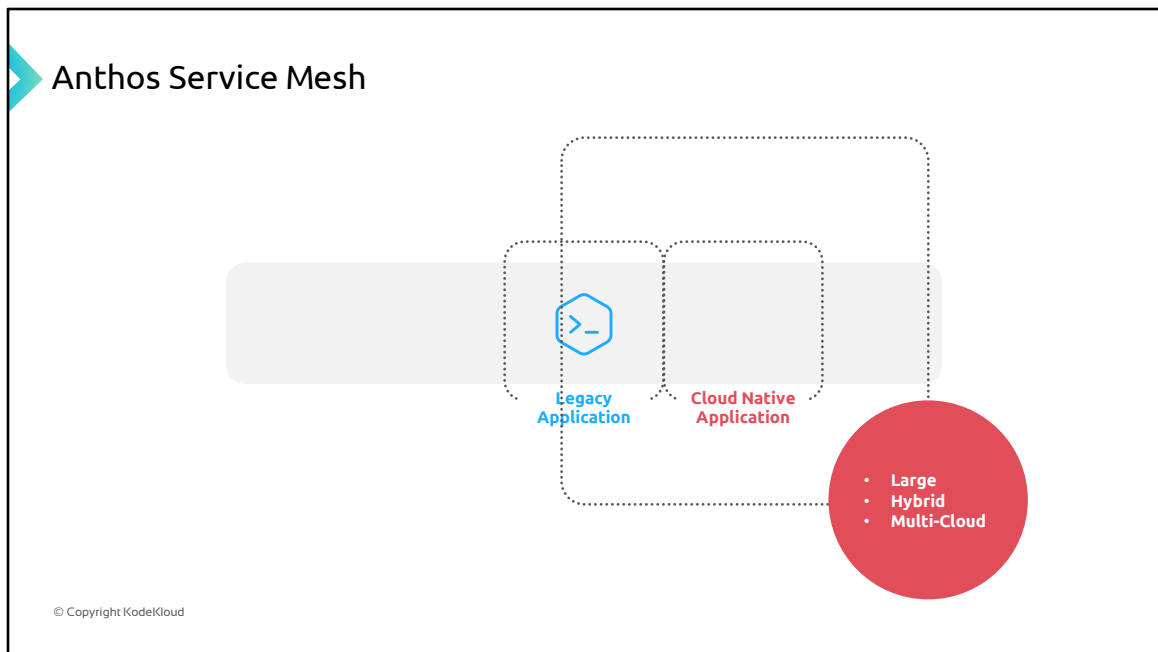
• **Direct traffic:** A service mesh can direct traffic between your services based on a variety of factors, such as the load on each service, the availability of each service, and the security requirements of the traffic.



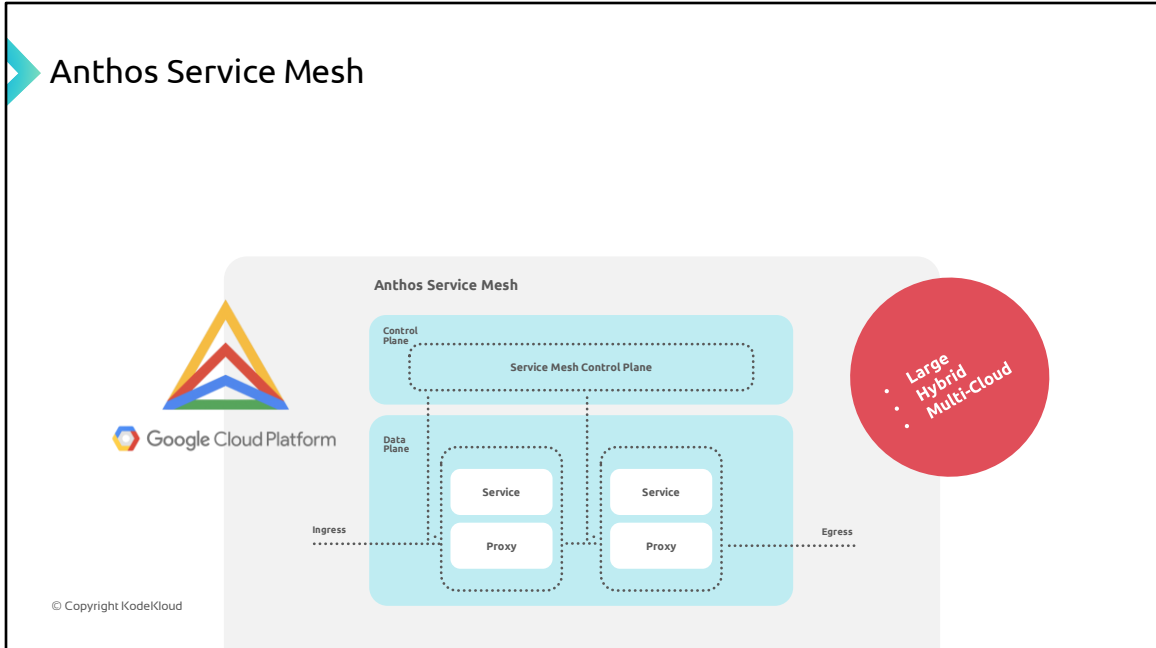
Monitor traffic: A service mesh can monitor all traffic between your services, so you can see how your applications are performing and identify any potential problems.



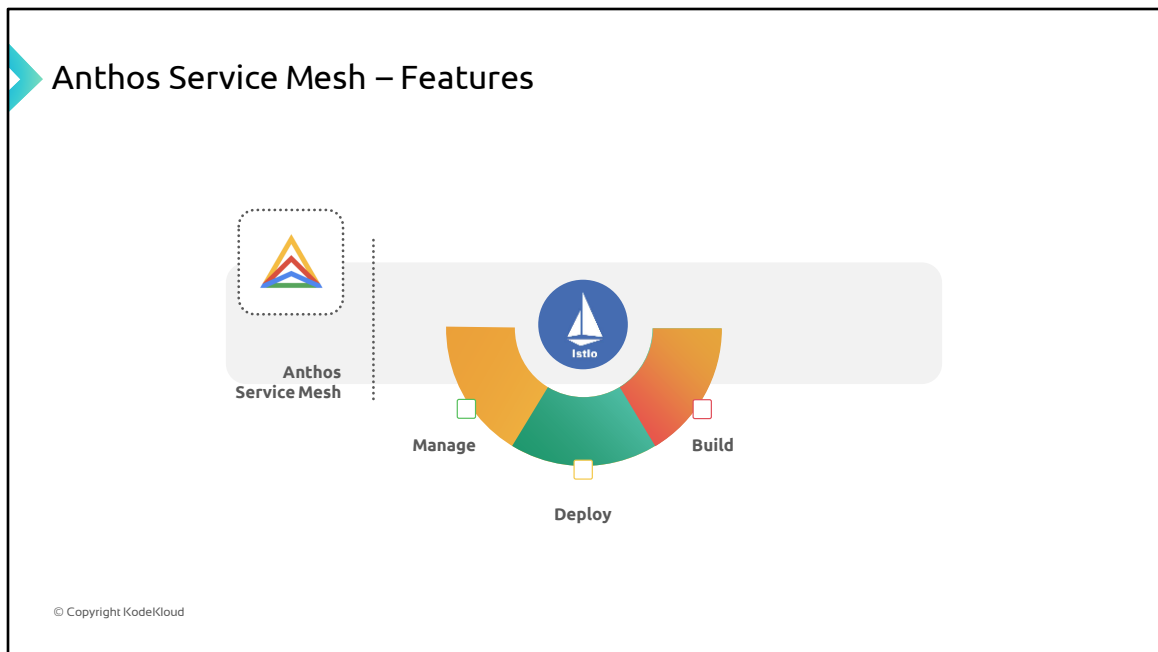
- **Control traffic:** A service mesh can control traffic between your services, so you can throttle traffic, block traffic, or even redirect traffic if necessary.



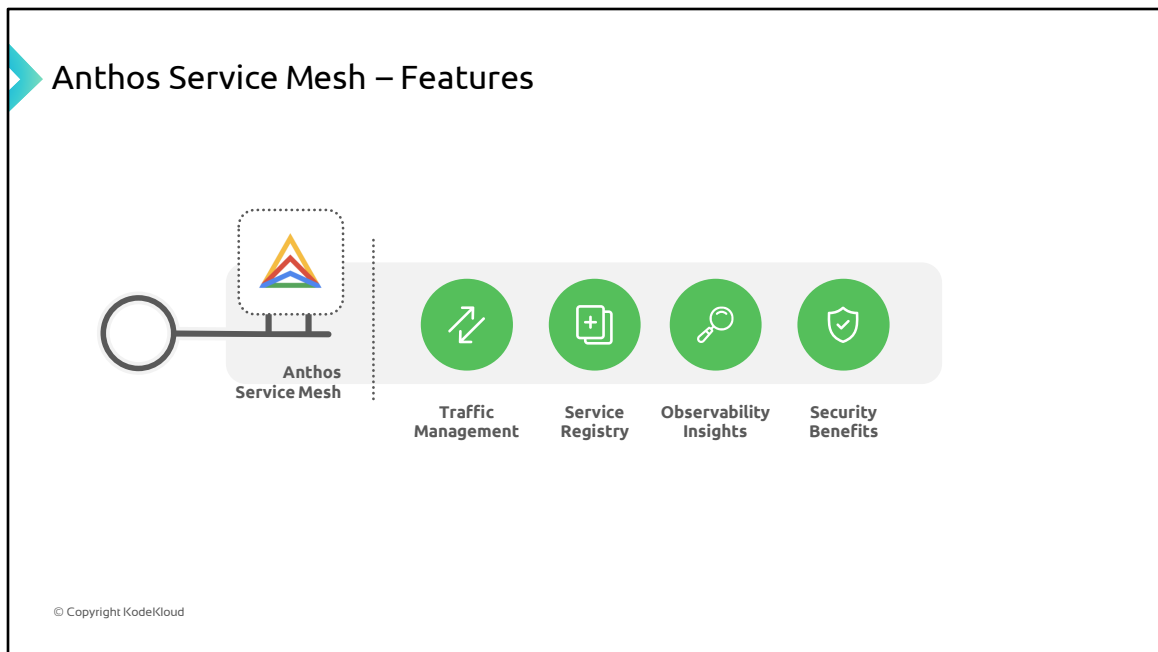
As organisations accelerate their moves to the cloud, by modernising their applications, the shift from legacy applications to cloud-native can raise challenges for DevOps teams such as managing the new cloud-native applications within increasingly large hybrid and multi-cloud environments.



Anthos is a platform developed by Google Cloud that helps with application modernization and management in hybrid and multi-cloud environments.



Anthos provides a range of tools and services to simplify the process of building, deploying, and managing applications across different infrastructures. One key component of Anthos is the Anthos Service Mesh, which is powered by the open-source service mesh platform, Istio.



Some key features of Anthos Service Mesh are:

Traffic Management: Anthos Service Mesh enables you to control the flow of traffic between services within the mesh, as well as inbound (ingress) and outbound (egress) traffic.

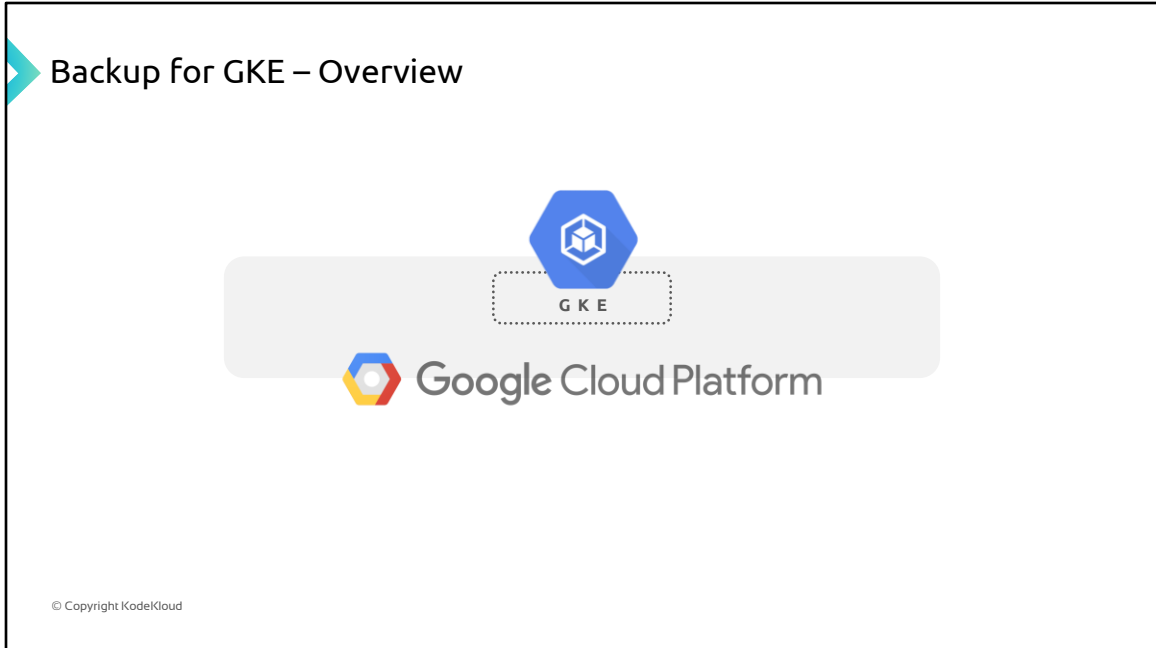
Service Registry: Anthos Service Mesh maintains a service registry that keeps track of all services within the mesh, including their names and endpoints. This registry allows the mesh to direct traffic to the appropriate endpoints based on the defined routing rules.

Observability Insights: Anthos Service Mesh provides observability features to gain insights into your service mesh. The service mesh pages in the Google Cloud console offer service metrics, logs, and preconfigured dashboards for HTTP traffic within your mesh's GKE cluster.

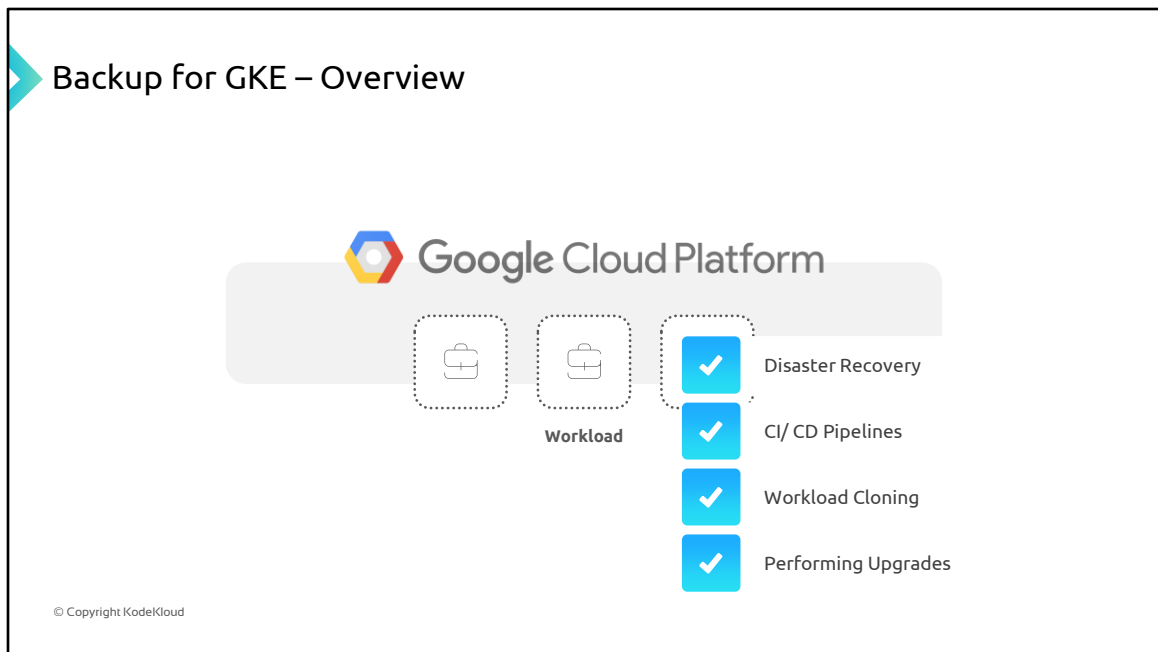
Security Benefits: Anthos Service Mesh uses mutual TLS (mTLS) certificates for peer authentication, reducing the risk of replay or impersonation attacks.

Backup for a GKE Cluster

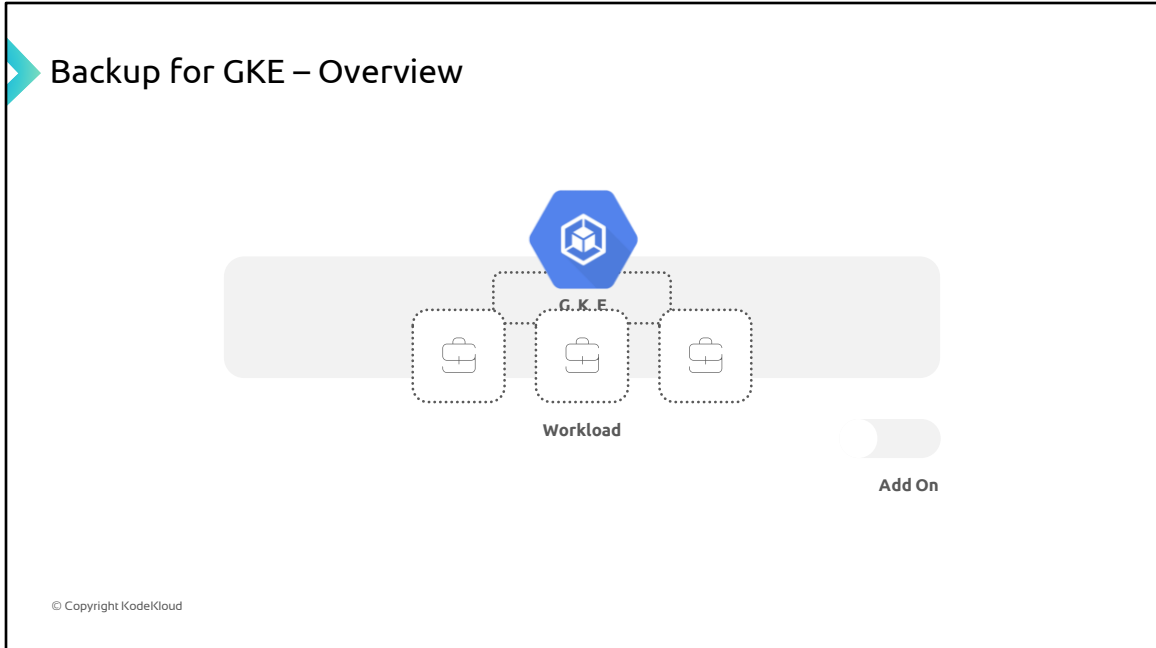
© Copyright KodeKloud



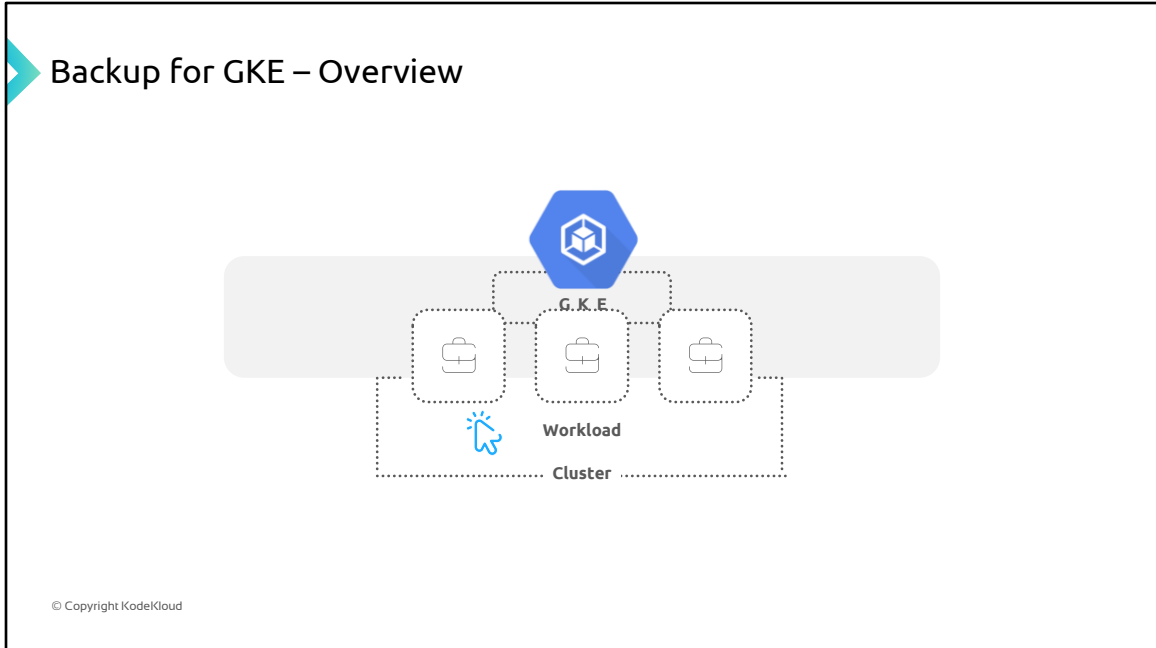
Backup for GKE is a service provided by Google Cloud that allows you to back up and restore workloads running in GKE clusters.



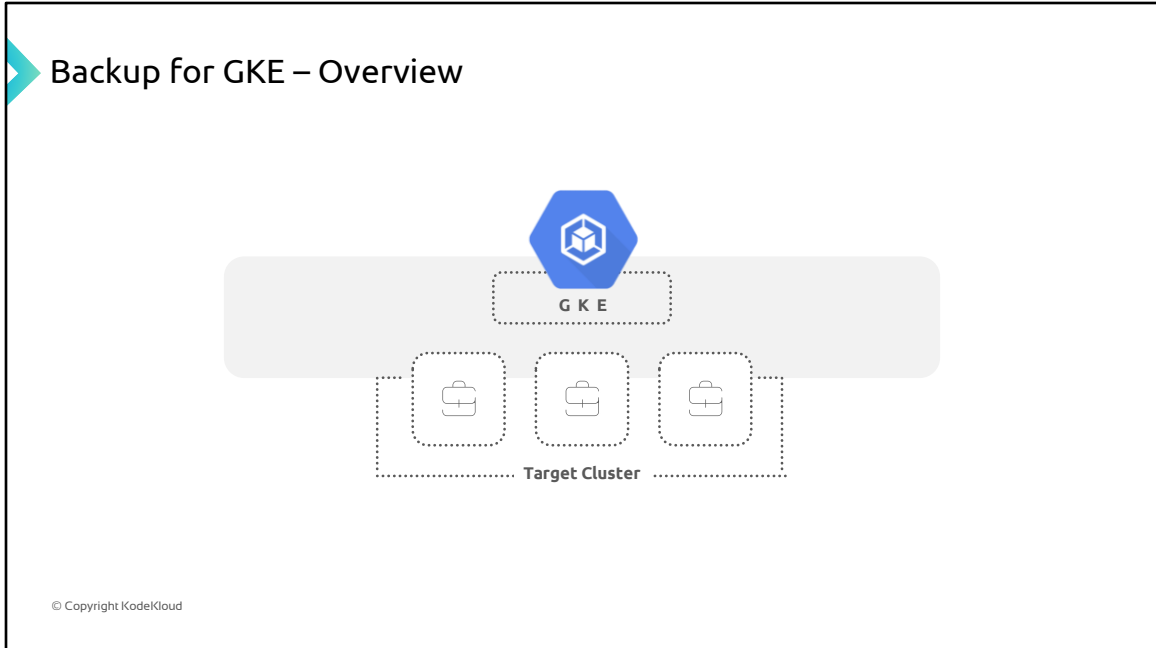
The purpose of backup is to create copies of your workloads, which can be useful for various scenarios such as **disaster recovery**, **CI/CD pipelines**, **workload cloning**, or **performing upgrades**. By having backups, you can ensure that you have recovery options in case of any issues or data loss helping you to meet recovery point objectives.



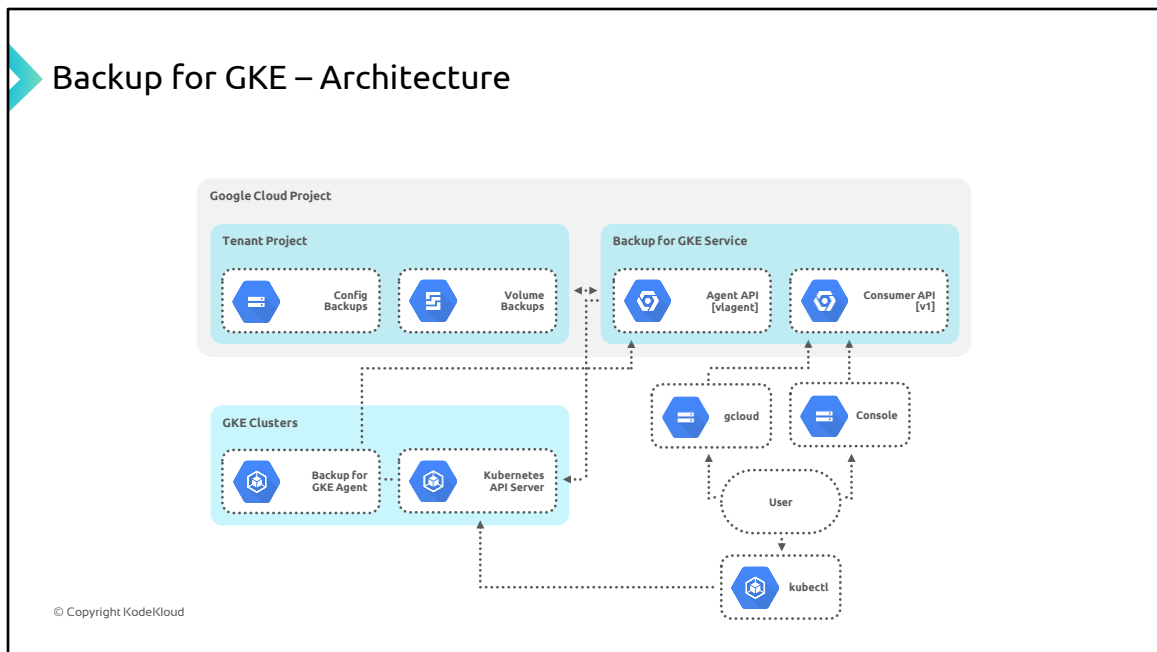
To use Backup for GKE, you first need to enable the Backup for GKE add-on in your GKE clusters. Once the add-on is enabled, you can start creating backups.



To create a backup, you need to specify the workloads that you want to back up. You can choose to back up all workloads in a cluster, or you can select specific workloads.



Once you have created a backup, you can restore it to a target cluster. The target cluster must have the Backup for GKE service enabled.

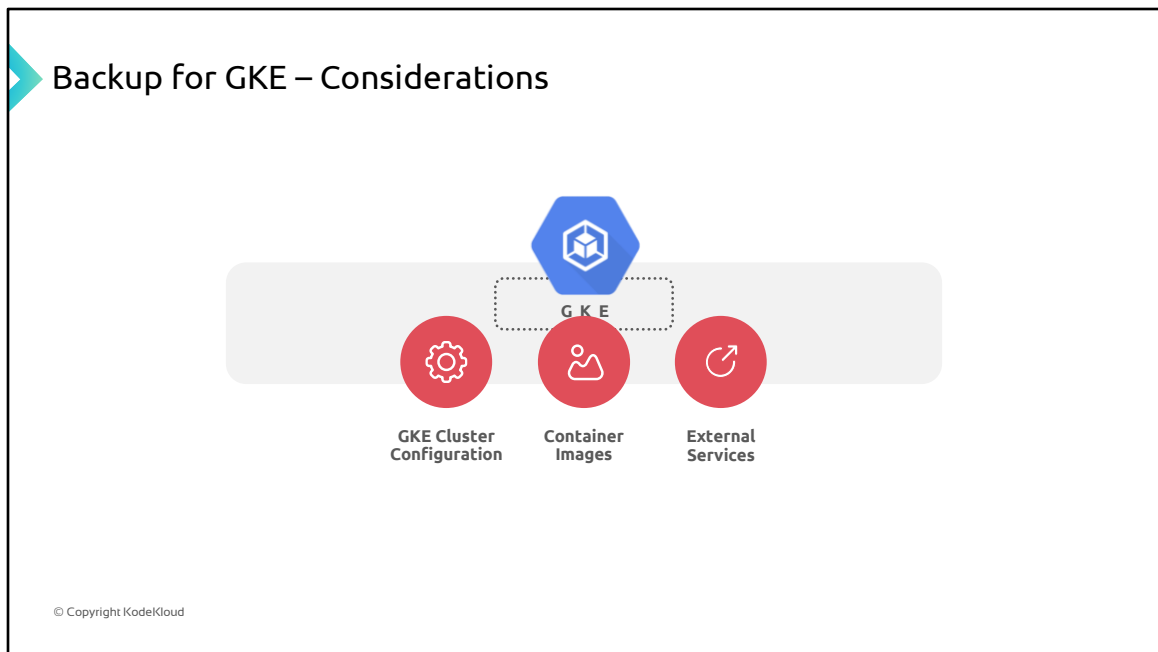


The architecture of Backup for GKE consists of two main components:

Backup for GKE API: The service, which serves as the control plane. It runs in Google Cloud and supports a resource-based REST API. It provides an API endpoint that clients can interact with to manage backup and restore operations.

Backup for GKE agent which is an add-on that runs within each GKE cluster that is configured to use backup for GKE. Its purpose is to execute the backup and restore activities within the cluster. The agent is responsible for orchestrating the backup

and restore processes, interacting with the Backup for GKE API to perform the necessary operations such as orchestrating the processes, fetching resources from the Kubernetes API server, serialising them into an archive, storing the archive, creating backups of persistent volumes, and handling restore operations.



Backup for GKE focuses on backing up Kubernetes resources and the underlying persistent volumes, but there are certain aspects that it does not back up:

GKE Cluster Configuration: Backup for GKE does not back up GKE cluster configuration information, such as node configuration, node pools, initial cluster size, or enabled features. These aspects of the cluster need to be managed separately.

Container Images: Only the Kubernetes resources that describe the workload and reference container images are backed up. Backup for GKE does not

back up the actual container images themselves. If an image referenced by a workload manifest in a backup is removed from its image repository, a subsequent restore of that configuration will not be able to restore the workload successfully. It's important to ensure that the required container images are available when performing a restore.

External Services: Backup for GKE does not back up configuration information or state of services that exist outside the GKE cluster. This includes services like Cloud SQL databases or external load balancers. If you have dependencies on external services, their configuration and state need to be managed separately.

It's crucial to have a comprehensive backup and recovery strategy that covers all the necessary components and services, including the cluster configuration, container images, and any external dependencies.