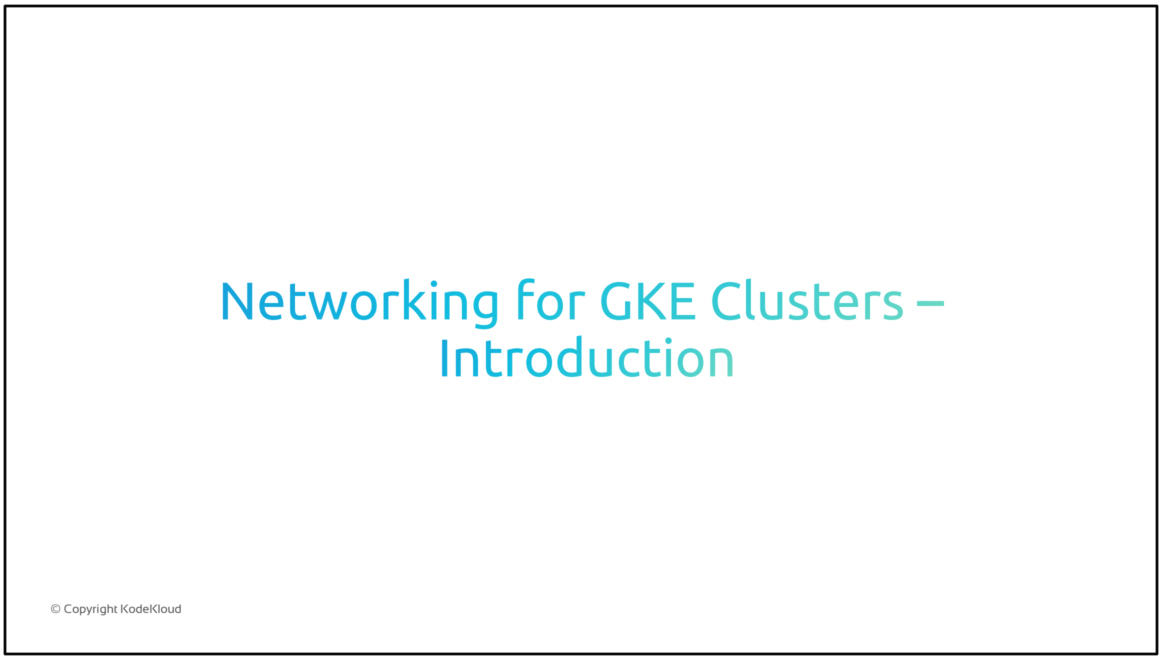
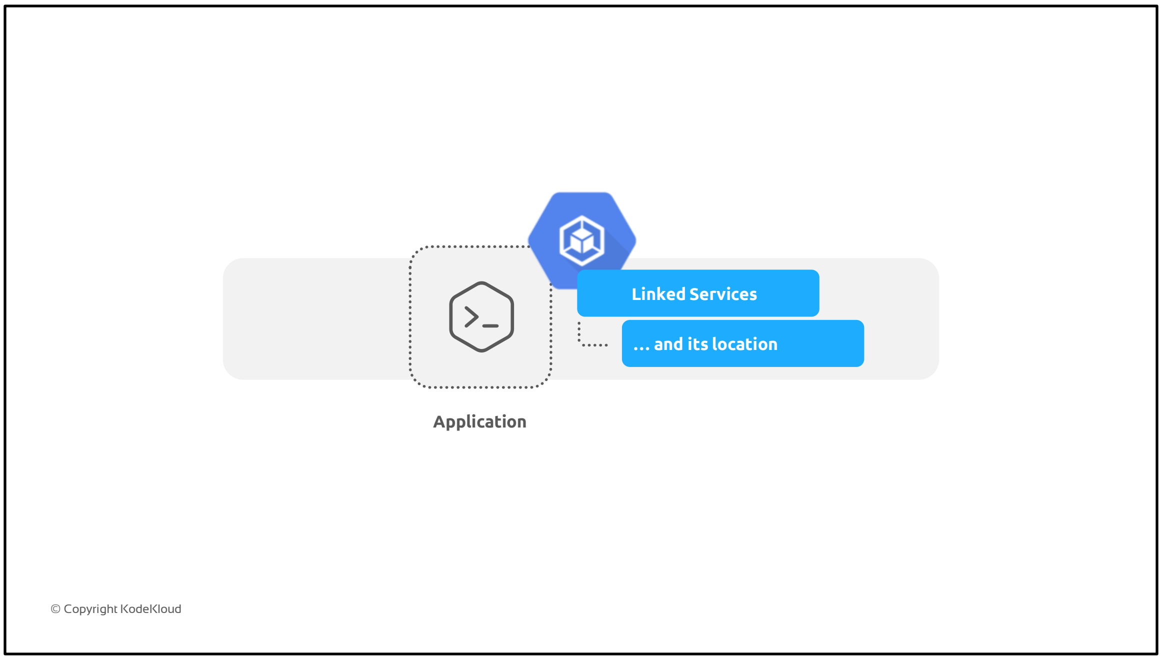
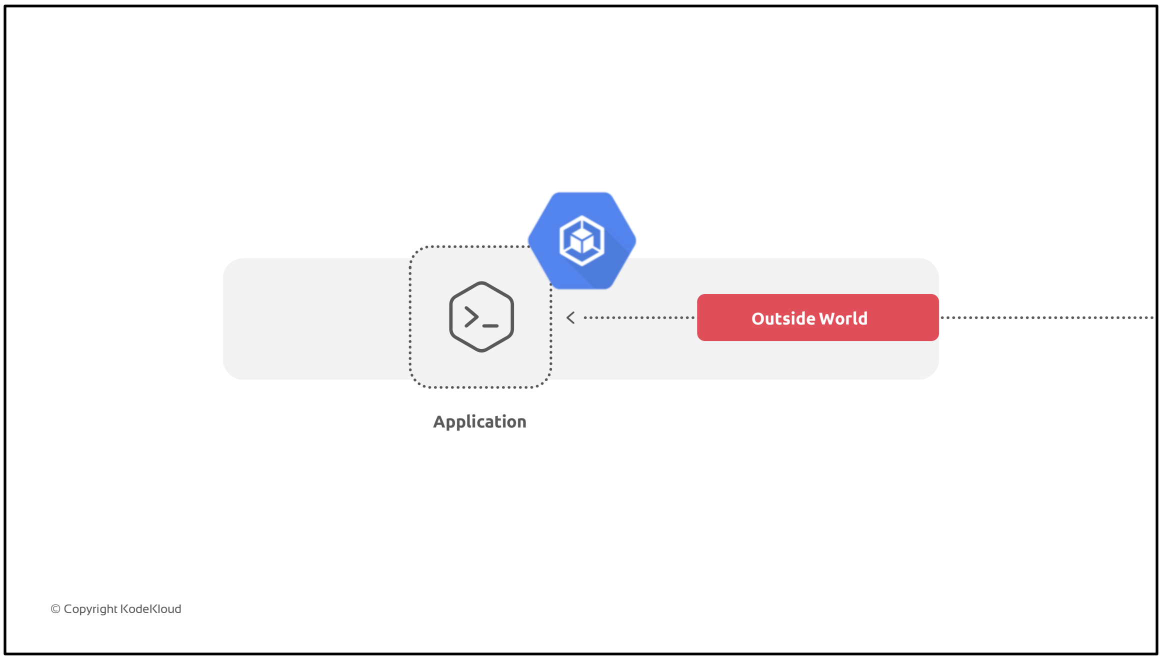


Networking for GKE Clusters – Introduction

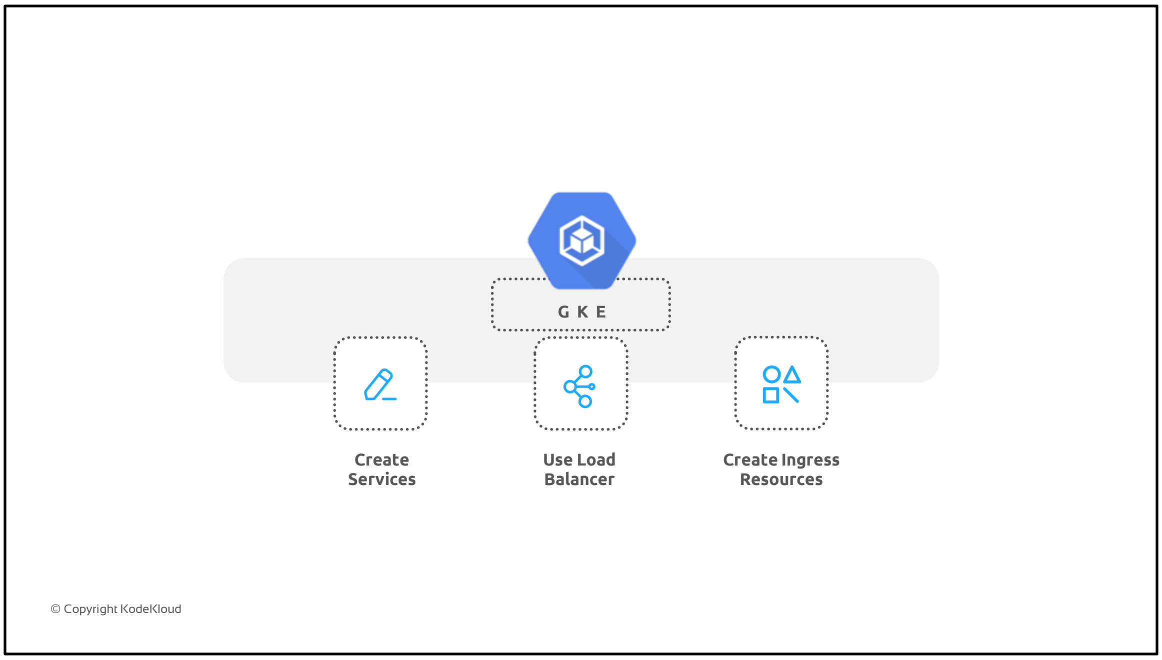
Welcome to the Google Kubernetes Engine Networking module.



When designing your applications, you need to understand what other types of services it will talk to and where they may be located.



You also need to consider how to expose your applications to the outside world for consumption and how to balance incoming traffic to cope with varying workload demands.
That's the theme for this module.

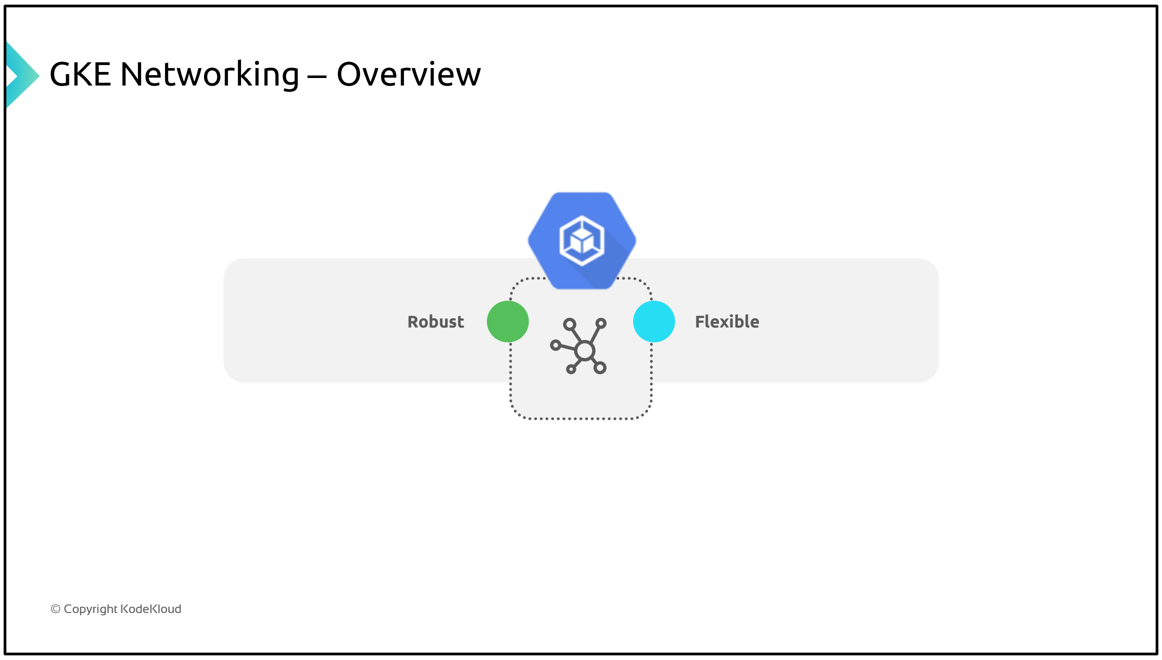


In this module, you will learn how to **create services** to expose applications running within a Pod, allowing them to communicate with each other and with the outside world.
Use **load balancers** to expose services to external clients, and spread incoming traffic across your cluster as evenly as possible.

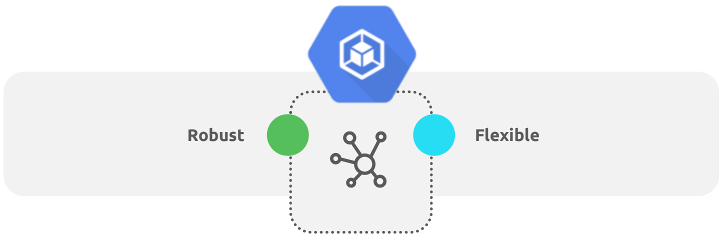
Create **ingress resources, which** defines rules for distributing incoming traffic to applications running in a cluster.
We'll also look at how leveraging container-native

load balancing with ingress improves network performance.

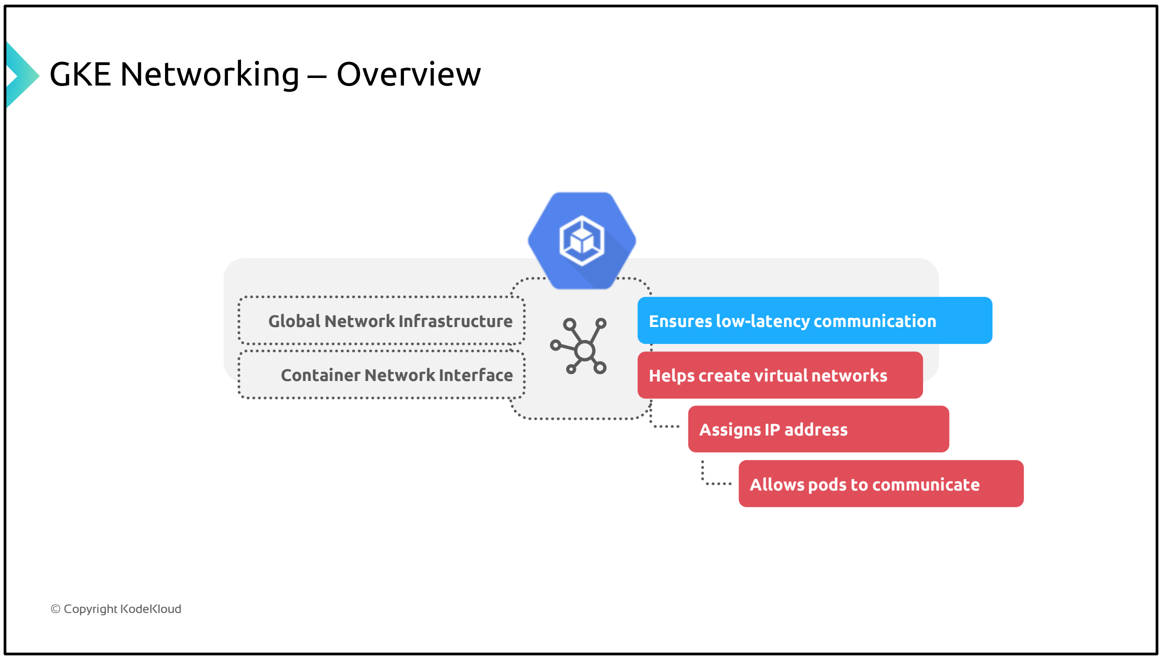
GKE Networking – Overview



GKE Networking – Overview



Google Kubernetes Engine (GKE) provides a robust and flexible networking infrastructure that allows for seamless communication and connectivity for containerized applications within Kubernetes clusters.

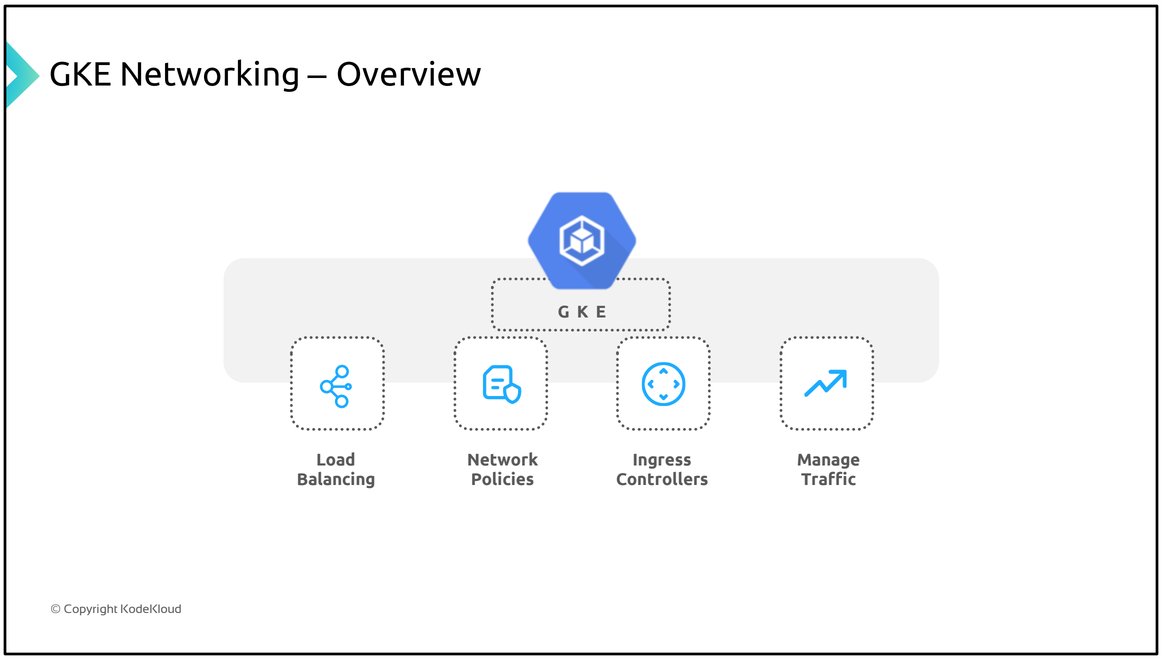


GKE Networking – Overview

Global Network Infrastructure
Container Network Interface
Ensures low-latency communication
Helps create virtual networks
Assigns IP address
Allows pods to communicate

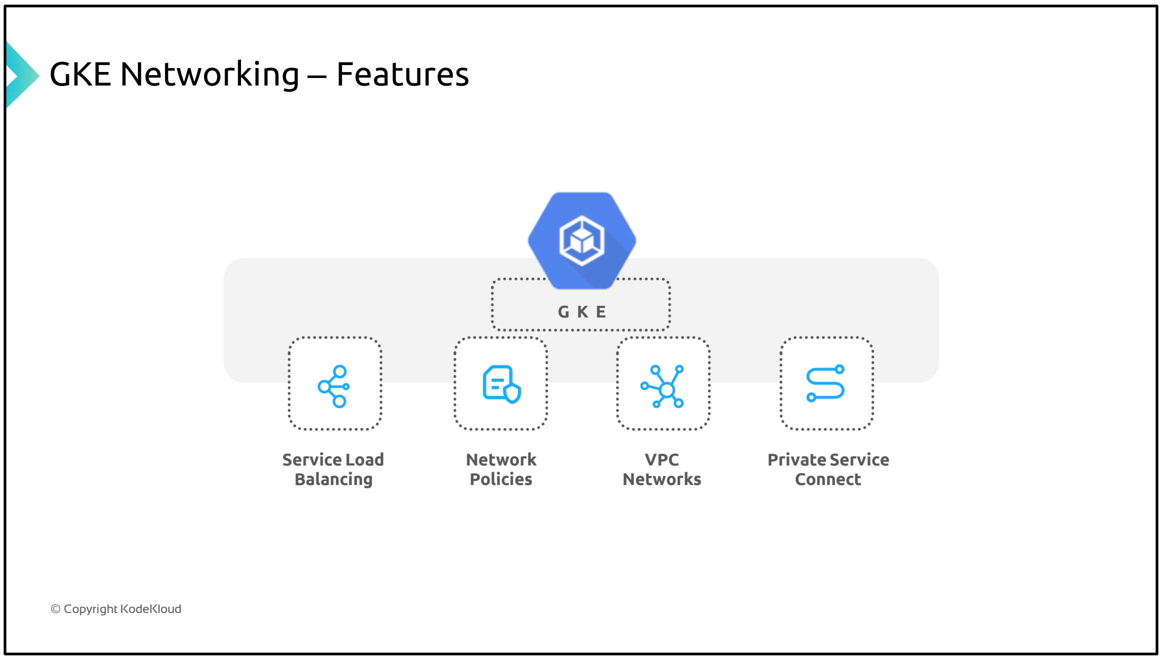
It leverages **Google Cloud's global network infrastructure**, which ensures **low-latency communication** between your cluster's nodes and other Google Cloud services.

GKE utilizes the **Container Network Interface (CNI) plugin**, which enables the **creation of virtual networks and the assignment of IP** addresses to pods. This allows **pods to communicate** with each other, both within the cluster and across different clusters.



GKE Networking – Overview

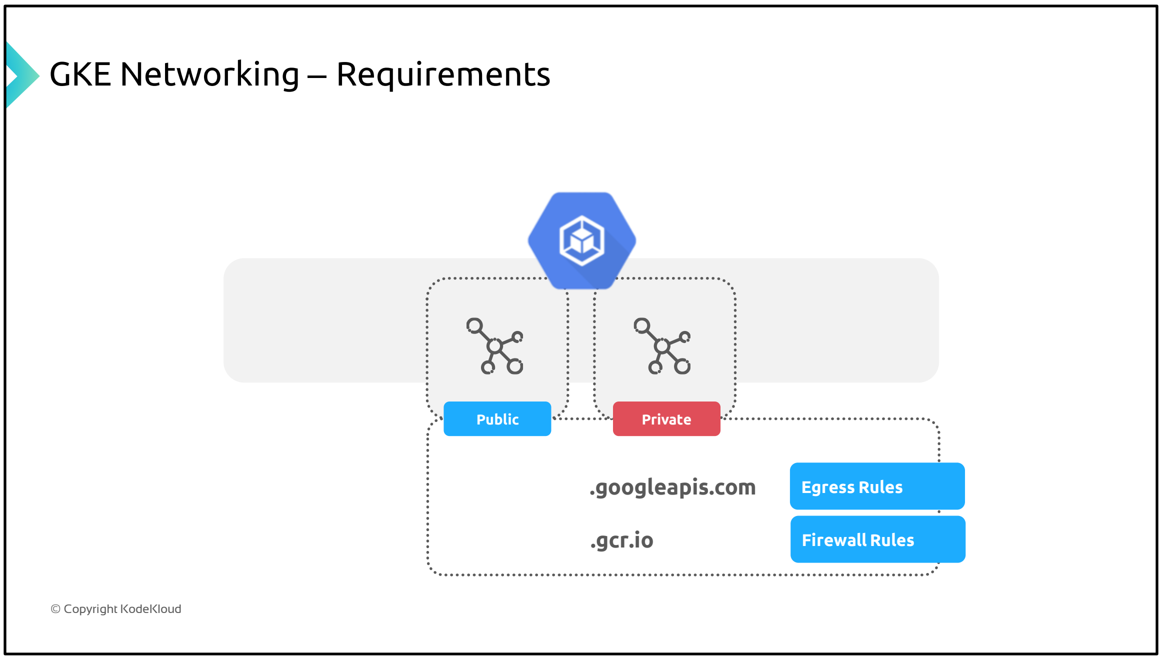
GKE also provides features such as **load balancing**, **network policies**, and **ingress controllers**, which enable us to easily expose and **manage your applications'** network traffic.



GKE Networking – Features

Some GKE's networking features include:
• Service load balancing: Using Service Load Balancing, GKE can automatically create and manage load balancers for your services, both internal and external.
• Network policies: By configuring network policies in GKE, you can control how pods communicate with each other and with the outside world to improve security and performance.
• VPC networks: GKE clusters are deployed in GCP VPC networks, which provides isolation and security for your applications.

• Private Service Connect: GKE can also be used to connect your clusters to private Google Cloud services, such as BigQuery and Cloud Storage. This allows you to keep your data secure while still taking advantage of the scalability and flexibility of GKE.

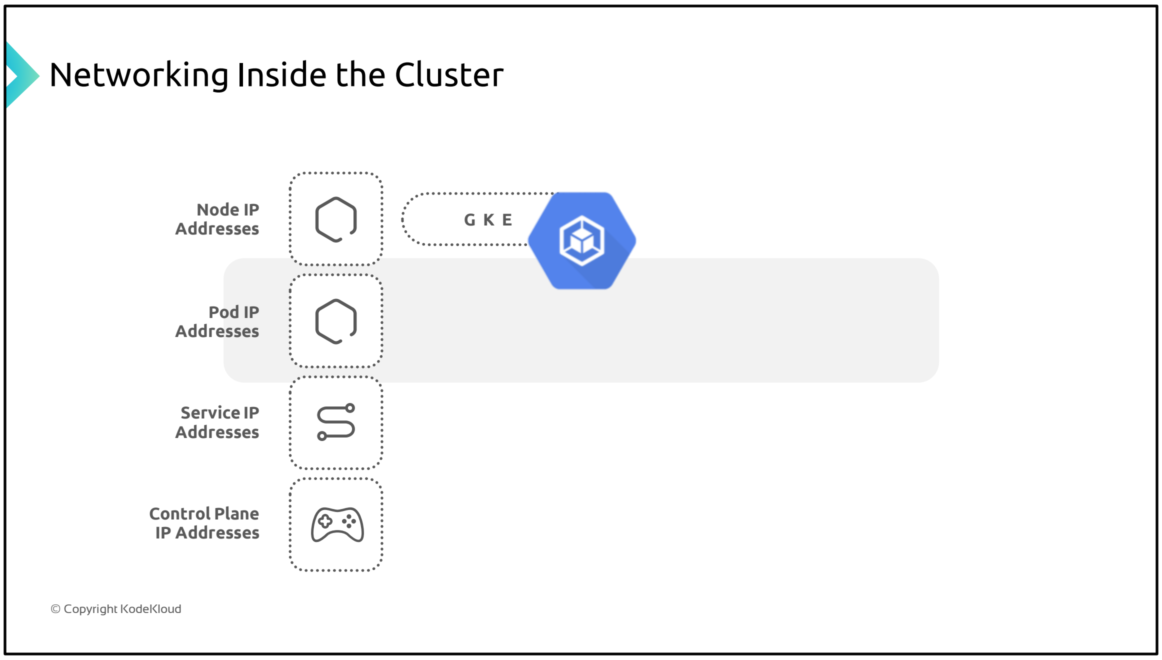


GKE Networking – Requirements

Based on how IP addresses are allocated inside a VPC, GKE clusters can either be created as public or private clusters. Both cluster types, however, require connectivity to following endpoints and IP address ranges:
*.googleapis.com
•*.gcr.io
•and the control plane IP address.

This requirement is satisfied by the implied allow egress rules and the automatically created firewall rules that GKE creates.

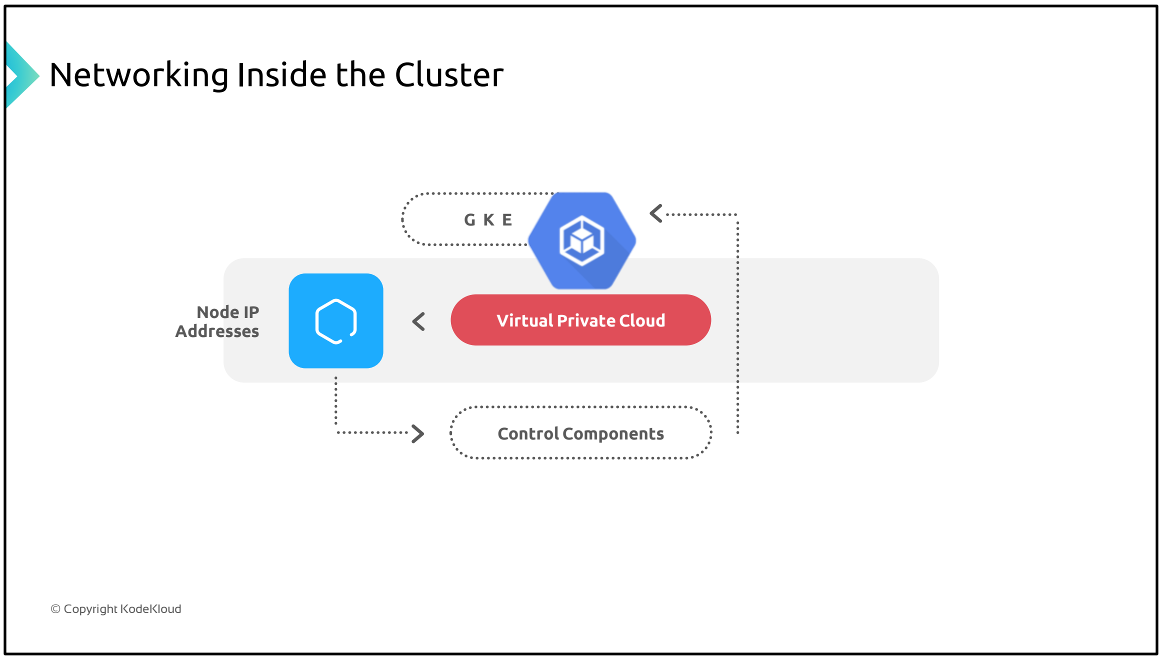
For public clusters, if you add firewall rules that deny egress with a higher priority, you must create firewall rules to allow *.googleapis.com, *.gcr.io, and the control plane IP address.



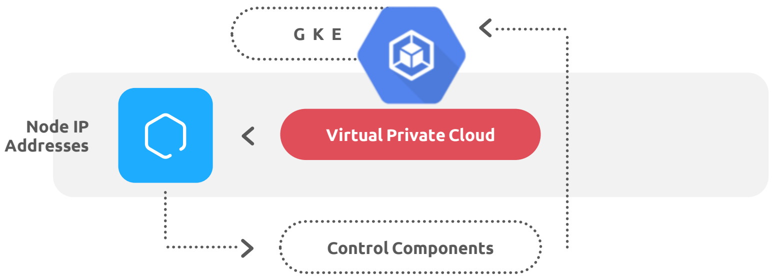
Networking Inside the Cluster

In a Google Kubernetes Engine (GKE) cluster, IP address allocation plays a crucial role in establishing connectivity between various components. The allocation of IP addresses in a GKE cluster follows a specific pattern:

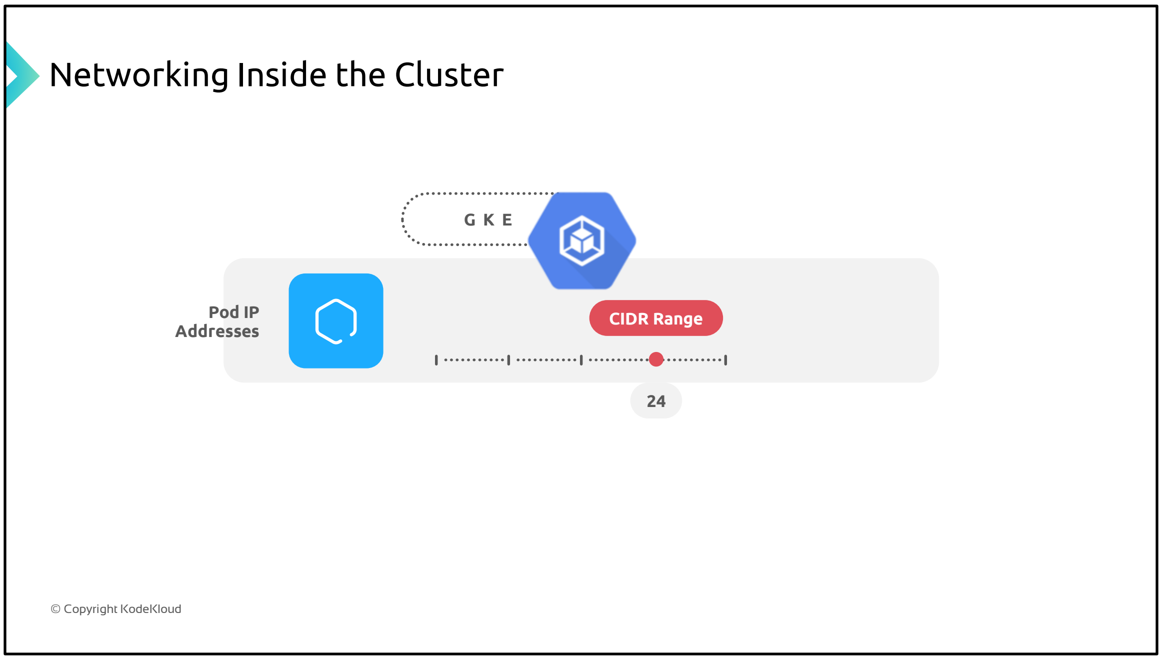
1. Node IP addresses:
2. Pod IP addresses:
3. Service IP addresses:
4. Control plane IP addresses:



Networking Inside the Cluster

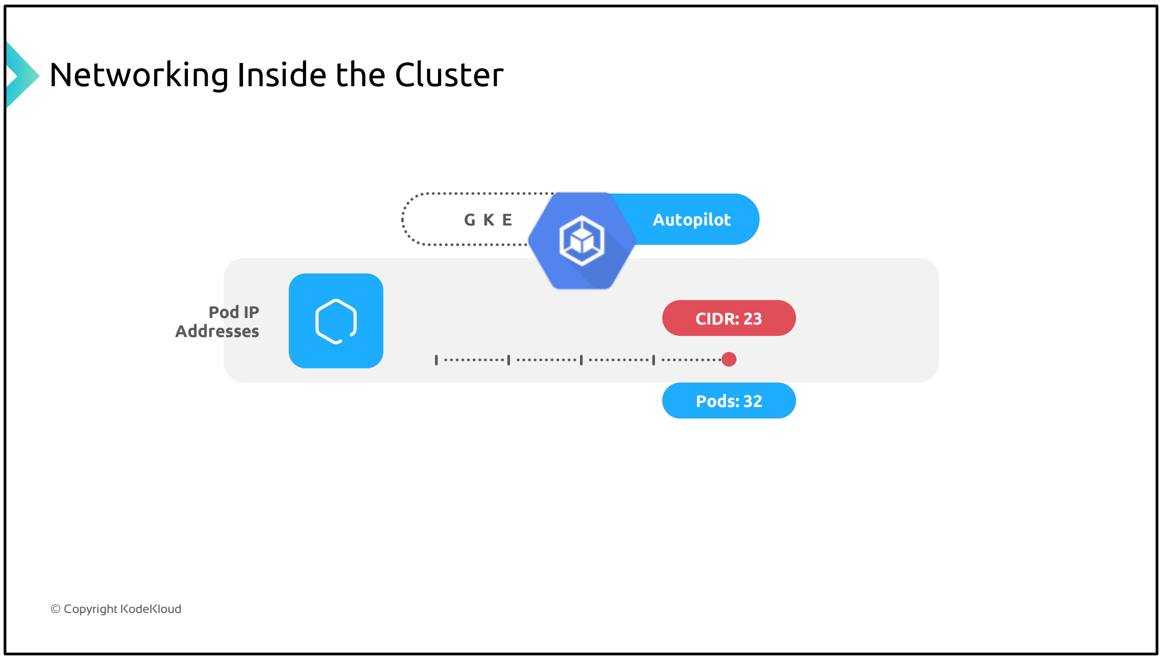


1: Node IP addresses: Each node in the cluster is assigned an IP address from the cluster's Virtual Private Cloud (VPC) network. This IP address allows the node to connect with control components like kube-proxy and the kubelet, facilitating communication with the Kubernetes API server. The node's IP address serves as its connection to the rest of the cluster.



Networking Inside the Cluster

2: Pod IP addresses: Each node in the cluster has a pool of IP addresses reserved for assigning to Pods running on that node. By default, GKE assigns a /24 CIDR block as the Pod IP range for each node. However, you can customize the range when creating the cluster using the Flexible Pod CIDR range feature. This flexibility allows you to adjust the size of the Pod IP range for nodes in a specific node pool.

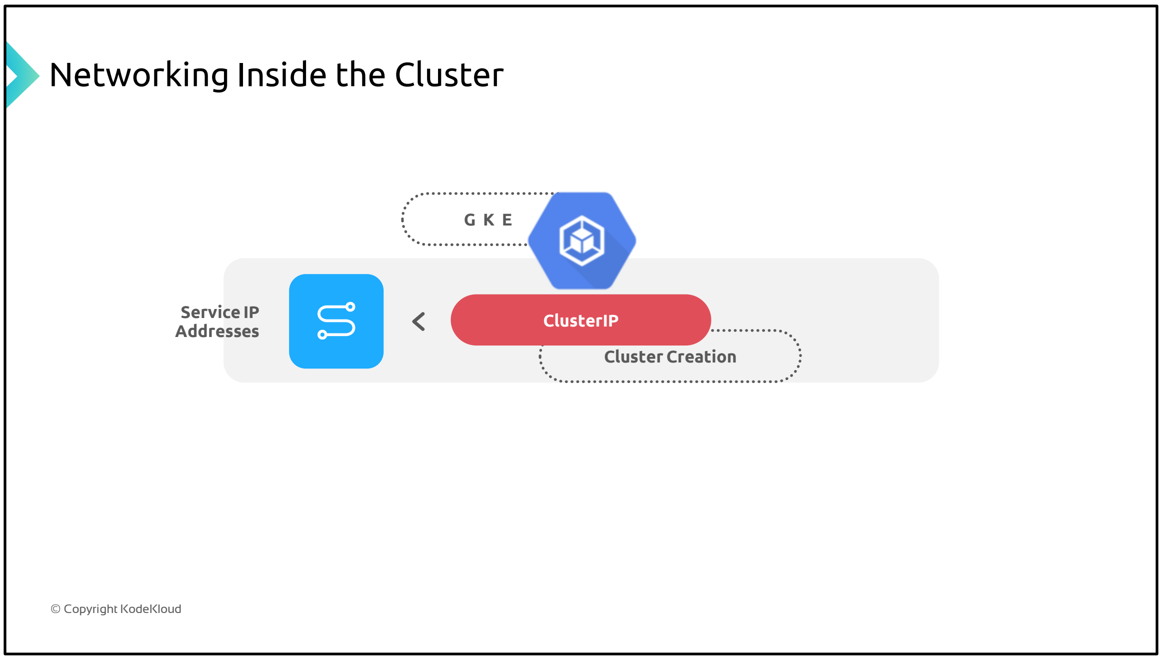


Networking Inside the Cluster

Note: It's important to note that for Standard clusters, the maximum number of Pods that can be scheduled on a node is determined by the size of the Pod IP range. For example, with a /23 range, a maximum of 256 Pods can be scheduled on a node, not 512 as one might expect. This buffer ensures that Pods don't become unschedulable due to temporary shortages of available IP addresses within the Pod IP range.

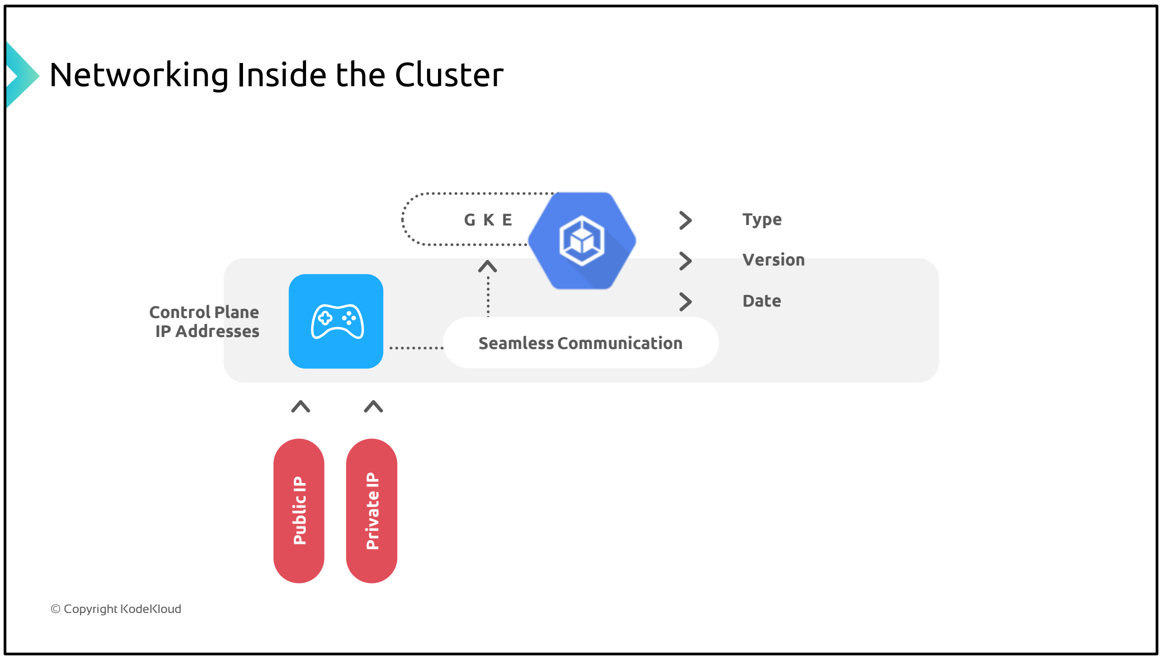
Autopilot clusters, on the other hand, allow a maximum of 32 Pods per node.

Each Pod running within the cluster is assigned a single IP address from the Pod CIDR range of its node. This IP address is shared among all the containers running within the Pod and enables communication between Pods within the cluster. It acts as the identifier for the Pod and helps establish connectivity with other Pods.



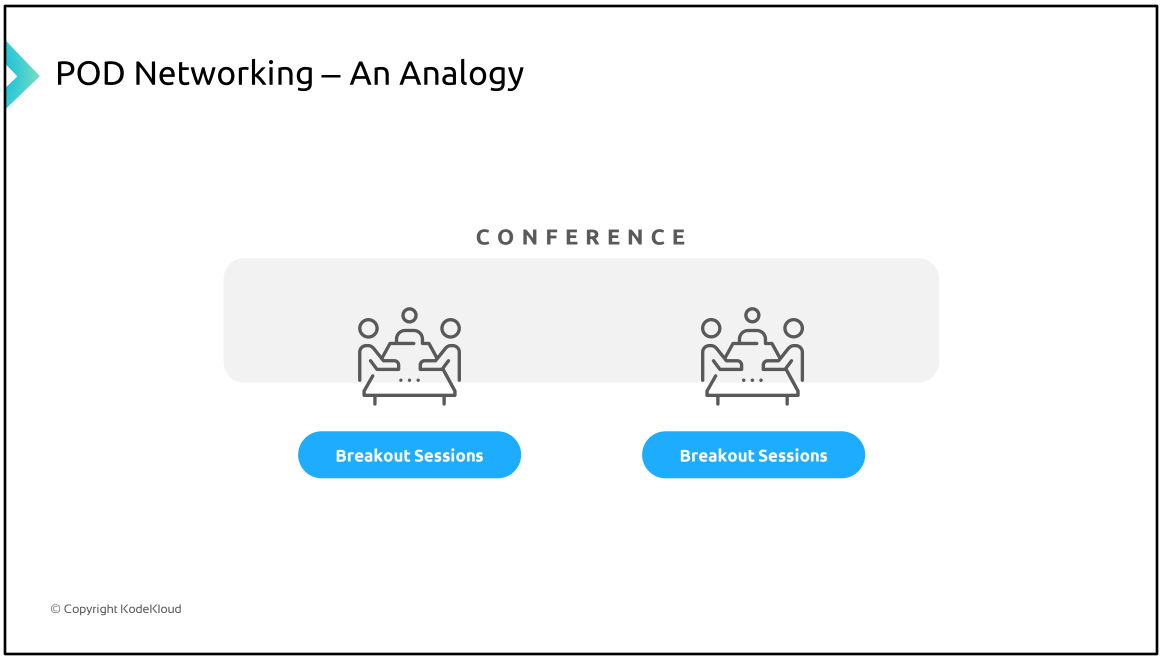
Networking Inside the Cluster

3. Service IP addresses: Services in a GKE cluster are assigned an IP address known as the ClusterIP. This IP address is obtained from the cluster's VPC network. Each Service is assigned a unique ClusterIP, which allows for communication with other components within the cluster. Customization of the VPC network can be done during the cluster creation process.


Networking Inside the Cluster

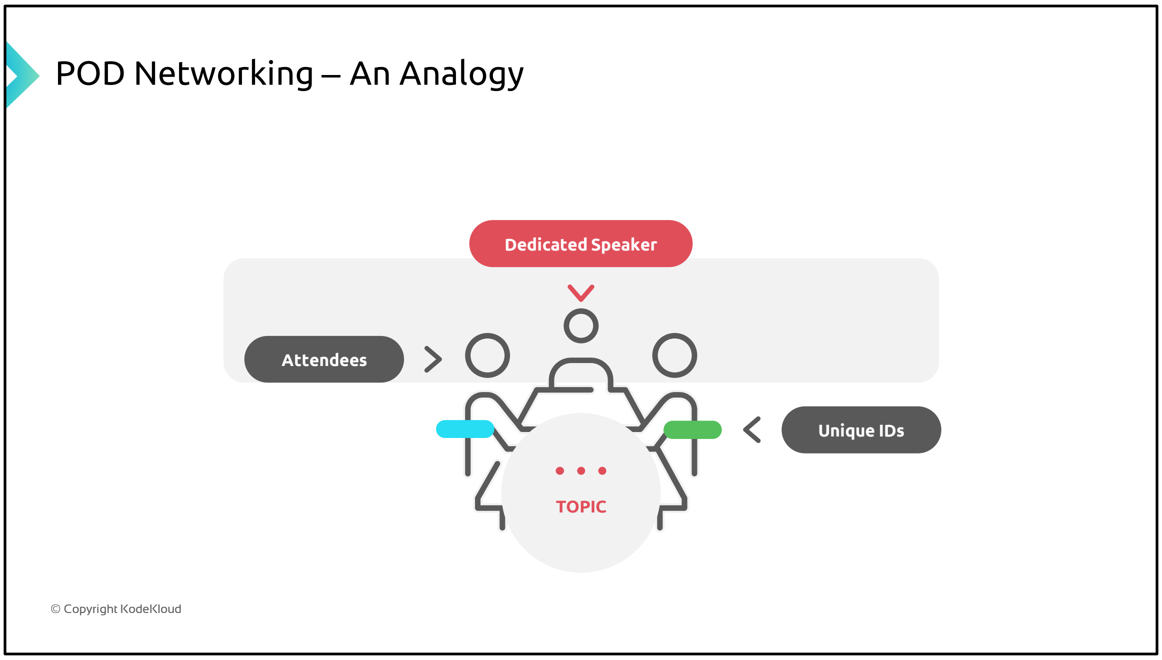
4: Control plane IP addresses: The control plane of a GKE cluster is associated with either a public or private IP address, depending on the cluster's type, version, and creation date. These IP addresses enable the control plane to manage and control the cluster's operations.

By effectively allocating IP addresses to nodes, Pods, and Services, GKE ensures seamless communication and connectivity within the cluster, facilitating the smooth operation of applications and services deployed on Kubernetes.



POD Networking – An Analogy

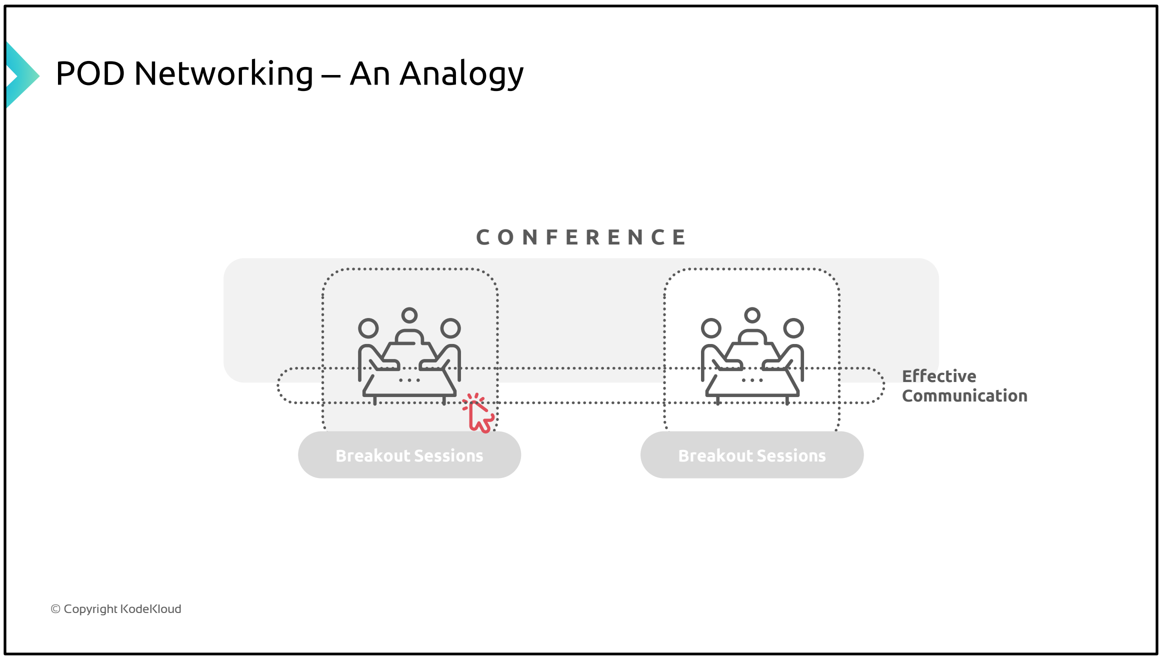
Imagine you are attending a conference with multiple breakout sessions.



POD Networking – An Analogy

Each session focuses on a specific topic and has a dedicated speaker who presents valuable information to the attendees.

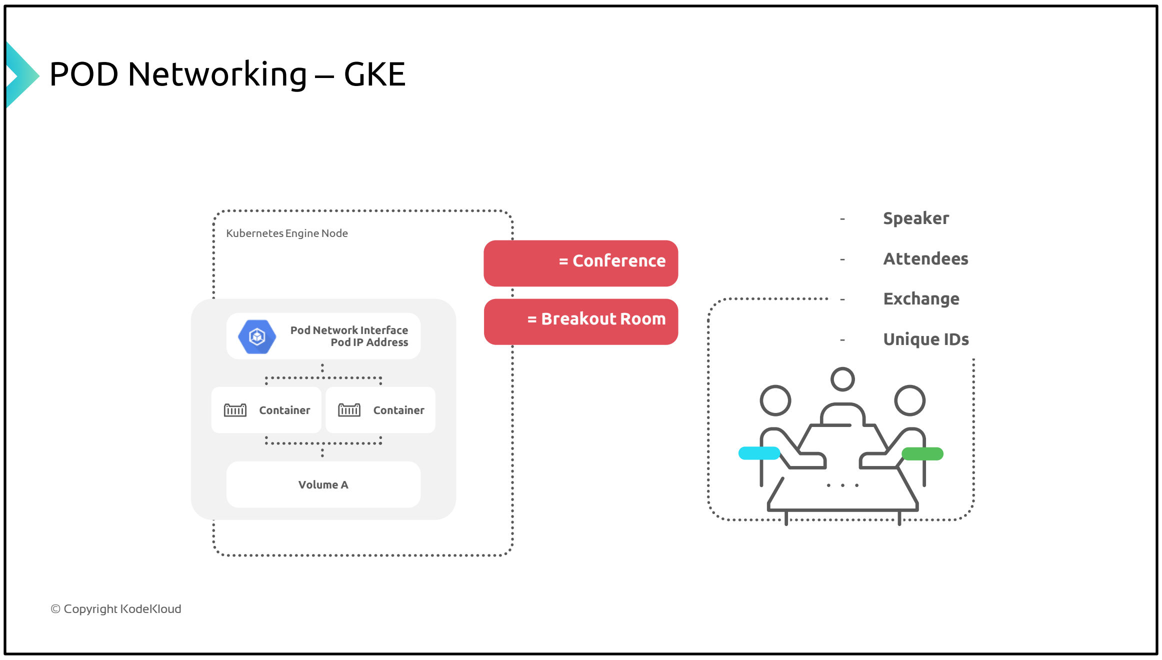
Before the sessions start, the conference organizers assign each attendee and the speakers a unique ID, which serves as their identification within the conference venue. The purpose of assigning an ID is to keep track of each attendee or speaker and ensure they are in the right session. This ID helps identify and locate them within the conference venue, making it easier for them to navigate and find their respective breakout sessions.



POD Networking – An Analogy

During the conference, multiple breakout sessions run simultaneously, allowing attendees to choose which session they want to participate in based on their interests and preferences. Each attendee belongs to a specific group based on the breakout session they decide to attend at that particular time.

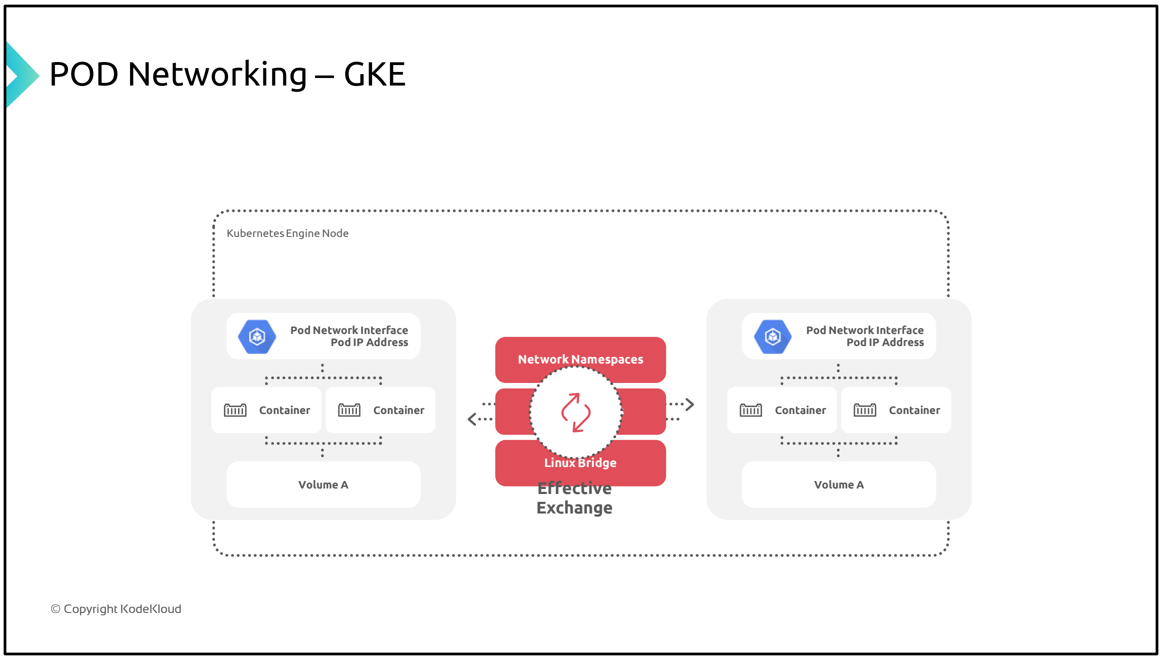
To ensure effective communication between the speakers and attendees, the conference venue designates specific areas or rooms for each breakout session. These designated areas provide a suitable environment where the speakers can present their information and engage with the attendees. It allows attendees to gather and interact with the speaker, ask questions, and discuss the topic at hand.



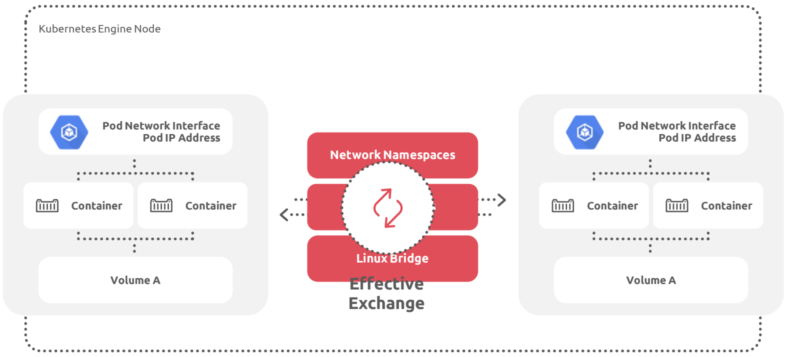
POD Networking – GKE

The networking for Pods operates within the cluster's infrastructure, providing connectivity for containers and facilitating communication between them.

A Pod represents the fundamental unit of deployment in Kubernetes and can run one or more containers. Within a GKE cluster, Pods run on nodes, which are virtual machines running as instances in Compute Engine. Multiple Pods can run on a single node, and each node belongs to a node pool.

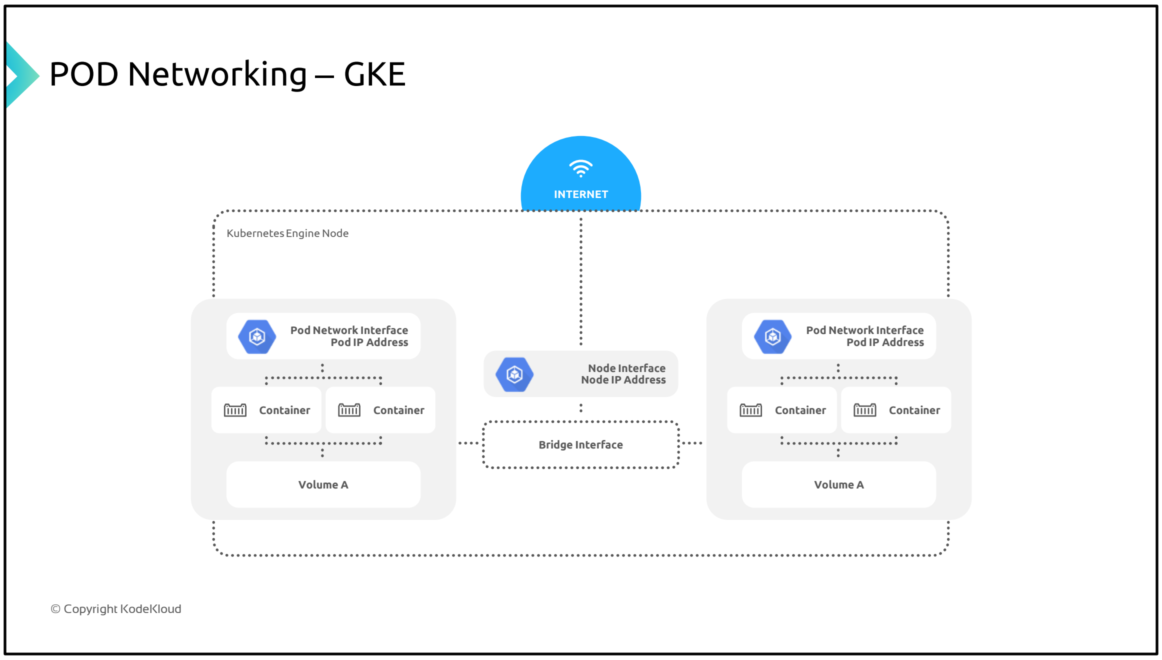


POD Networking – GKE



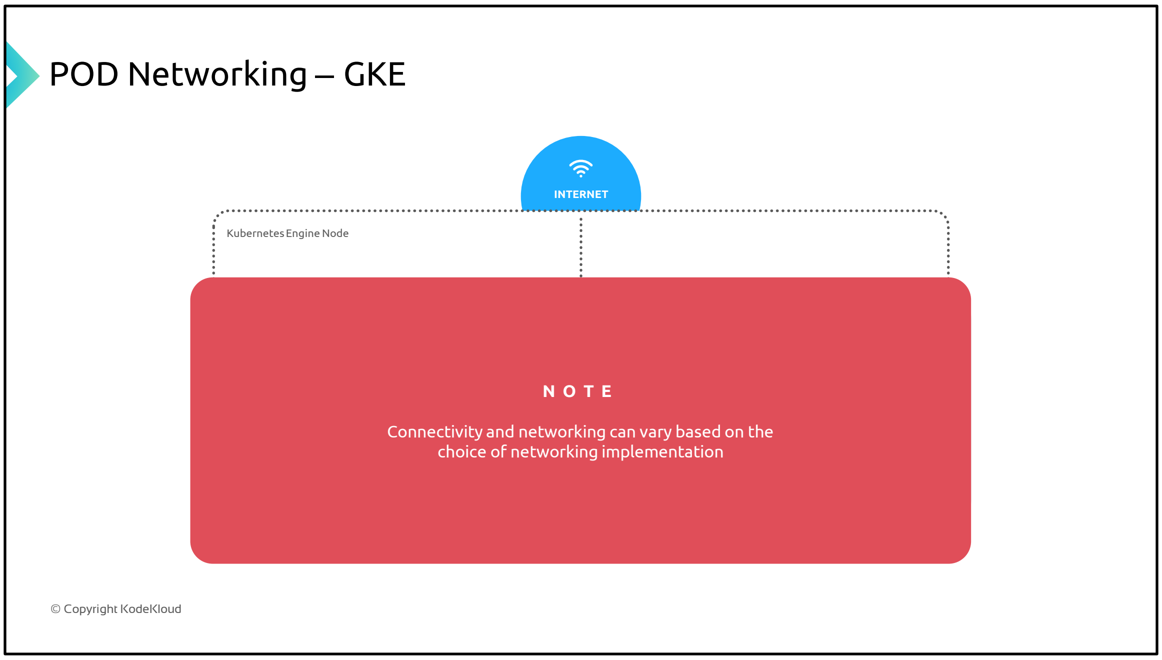
GKE enables seamless communication and networking between Pods within a cluster by establishing
- network namespaces and
- Pod IP addresses and by
- utilizing Linux bridges.

This infrastructure ensures that containers within a Pod can communicate effectively, and Pods can exchange data and interact with each other as required by the application architecture.



POD Networking – GKE

When Kubernetes schedules a Pod to run on a node, it creates a network namespace for the Pod within the node's Linux kernel. This network namespace establishes a connection between the node's physical network interface (e.g., eth0) and the Pod using a virtual network interface. This connection allows packets to flow to and from the Pod



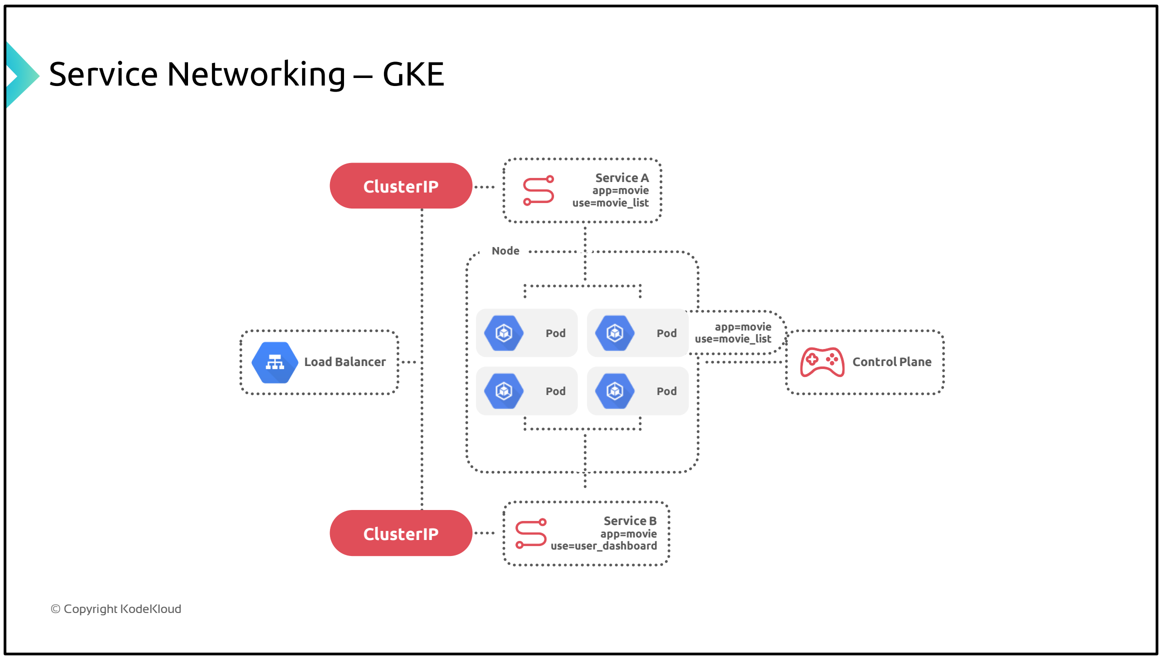
POD Networking – GKE

INTERNET
Kubernetes Engine Node

NOTE

Connectivity and networking can vary based on the choice of networking implementation

An important note here: The connectivity and networking capabilities within the cluster can vary based on the choice of networking implementation. GKE offers a native Container Network Interface (CNI) for Pod networking.



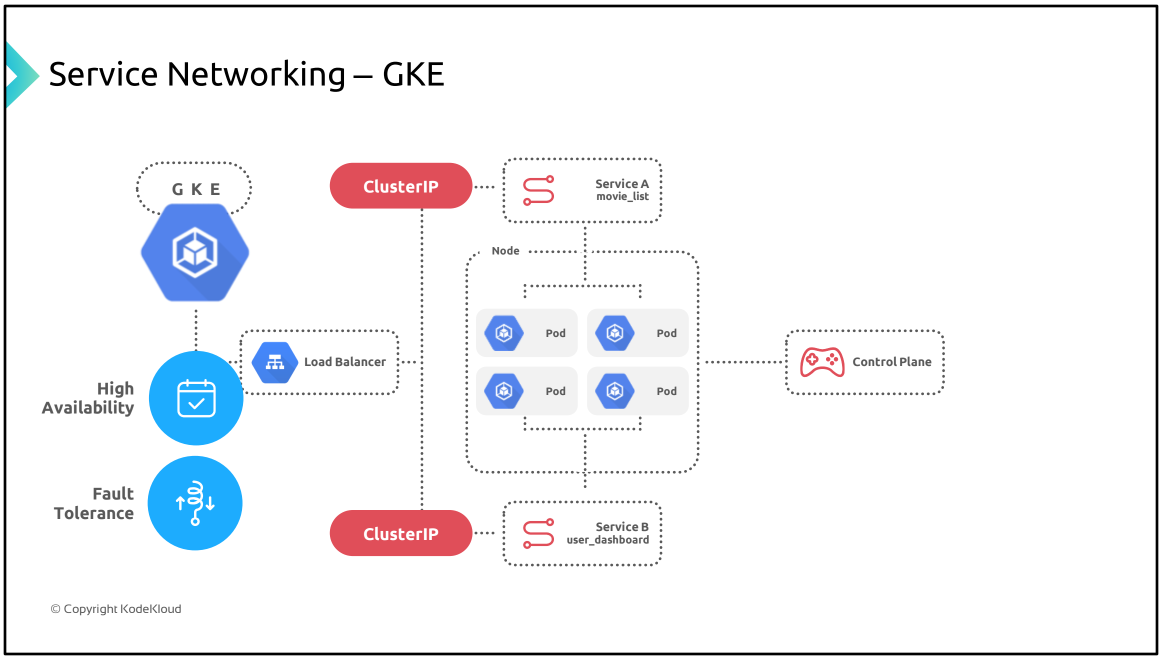
Service Networking – GKE

In Kubernetes, labels are used to group related Pods into logical units called Services. These labels are arbitrary key-value pairs that can be assigned to any Kubernetes resource. When creating a Service, you define a label selector that specifies which Pods should be included in the Service based on their labels.

Each Service in Kubernetes is assigned a stable and reliable IP address known as the ClusterIP. This IP address is chosen from the cluster's pool of available Service IP addresses and remains constant throughout the lifecycle of the Service. Additionally, Kubernetes assigns a hostname to the ClusterIP by creating a DNS entry. The ClusterIP and hostname are unique within the cluster, allowing other components and applications to reliably access the Service.

The purpose of a Service is to provide load balancing across the set of Pods that match the labels specified in the label selector. For example, in this diagram, there are two separate Services: "movie_list" and "user_dashboard." for the same application "movie". Each Service consists of multiple Pods that have the label "app=movie," but with differing additional labels "use" as "movie_list" and "user_dashboard." The Services ensure that incoming traffic is distributed evenly across the Pods that meet the label criteria.

It is important to note that a Kubernetes Service is different from a Docker service. In Kubernetes, a Service provides load balancing and routing functionality, while in Docker, a service is more closely related to a Kubernetes Pod.



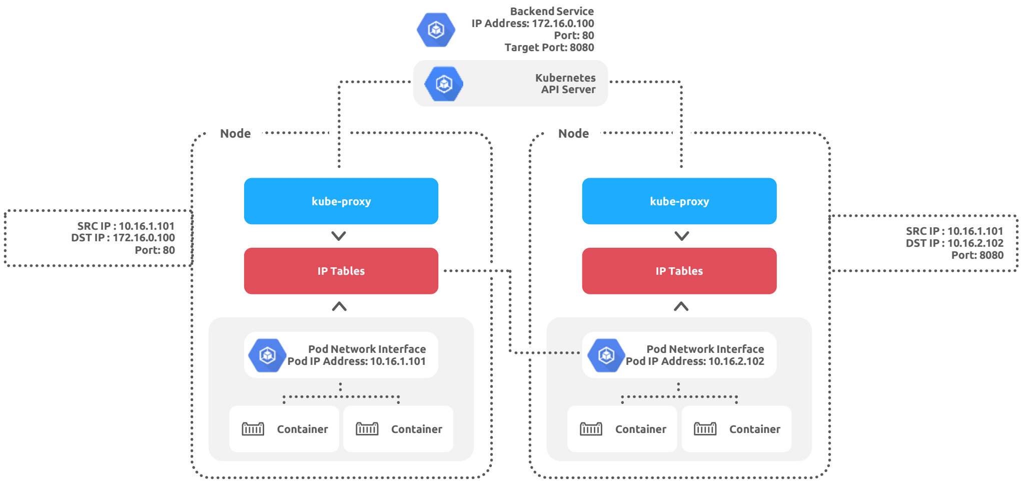
Service Networking – GKE

Kubernetes employs load balancing strategies to evenly distribute traffic among the Pods associated with a Service. This helps ensure **high availability** and **fault tolerance** within the cluster. Even if one or more nodes in the cluster experience an outage, the traffic will be distributed across the remaining healthy Pods on other nodes, allowing the cluster to withstand partial failures.

Accessing a healthy Pod running an application can be achieved using either the ClusterIP or the hostname associated with the Service. Kubernetes guarantees the stability and uniqueness of these addresses, allowing seamless communication with the desired Pods.

By leveraging Service networking in a GKE cluster, applications can benefit from **load balancing, fault tolerance, and reliable communication** among Pods, ultimately enhancing the availability and scalability of the overall system.

Kube-Proxy Networking – GKE



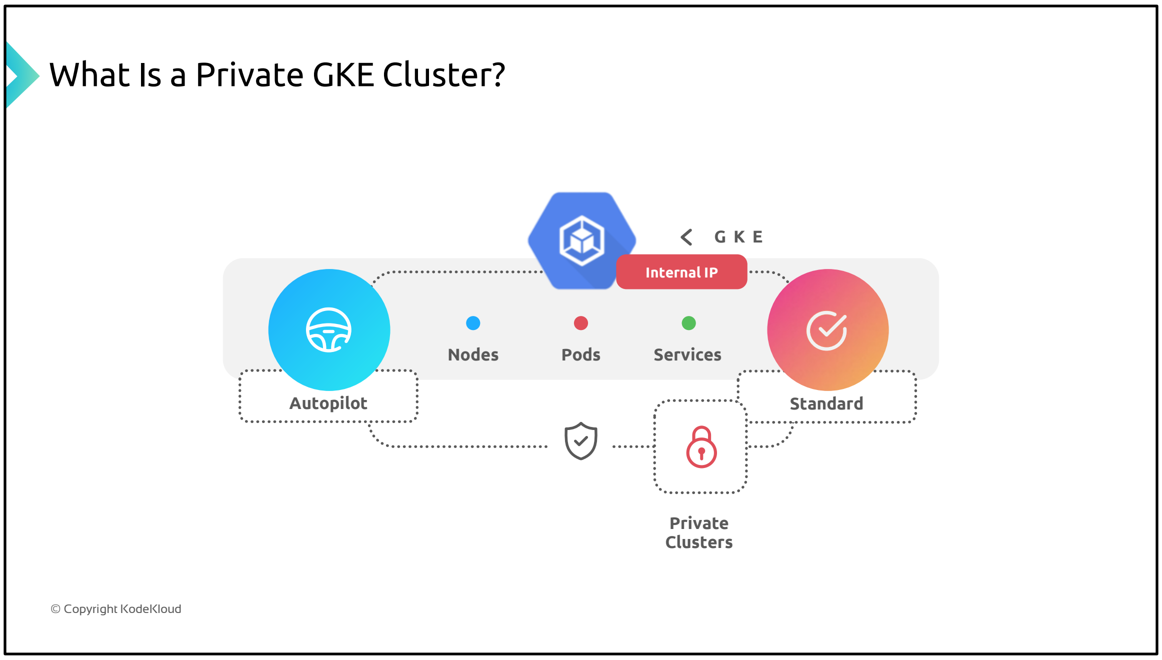
In a GKE or Google Kubernetes Engine cluster, connectivity between Pods and Services is managed by the **kube-proxy component**. By default, kube-proxy is deployed as a DaemonSet, ensuring that a Pod running kube-proxy is available on each node in the cluster. It operates as an egress-based load-balancing controller rather than an in-line proxy. kube-proxy continuously monitors the Kubernetes API server to maintain a mapping between the Service's ClusterIP and the healthy Pods associated with it.
kube-proxy achieves this by dynamically adding and

removing Destination NAT (DNAT) rules to the node's IP tables subsystem. When a container in a Pod sends traffic to a Service's ClusterIP, the node, through kube-proxy, randomly selects a healthy Pod and routes the traffic to it.

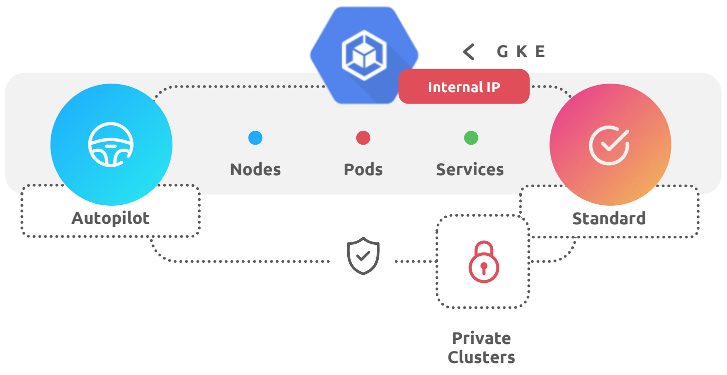
When configuring a Service, you have the option to remap the listening port and the targetPort. The port is the entry point where clients reach the application, while the targetPort is the port where the application is actually listening for traffic within the Pod. kube-proxy manages this port remapping by manipulating the IP tables rules on the node.

This diagram here illustrates the flow of traffic from a client Pod on the left hand side to a server Pod on the right hand side on a different node. The client connects to the Service's IP and port (e.g., 172.16.0.100:80). The Kubernetes API server maintains a list of Pods running the application, and kube-proxy on each node utilizes this information to create an IP tables rule that directs the traffic to the appropriate Pod (e.g., 10.16.2.102:8080). The client Pod does not require knowledge of the cluster's topology or specific details about individual Pods or containers.

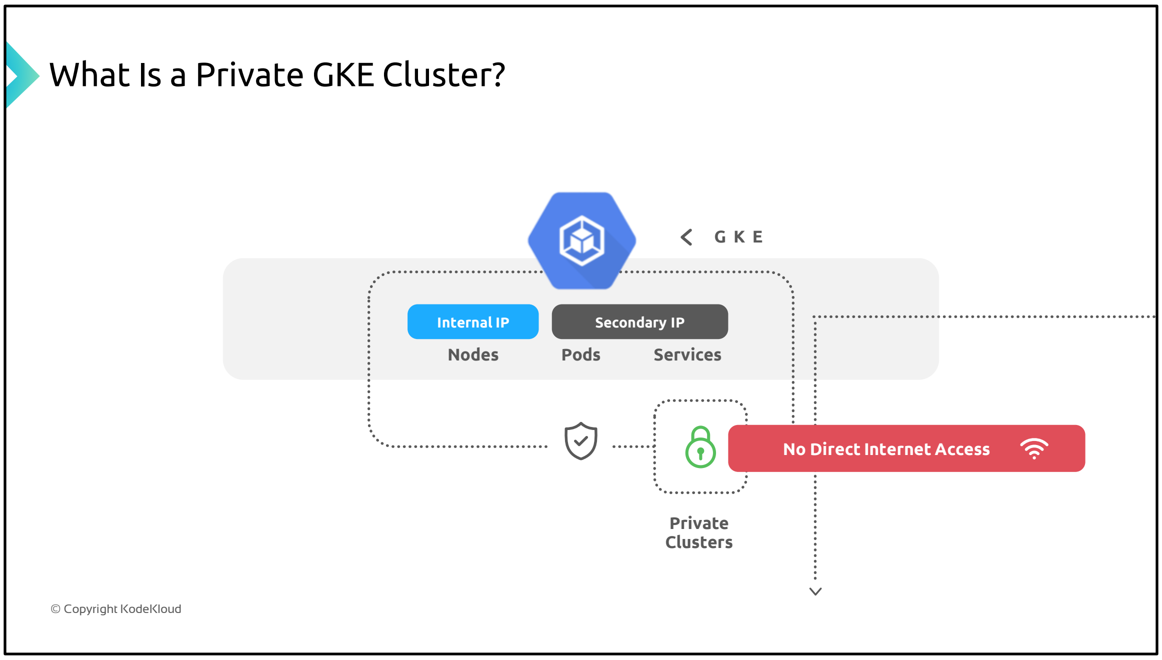
What Is a Private GKE Cluster?



What Is a Private GKE Cluster?



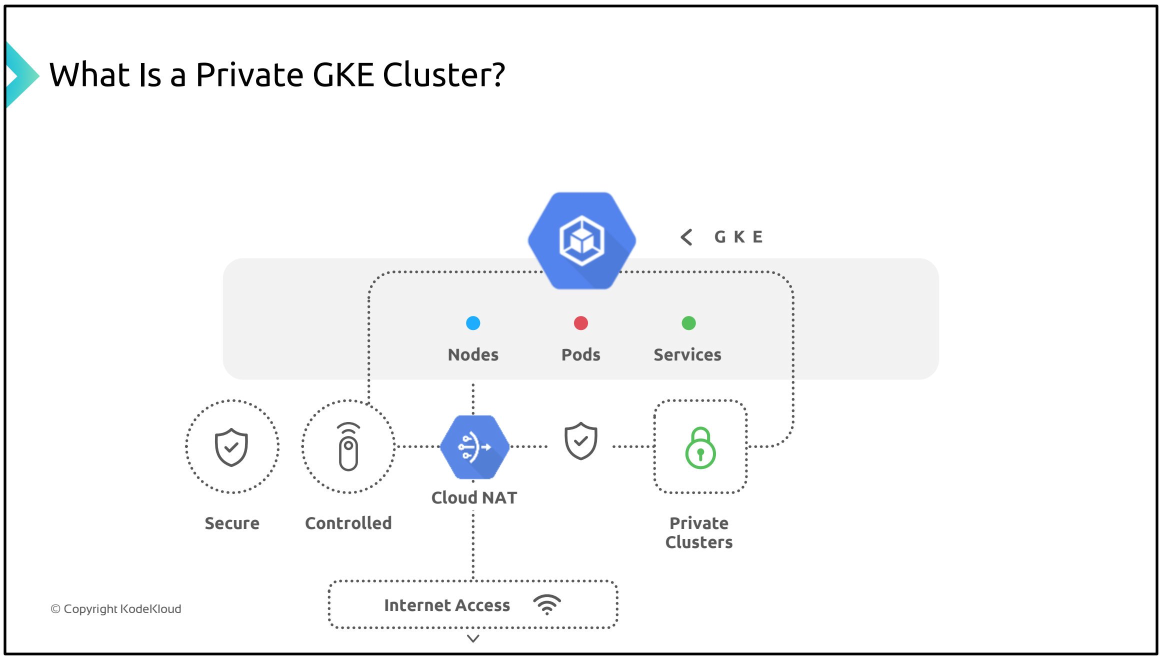
A private GKE, or Google Kubernetes Engine, cluster operates using only internal IP addresses. It is designed to provide enhanced security and isolation by ensuring that nodes, Pods, and Services within the cluster can only be accessed using internal networking.
Private clusters can be created and configured in either Standard or Autopilot mode.



What Is a Private GKE Cluster?

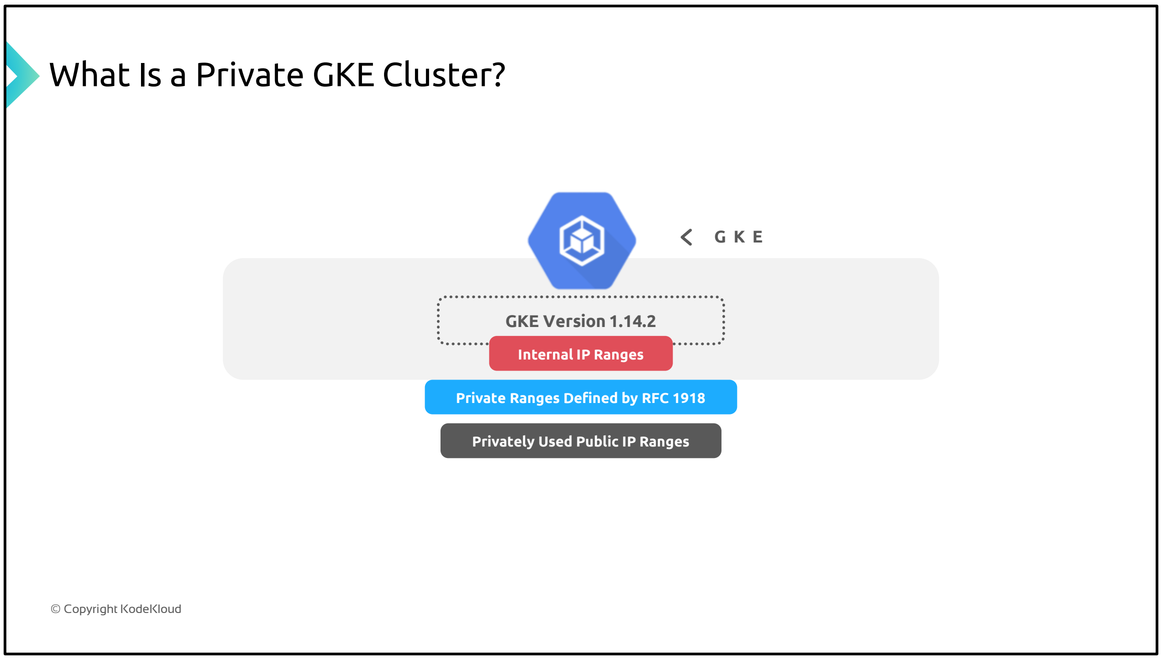
By default, a private cluster does not allow direct internet access for nodes and Pods. This isolation ensures that the cluster's resources are not directly exposed to the public internet.
In a private cluster, nodes are assigned only internal IP addresses. These IP addresses are obtained from the primary IP address range of the subnet that you choose for the cluster. This means that the nodes' communication is limited to the internal network, and they cannot be reached from the internet.

Similarly, Pod IP addresses and Service IP addresses within a private cluster also come from secondary IP address ranges of the same subnet. These IP addresses are unique within the cluster and are used for internal communication between Pods and Services.



What Is a Private GKE Cluster?

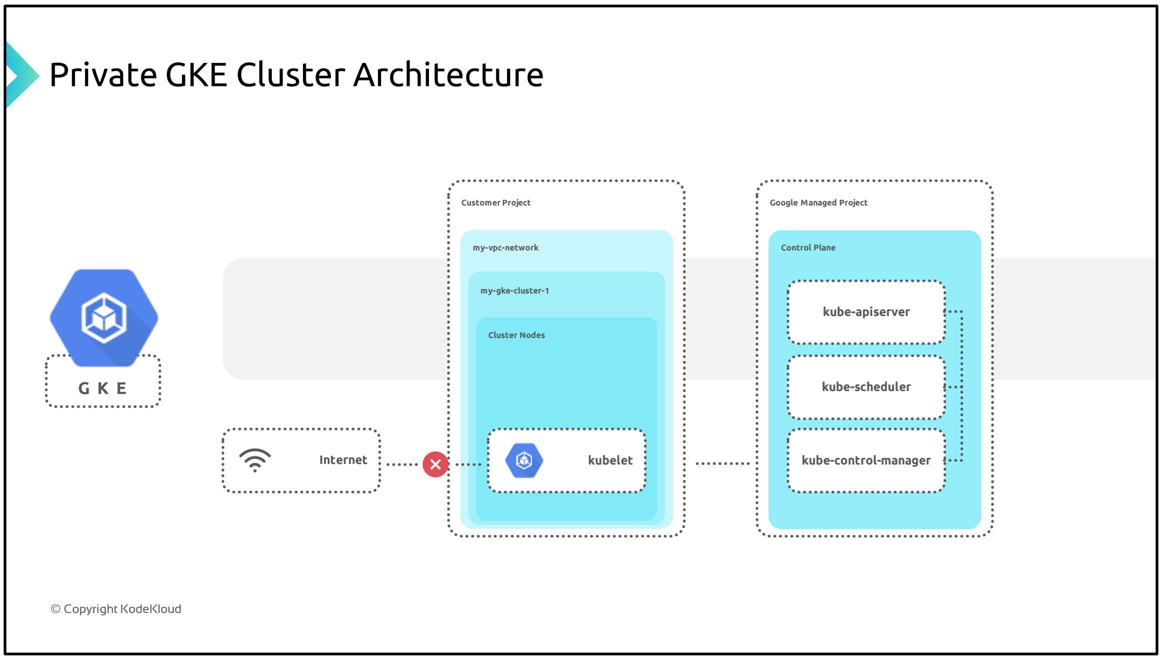
If there is a requirement to provide outbound internet access for specific nodes in the private cluster, you can utilize Cloud NAT, also called Network Address Translation. Cloud NAT allows nodes to access the internet while maintaining their internal IP addresses. This enables controlled and secure outbound connectivity for the nodes without exposing them directly to the internet.



What Is a Private GKE Cluster?

GKE
GKE Version 1.14.2
Internal IP Ranges
Private Ranges Defined by RFC 1918
Privately Used Public IP Ranges

It's important to note that GKE version 1.14.2 and later versions support various internal IP address ranges, including private ranges defined by RFC 1918 and other privately used public IP address ranges. This flexibility allows you to choose IP address ranges that align with your network configuration and address space requirements.



Private GKE Cluster Architecture

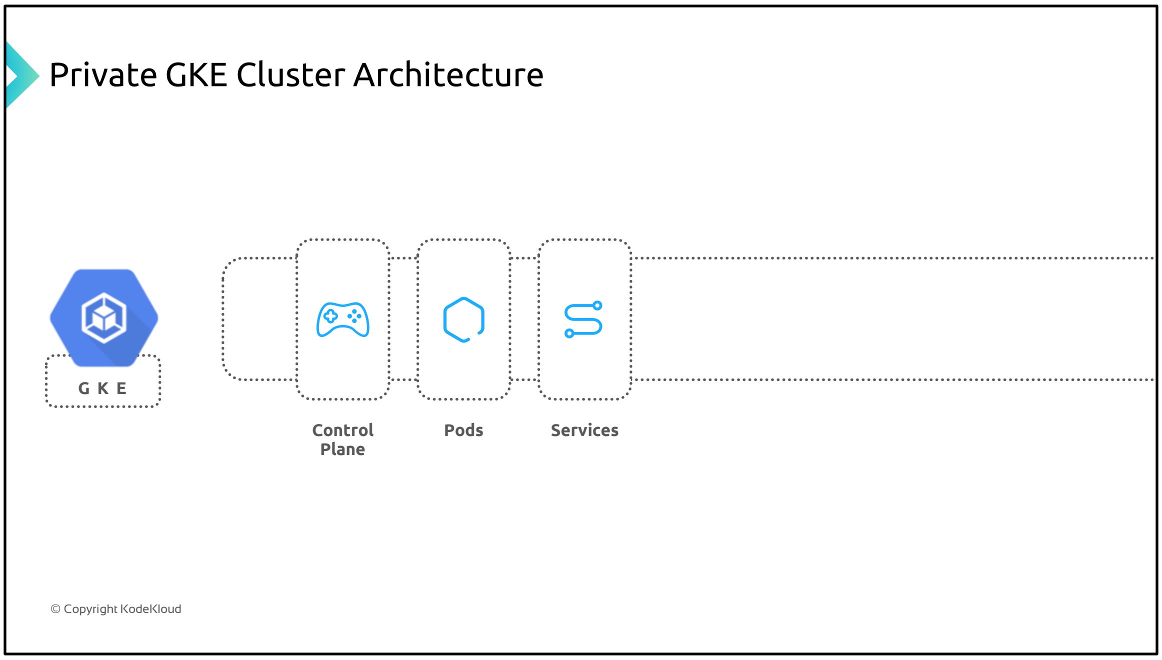
© Copyright KodeKloud

A private GKE cluster follows a specific architecture to ensure secure and isolated operations. The different components of a GKE cluster are separated out and created in two different projects, known as a customer project and Google managed project.

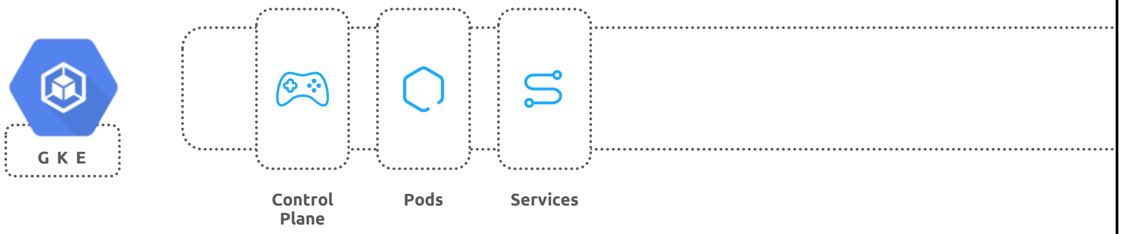
Under the Shared Responsibility Model, Google manages all the GKE control plane components which include the Kubernetes API server, etcd storage, scheduler, and other controllers. All the security aspects of these components in a control plane are managed by Google in a project that's created separately by Google. As a user, you don't have visibility of or access to this project, though you are able to configure certain options for control plane based on your requirements.

You are responsible for securing all the workload-related components in the cluster like the nodes, containers, and Pods in a customer project, which is the project that is specified during cluster creation time.

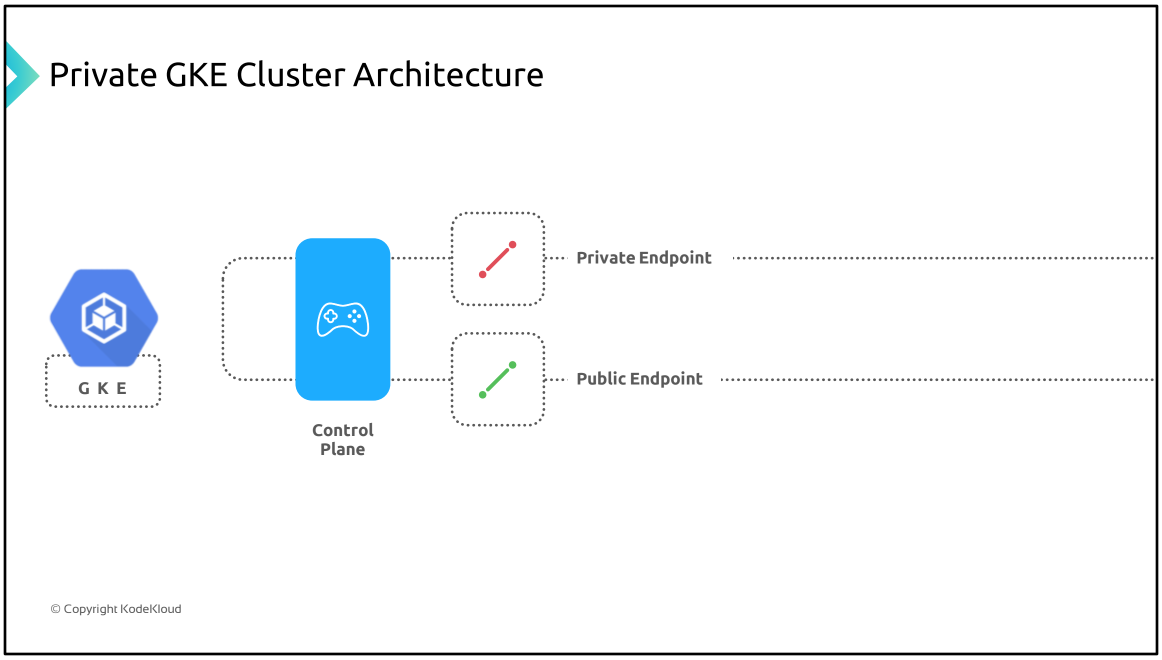
We’ll discuss more about the shared responsibility model and the different user configurable options for a control plane in subsequent sections.



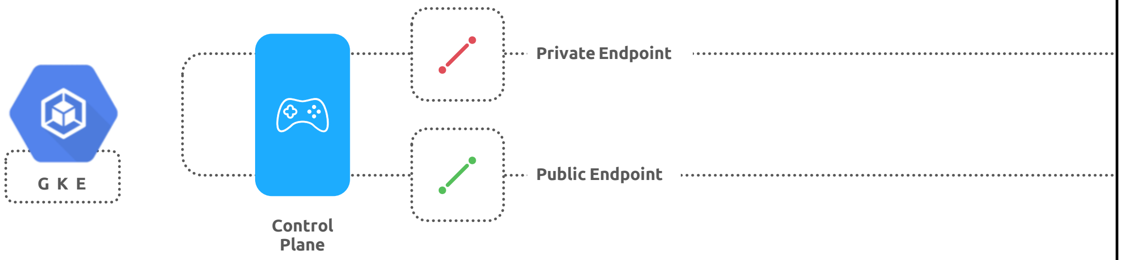
Private GKE Cluster Architecture



Let's delve into the key components and their interactions within a private GKE cluster.

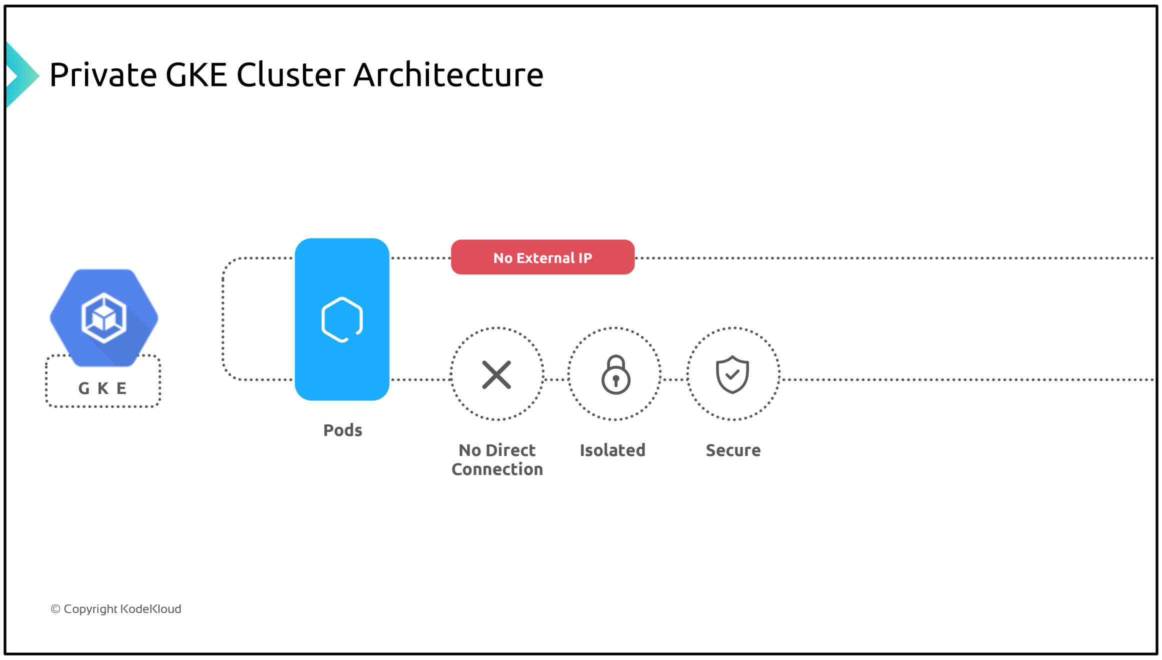


Private GKE Cluster Architecture



Control Plane:

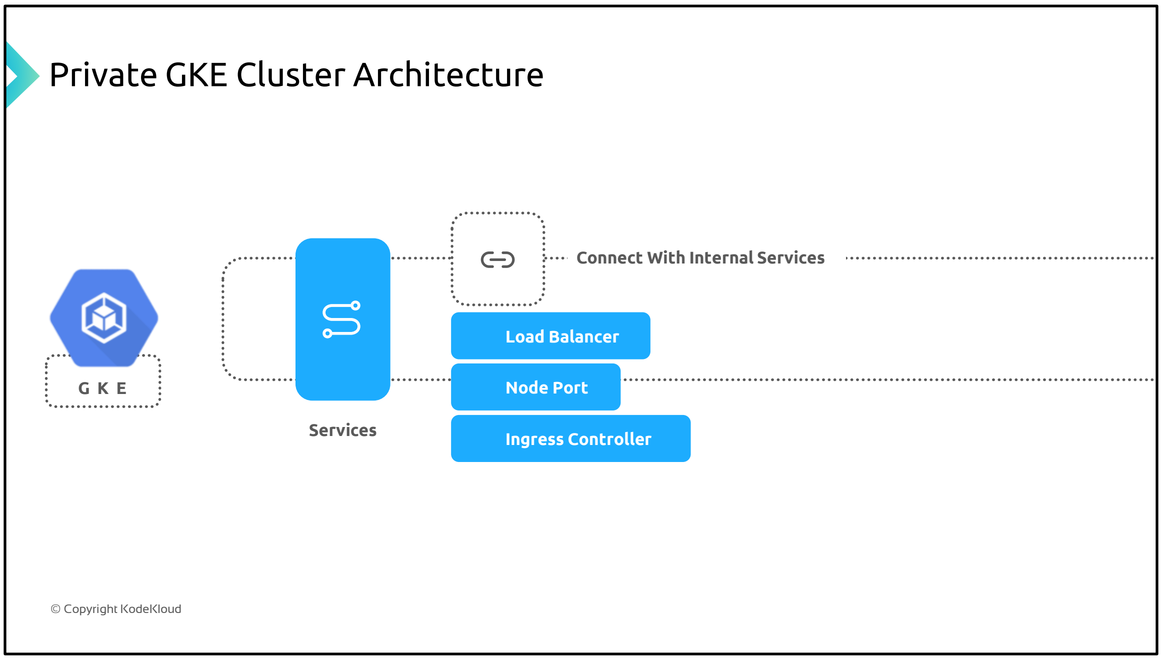
- Private Endpoint: The control plane has a private endpoint that is accessible only within the cluster's VPC network. This private endpoint allows communication between the control plane components and other cluster resources.
- Public Endpoint (optional): In addition to the private endpoint, the control plane can also have a public endpoint. However, you have the option to disable this public endpoint for enhanced security. The public endpoint enables external access to the control plane for management and administrative purposes.



Private GKE Cluster Architecture

Nodes: The nodes within a private cluster do not have external IP addresses. They are assigned internal IP addresses from the primary IP address range of the chosen subnet within the cluster's VPC network. These internal IP addresses are used for intra-cluster communication and are not accessible from the public internet.

Due to the lack of external IP addresses, clients on the internet cannot directly connect to the nodes in a private cluster. This ensures that the nodes are isolated from external access and helps enhance the security of the cluster.

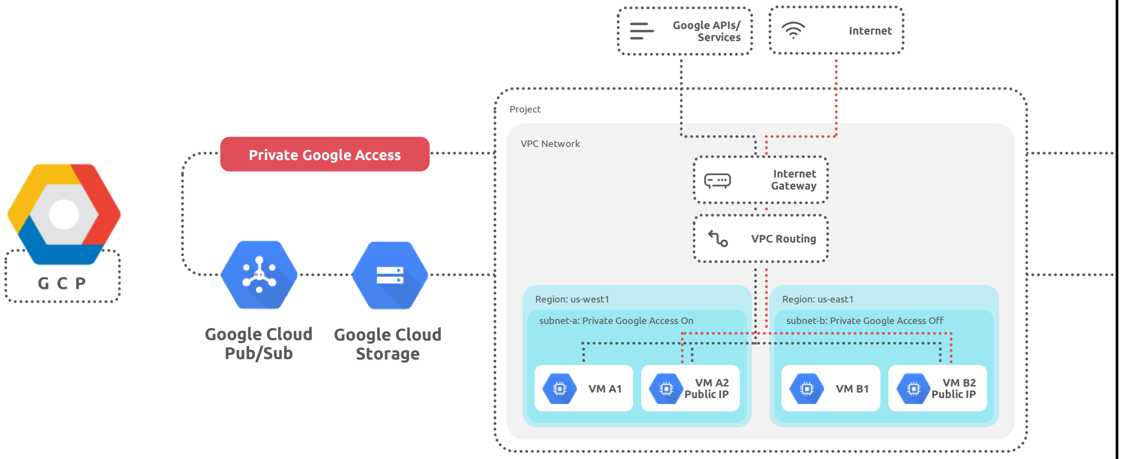


Private GKE Cluster Architecture

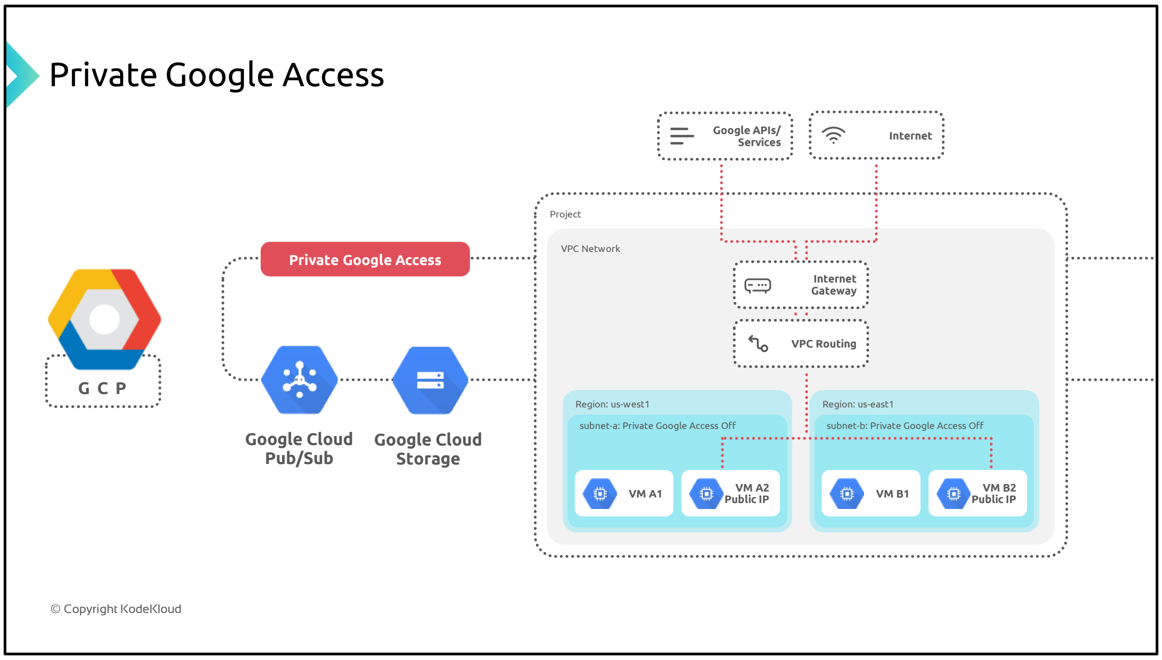
Services: Although the nodes in a private cluster do not have external IP addresses, external clients can still connect to Services hosted within the cluster.

External clients can connect to external services within the cluster by leveraging load balancers or Ingress resources using different Service types:
•LoadBalancer: A service of type LoadBalancer can be exposed to external clients using a Google Cloud Load Balancing service. This will create a public IP address that can be used to access the service.
•NodePort: A service of type NodePort can be exposed to external clients by opening a port on each node in the cluster. This allows clients to connect to the service directly, without going through the control plane .
•Ingress: An Ingress controller can be used to expose services to external clients using a variety of protocols, such as HTTP, HTTPS, and TCP.

Private Google Access

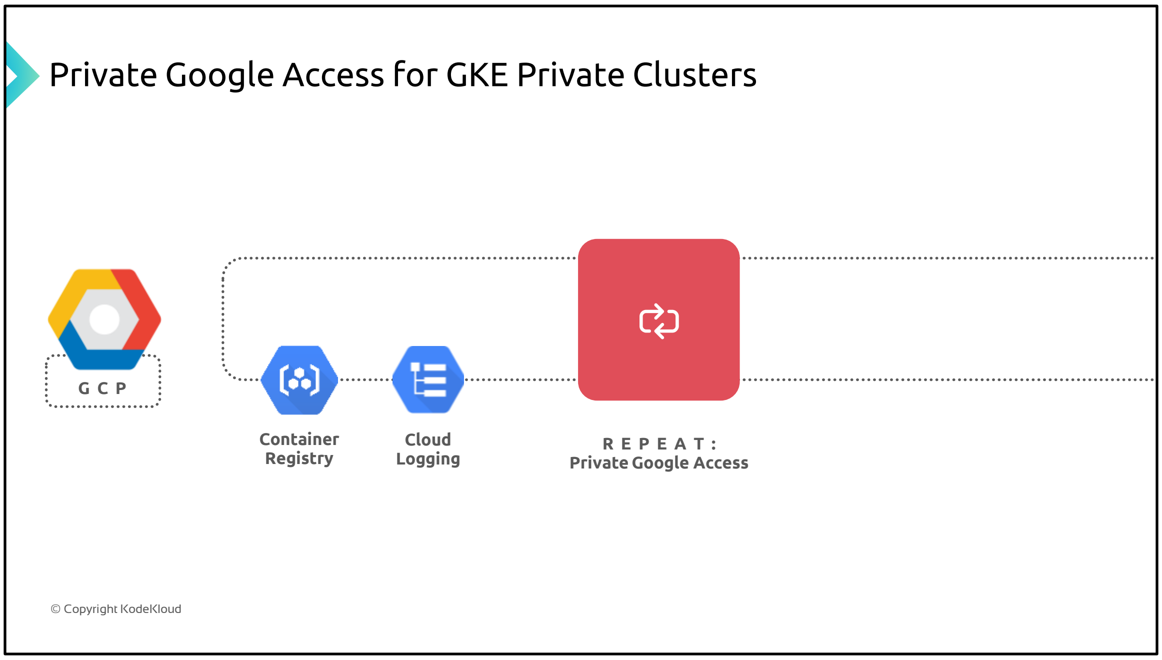


A quick note here on Private Google Access in Google Cloud Platform (GCP). This feature allows VM instances with only internal IP addresses and no external IP addresses to access Google APIs and services. It enables these instances to reach the external IP addresses associated with Google services, even though they don't have direct internet access. This allows these instances to interact with various Google services, such as Cloud Storage, Cloud Pub/Sub, and others, without requiring direct internet access. It is enabled at the subnet level within a VPC network.



Private Google Access

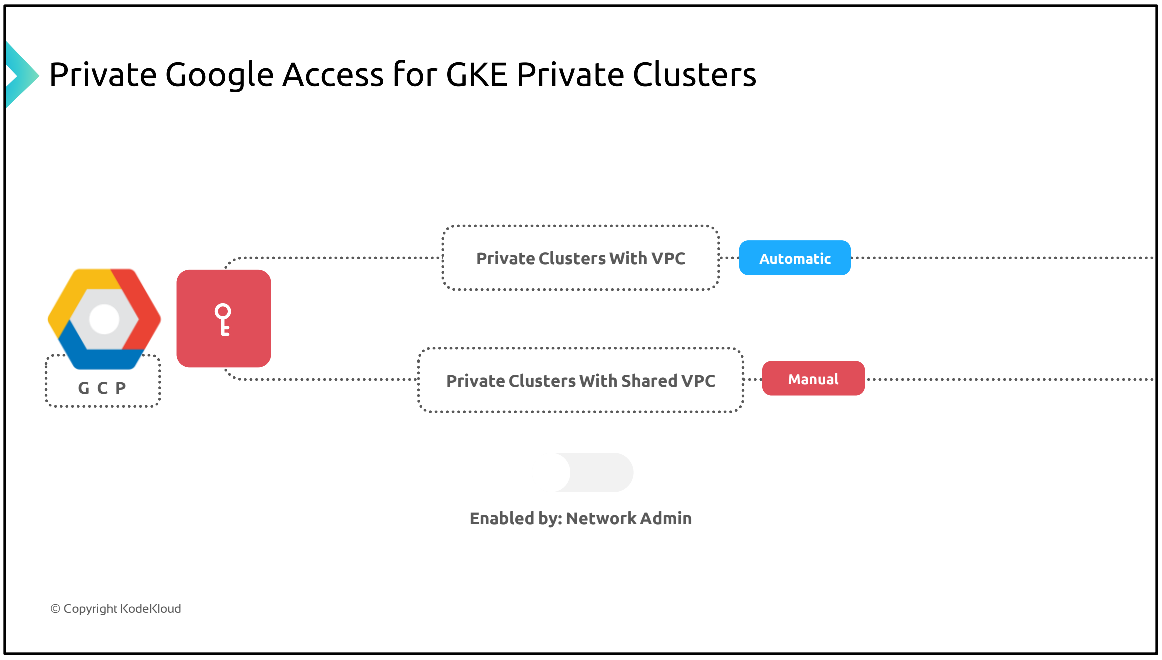
If Private Google Access is disabled for a subnet, VM instances within that subnet lose the ability to communicate with Google APIs and services. In this case, the instances can only send traffic within the VPC network and cannot establish connections to external Google services.



Private Google Access for GKE Private Clusters

GCP
Container Registry
Cloud Logging
REPEAT:
Private Google Access

Now, we learned that the nodes in a private GKE cluster only have internal IP address without being assigned an external public IP address. So, Private Google Access is utilized to provide private nodes and their workloads access to Google Cloud APIs and services over Google's private network. Private Google Access allows the cluster to interact with various Google services and resources, such as container images from Artifact Registry and sending logs to Cloud Logging.



Private Google Access for GKE Private Clusters

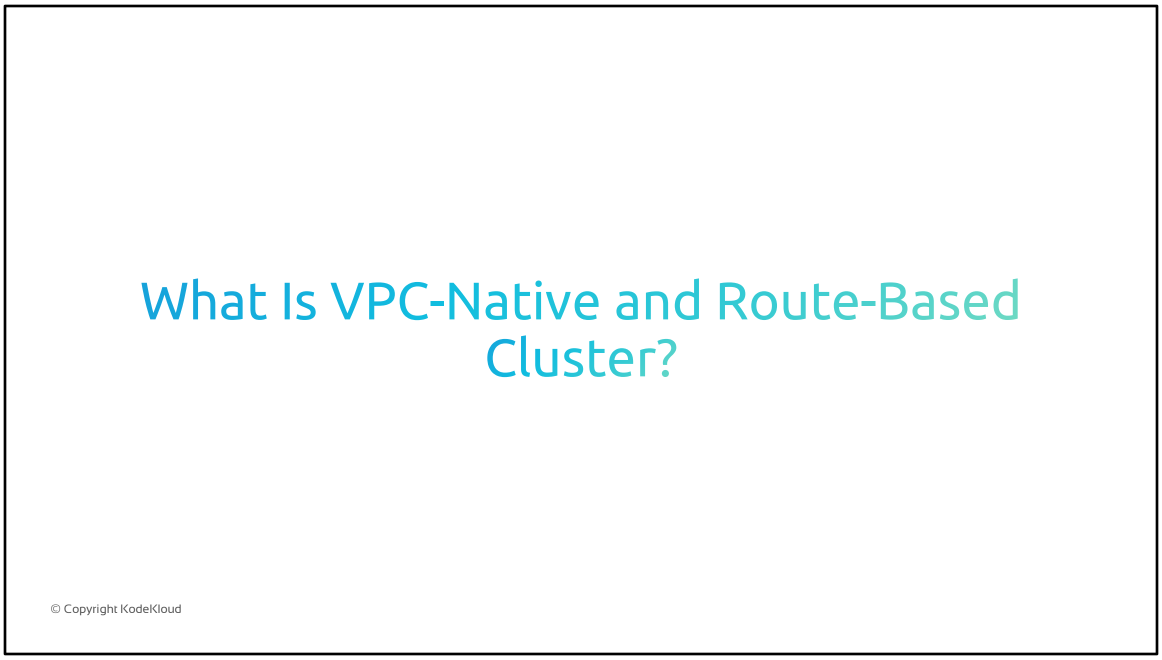
G C P
Private Clusters With VPC
Automatic
Private Clusters With Shared VPC
Manual
Enabled by: Network Admin

By default, Private Google Access is enabled for private clusters that use VPC networks in the same project as the cluster. When creating the private cluster, GKE ensures that Private Google Access is enabled on the subnet used by the cluster.

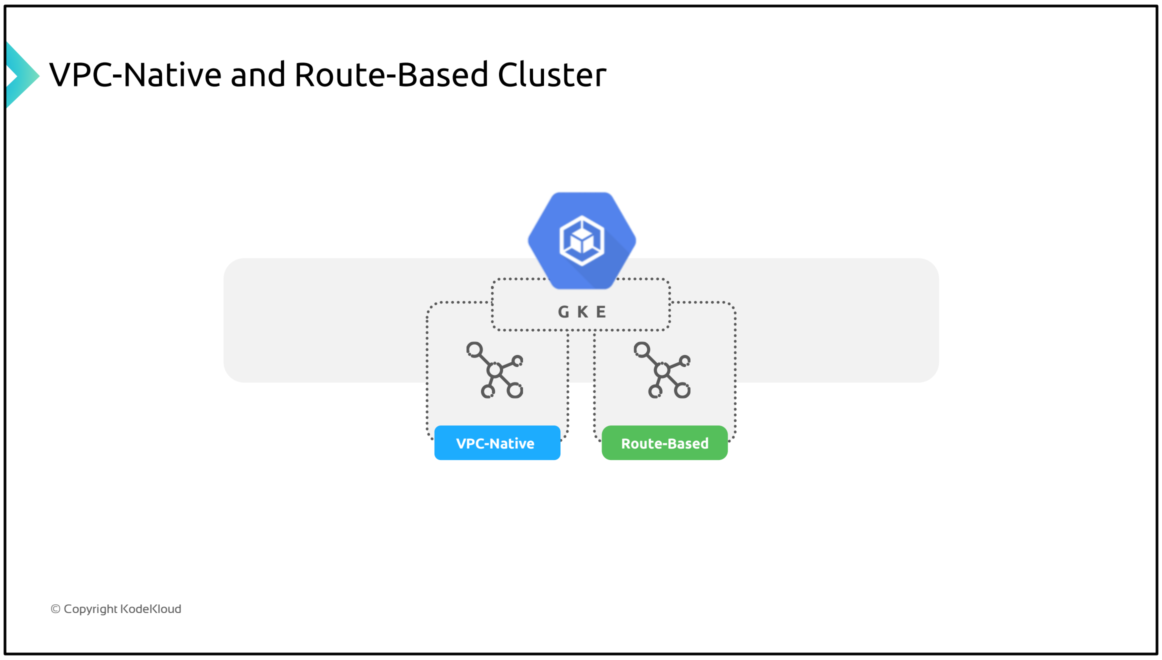
However, in the case of private clusters created in Shared VPC service projects, Private Google Access is not automatically enabled. A network admin, project owner, or project editor for the Shared VPC host project needs to manually enable Private Google Access on the subnets used by the private clusters.

https://kodekloud.com/

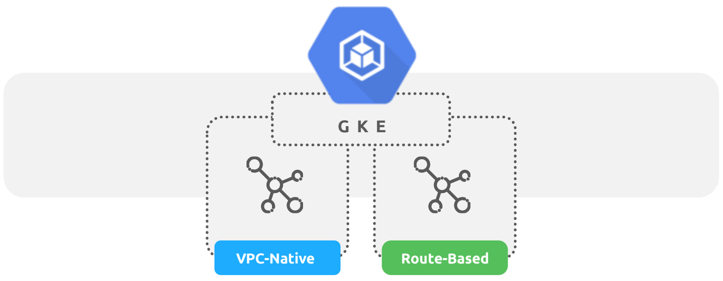
https://kodekloud.com/courses/gke-google-
kubernetes-engine/



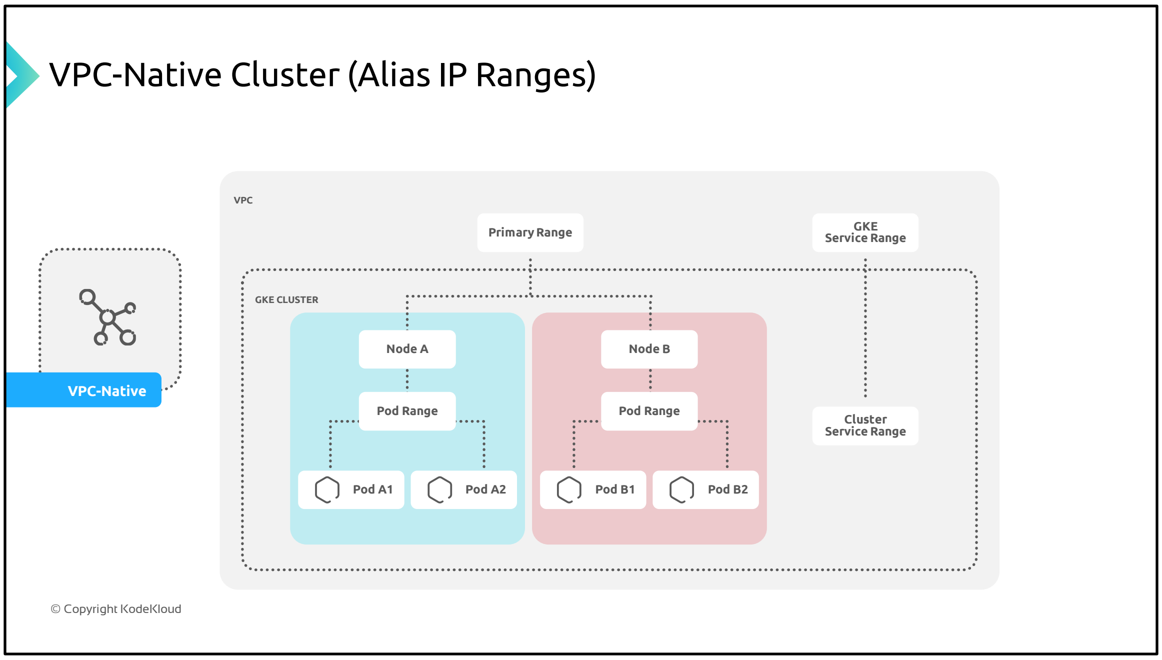
What Is VPC-Native and Route-Based Cluster?



VPC-Native and Route-Based Cluster

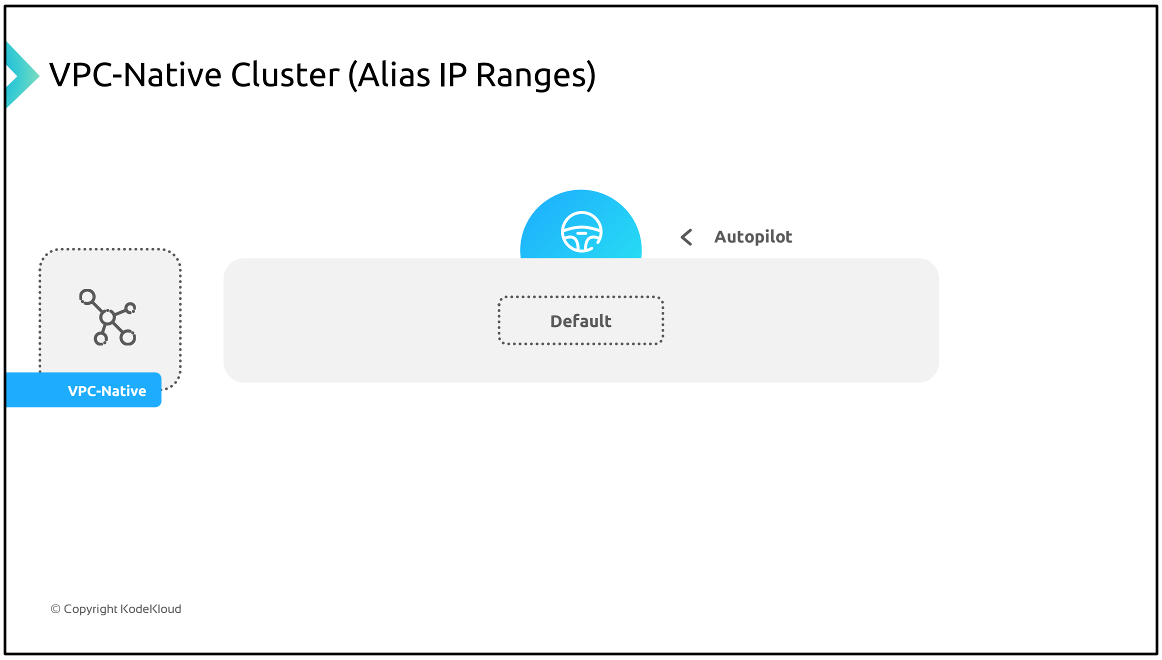


In Google Kubernetes Engine, clusters can be categorized based on how they route traffic between Pods. There are two types of clusters: VPC-native clusters and route-based clusters.



VPC-Native Cluster (Alias IP Ranges)

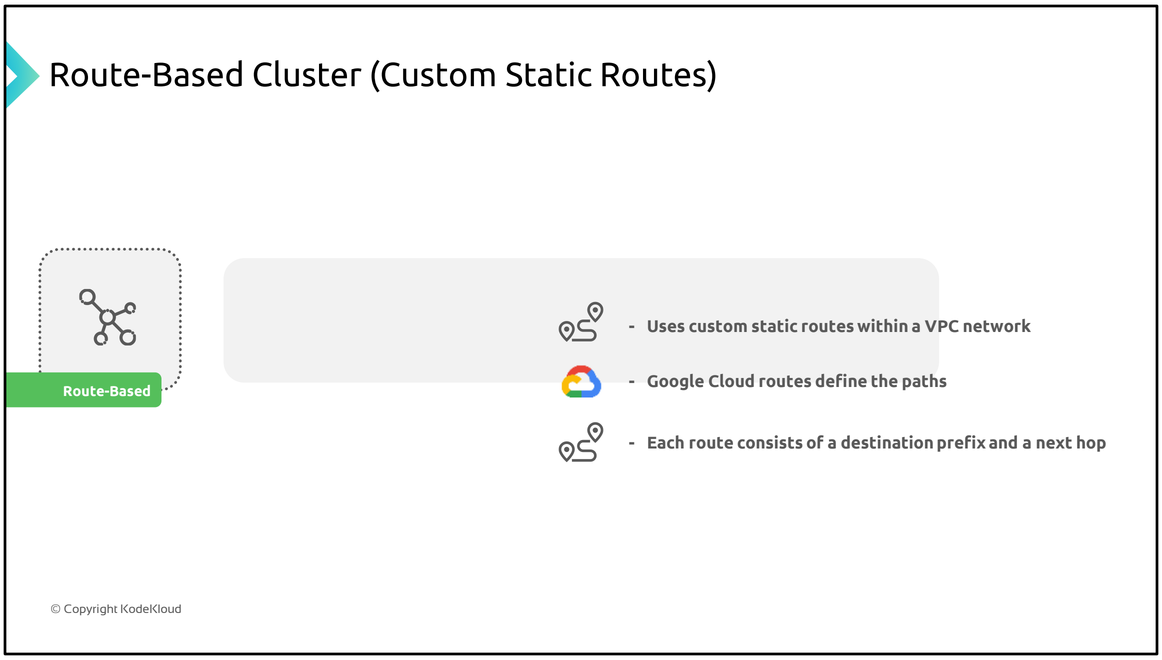
A VPC-native cluster utilizes alias IP address ranges. Alias IP ranges allow you to assign specific internal IP addresses as aliases to the network interfaces of virtual machines (VMs) or GKE Pods. This is useful when you have multiple services running on a VM or multiple Pods within a GKE cluster and you want to assign unique IP addresses to each service or Pod. In a VPC-native cluster, traffic routing between Pods is handled using these alias IP addresses.



VPC-Native Cluster (Alias IP Ranges)

Autopilot
Default
VPC-Native

It's important to note that for GKE Autopilot clusters, which are a fully managed option in GKE, VPC-native traffic routing is enabled by default.

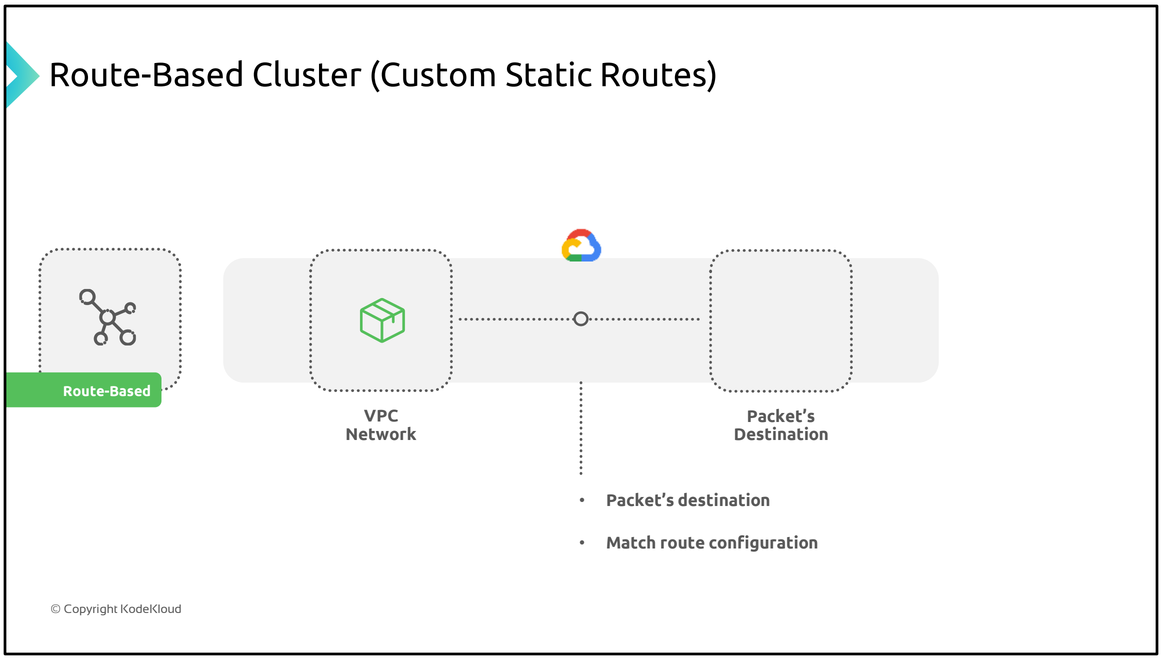


Route-Based Cluster (Custom Static Routes)

Route-Based

- Uses custom static routes within a VPC network
- Google Cloud routes define the paths
- Each route consists of a destination prefix and a next hop

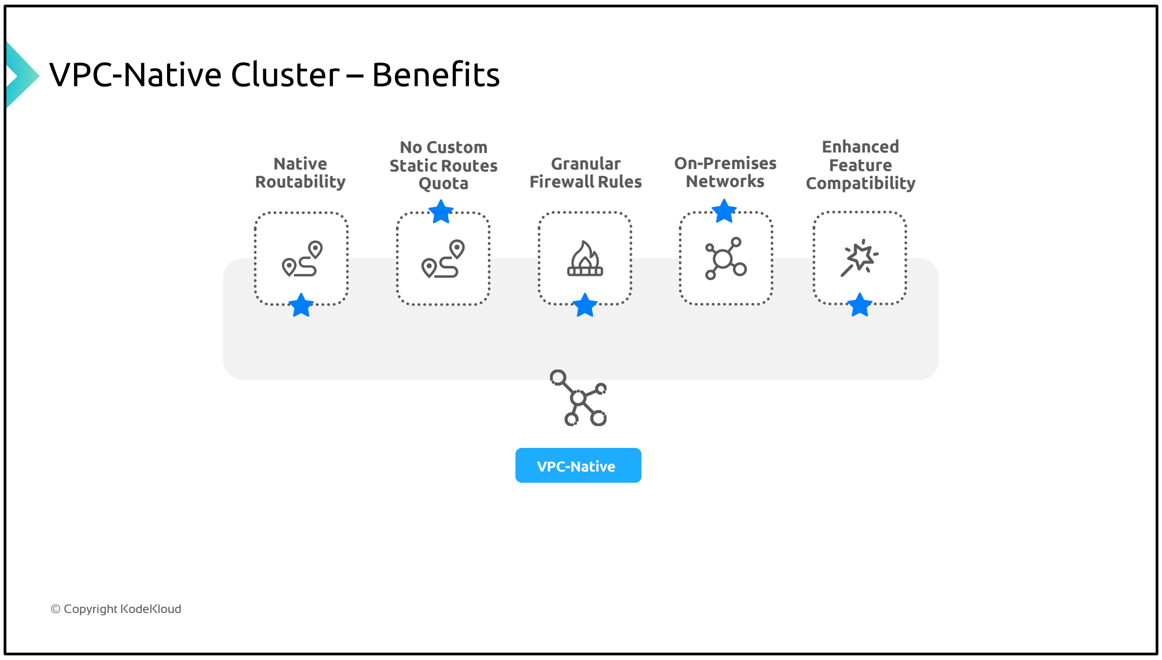
A route-based cluster, on the other hand, **uses custom static routes** within a VPC network for traffic routing between Pods. In a route-based cluster, **Google Cloud routes define the paths** that network traffic takes from one instance to another within the VPC network. **Each route consists of a destination prefix (specified in CIDR format) and a next hop**.



Route-Based Cluster (Custom Static Routes)

- Packet's destination
- Match route configuration

When a packet is sent from an instance in the VPC network, Google Cloud delivers the packet to the appropriate next hop based on the packet's destination address and the matching route configuration. Route-based clusters rely on custom static routes for internal traffic routing.



VPC-Native Cluster – Benefits

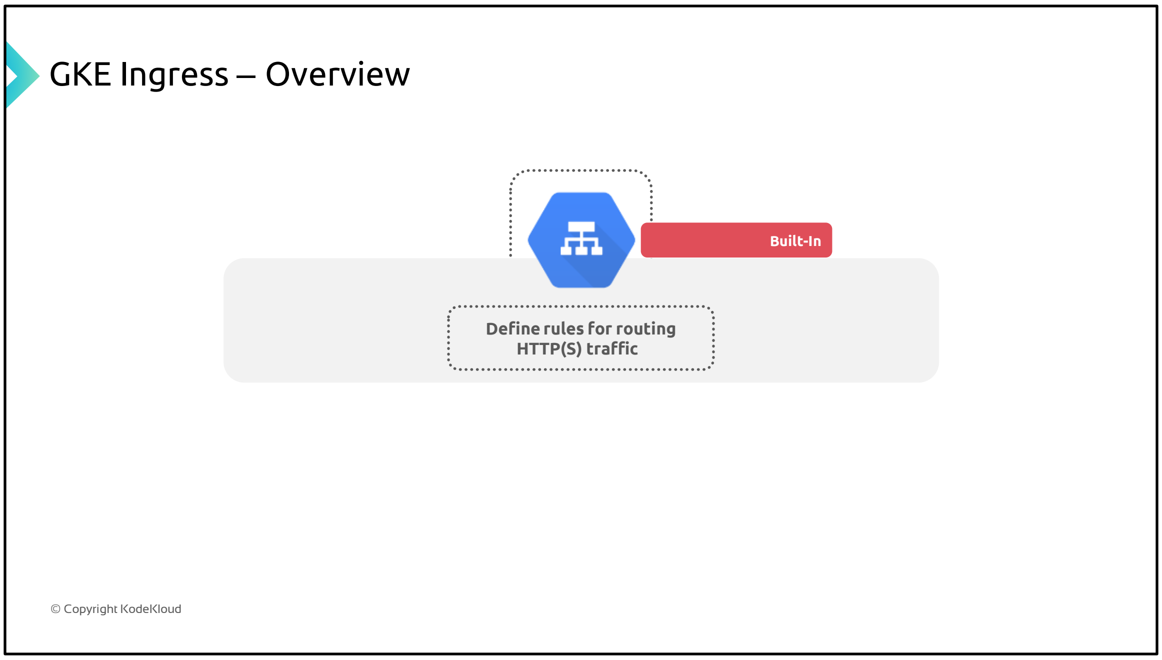
VPC-native clusters in GKE offer several benefits compared to route-based clusters, such as:

1. Native Routability: Pod IP addresses within a VPC-native cluster are natively routable within the cluster's VPC network and any other VPC networks connected to it through VPC Network Peering.

2. No Custom Static Routes Quota: In VPC-native clusters, Pod IP address ranges are handled differently than route-based clusters. They do not depend on custom static routes and, therefore, do not consume the system-generated and custom static routes quota.

3. **Granular Firewall Rules:** VPC-native clusters allow you to create firewall rules that specifically target Pod IP address ranges. This level of granularity provides enhanced security and control by allowing you to apply firewall policies directly to Pods, rather than applying them to all IP addresses associated with the cluster's nodes.
4. **Connectivity with On-Premises Networks:** The Pod IP address ranges in VPC-native clusters, including the secondary IP address ranges, can be accessed from on-premises networks connected to the cloud environment through Cloud VPN or Cloud Interconnect using Cloud Routers. This enables seamless integration between your on-premises infrastructure and the Pods running in the VPC-native cluster.
5. **Enhanced Feature Compatibility:** Certain features and services, such as network endpoint groups (NEGs), are designed to work specifically with VPC-native clusters.

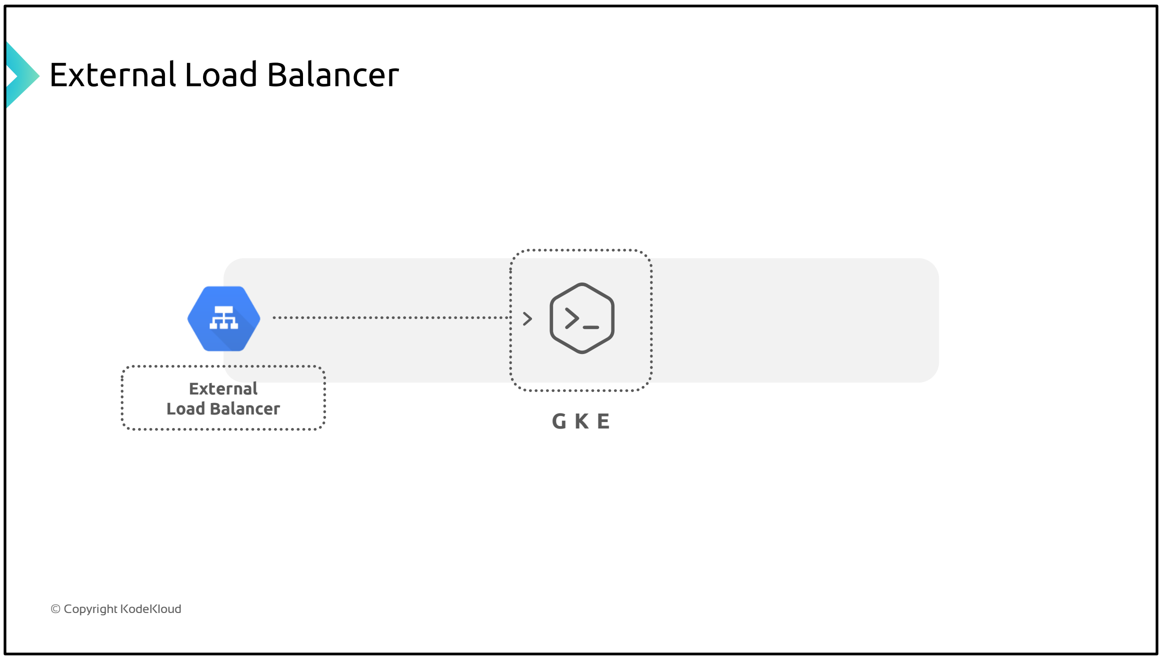
GKE Ingress – Overview



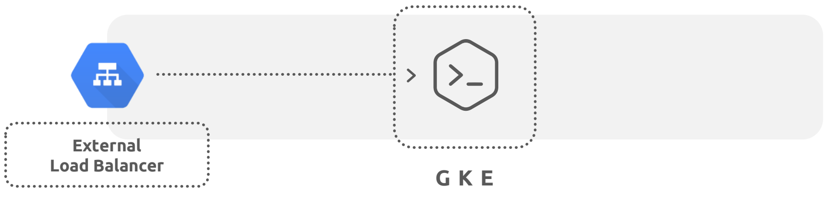
GKE Ingress – Overview

Kubernetes Ingress is a way to define rules that route incoming HTTP(S) traffic to applications running inside a Kubernetes cluster. In the GKE world, Ingress for incoming traffic is managed by Google by creating a load balancer that sits in front of the applications and distributes traffic to them based on the defined rules.

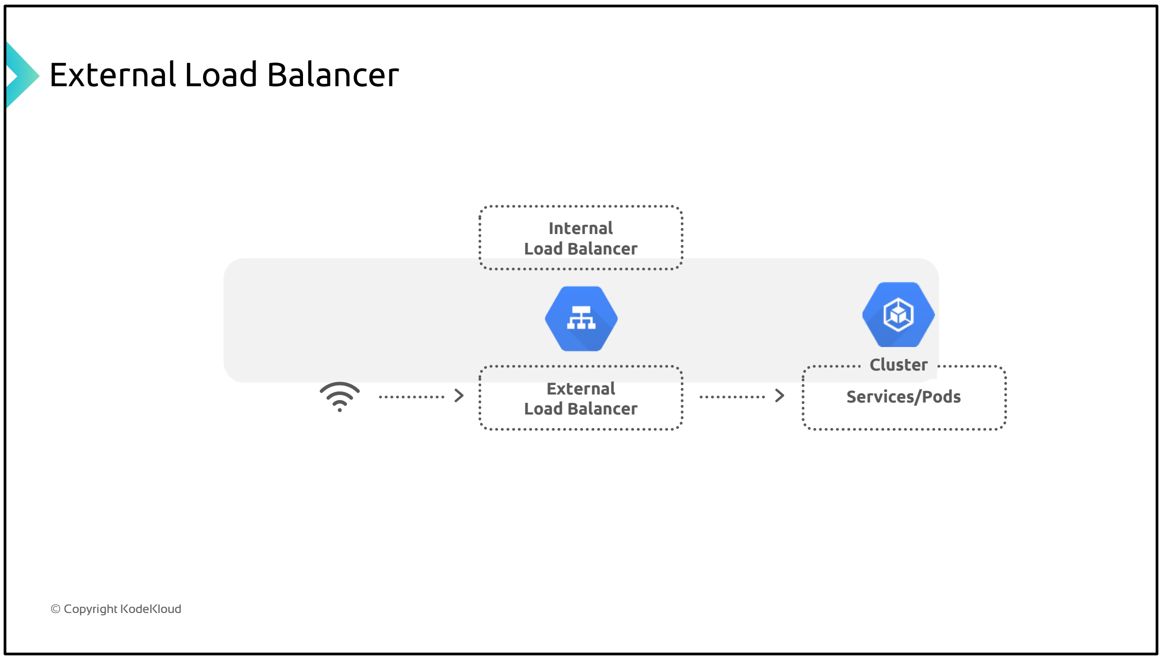
When an Ingress object is created, GKE creates and configures an external or internal load balancer based on the information in the Ingress and associated services.



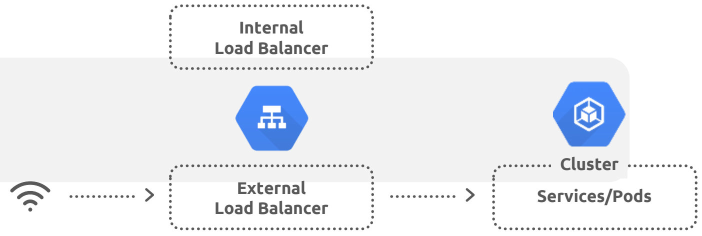
External Load Balancer



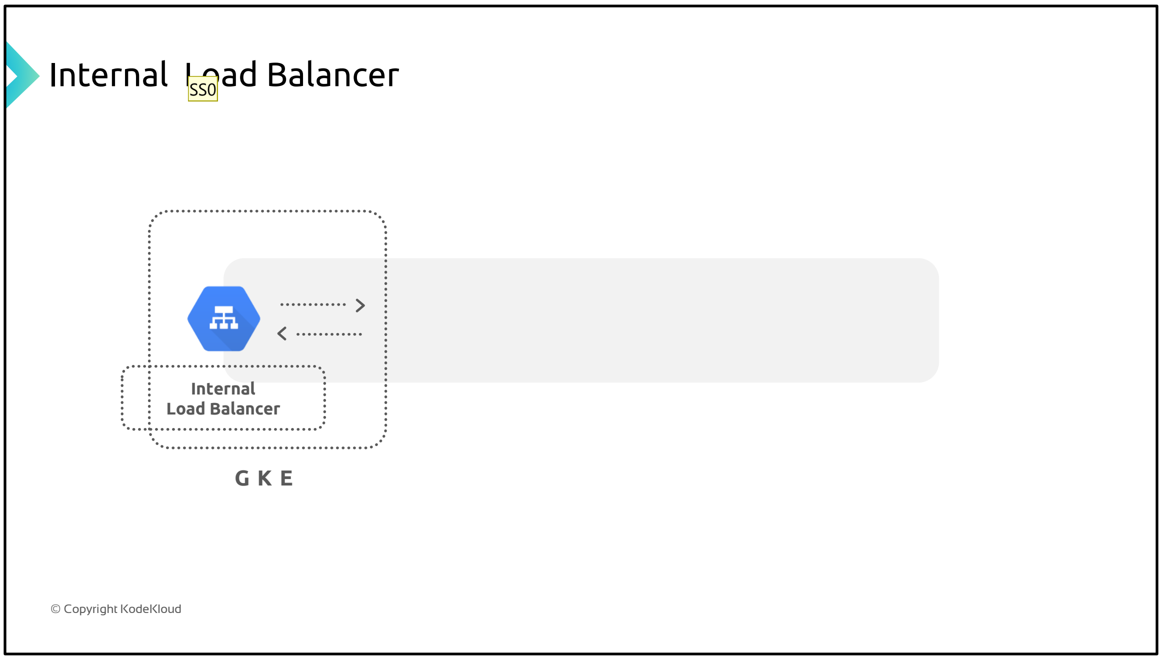
•External load balancers are used to route traffic from the internet to applications running in GKE.



External Load Balancer



So, when the ingress is configured for the HTTP(S) traffic coming from the external network, an external Application Load Balancer directs it to the appropriate Services and Pods.

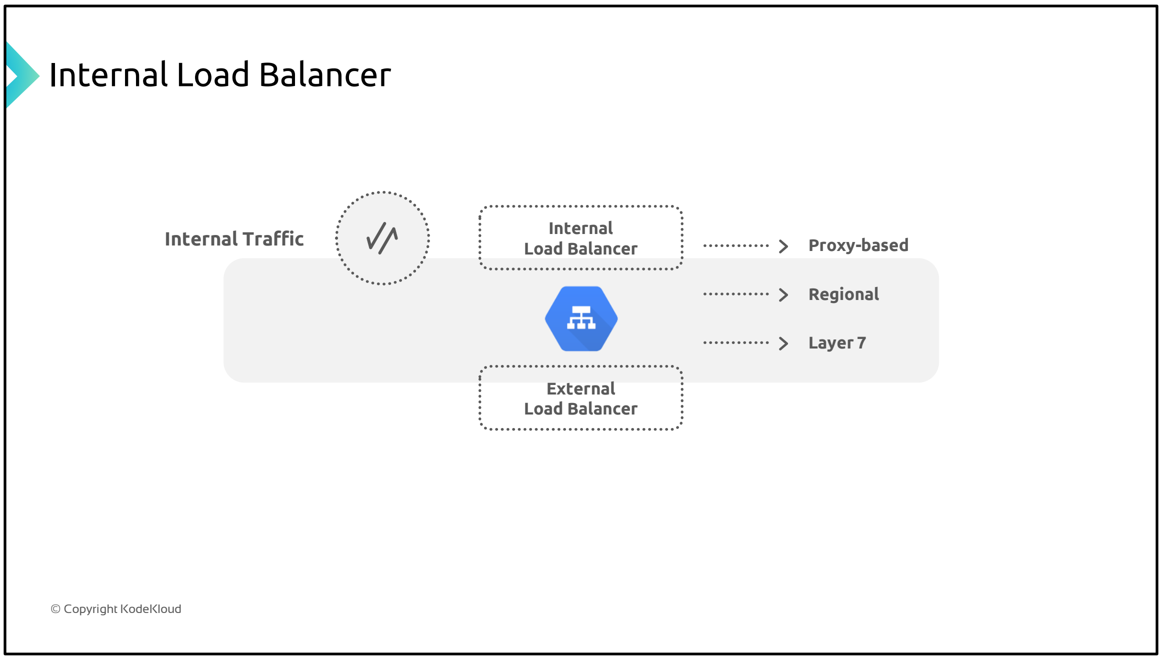


Internal Load Balancer

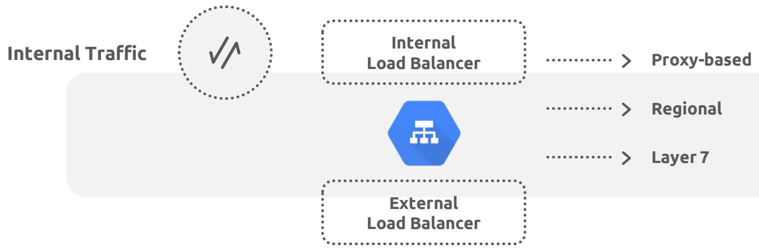
SS0

Internal
Load Balancer
G K E

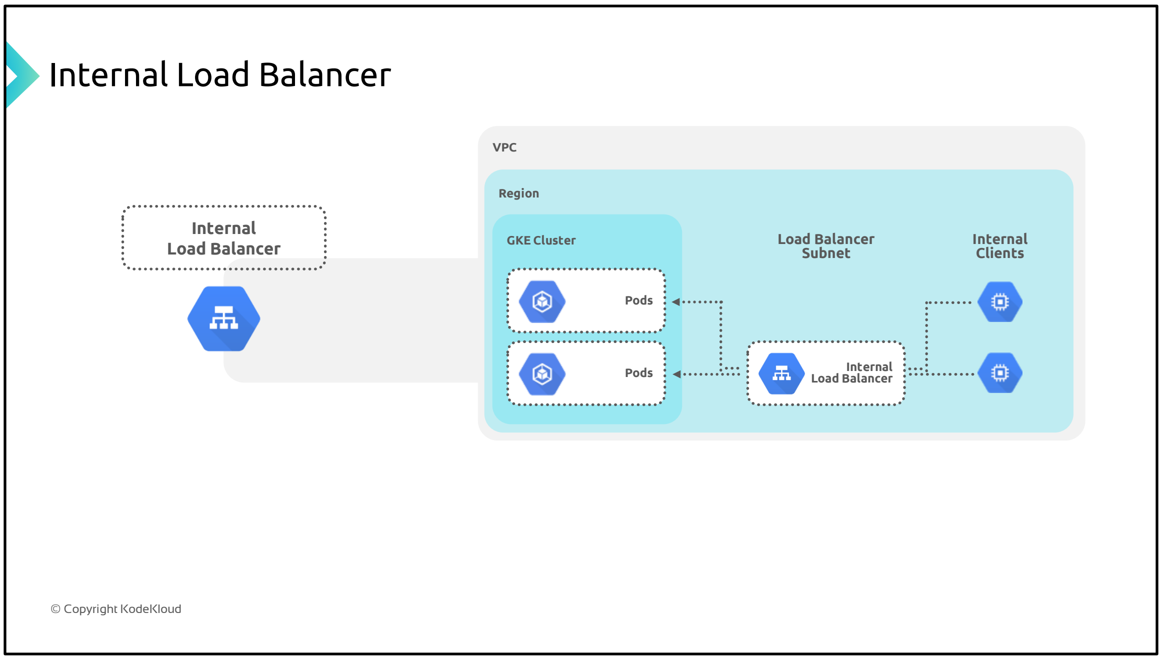
•Internal load balancers on the other hand are used to route traffic within a GKE cluster or VPC network.



Internal Load Balancer

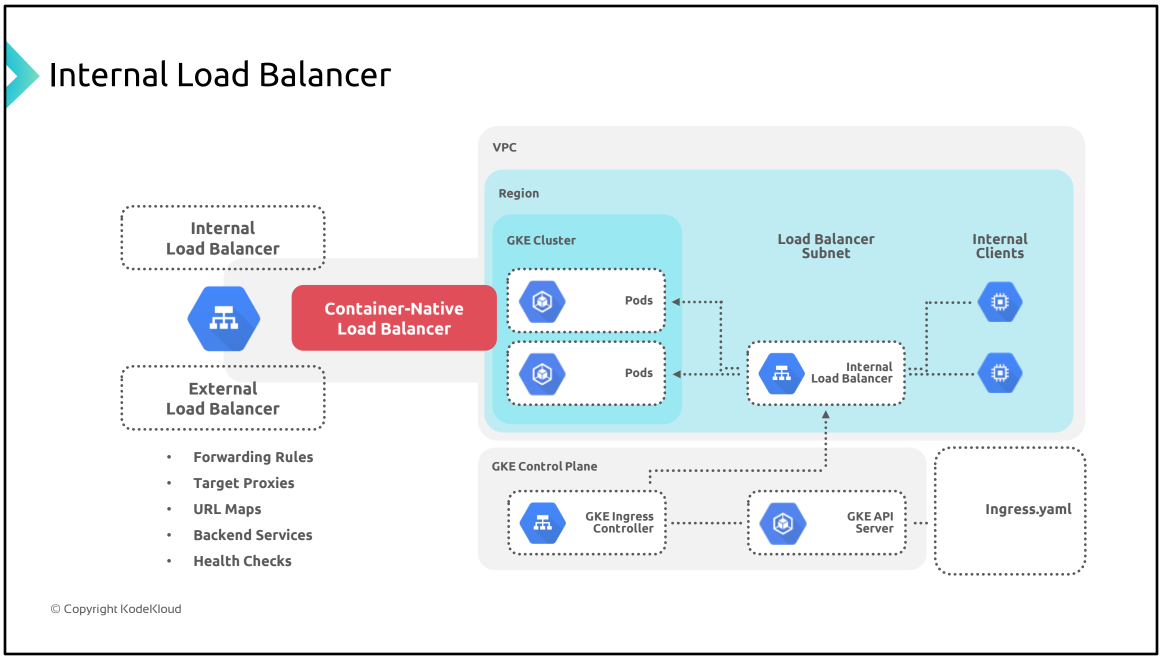


So, when the ingress is configured for the traffic coming from within the cluster or same VPC network, an internal Application Load Balancer distributes that traffic to relevant services using their internal load balancing IP addresses. The internal application load balancer is a proxy based, regional load balancer.



Internal Load Balancer

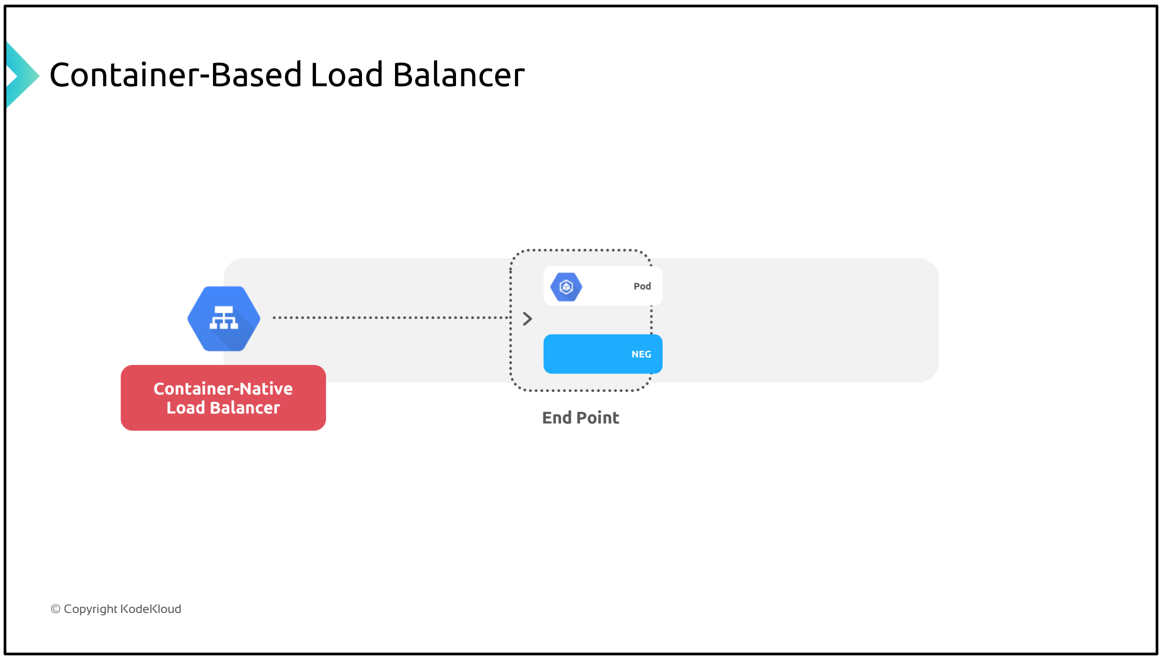
This means that when the traffic from source comes to the internal Application load balancer, it distributes that traffic to the backend services as a new connection without any information of the source to those destination services.



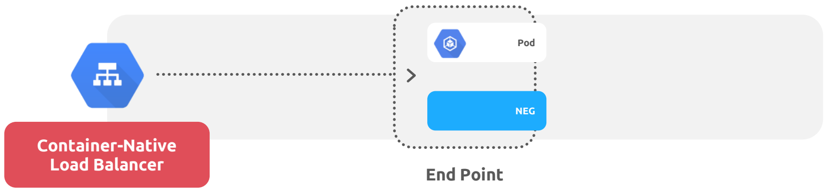
Internal Load Balancer

- Forwarding Rules
- Target Proxies
- URL Maps
- Backend Services
- Health Checks

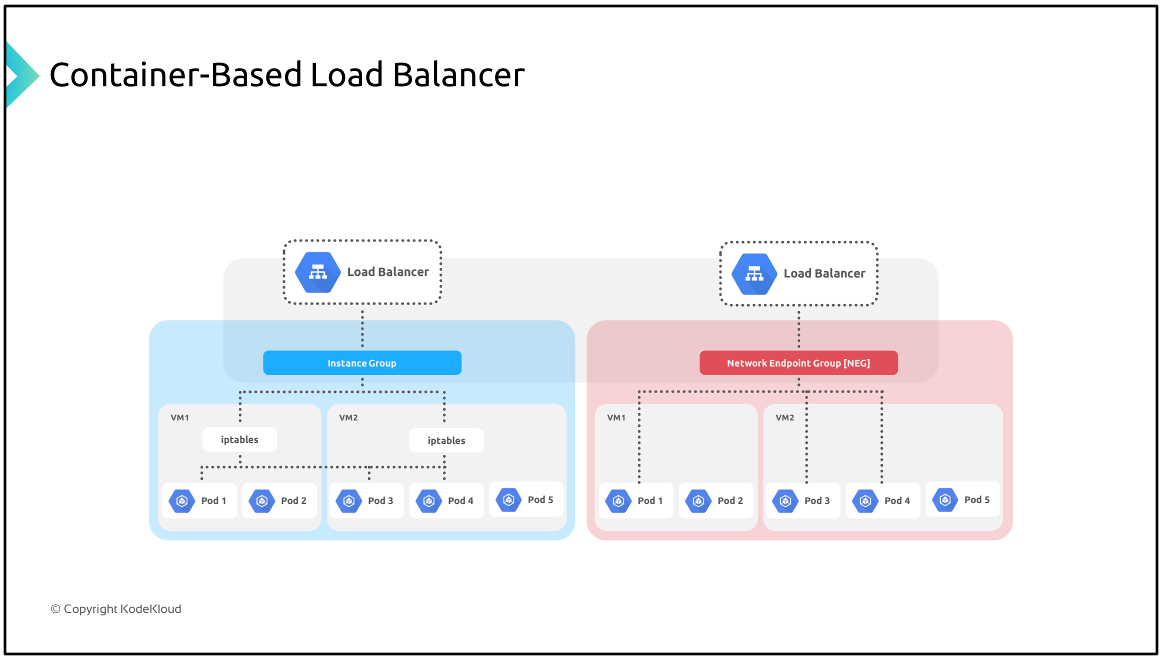
It's important to note that when GKE creates an external or internal Application Load Balancer through an Ingress object, any manual configuration changes made to the load balancer's objects (such as forwarding rules, target proxies, URL maps, backend services, and health checks) outside of GKE may be overwritten during updates or upgrades. To manage load balancers outside of GKE, it's recommended to use container-native load balancing with a group of pods endpoints known as Network Endpoint Groups (NEGs).



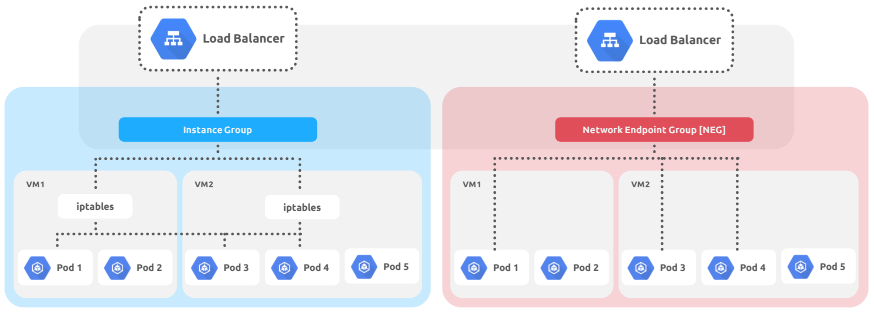
Container-Based Load Balancer



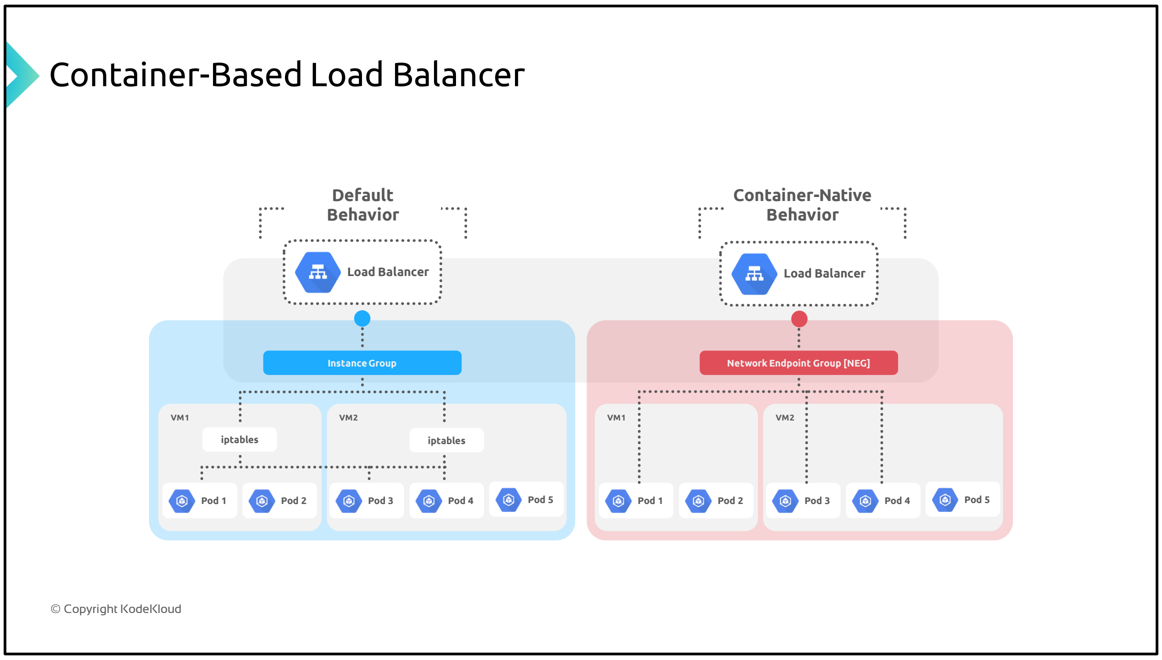
Container-native load balancing is a GKE-specific feature for load balancing that directly routes traffic to the endpoints of Pods using Network Endpoint Groups, rather than to the IP addresses of nodes that the pods are running on.



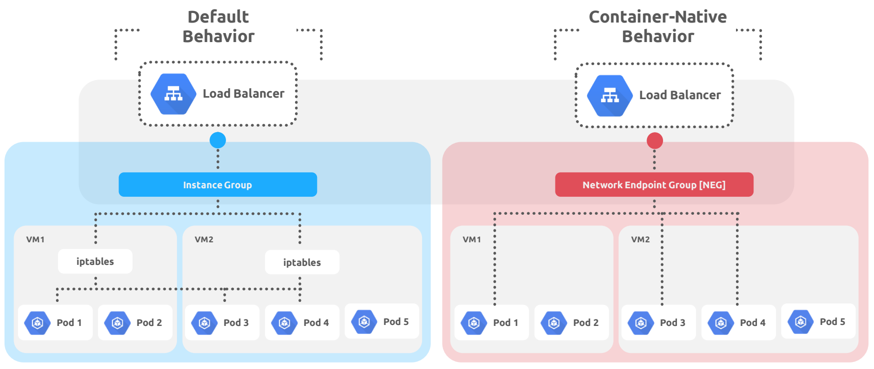
Container-Based Load Balancer



Container Native Load Balancer provides more efficient and stable traffic routing compared to traditional load balancing methods.

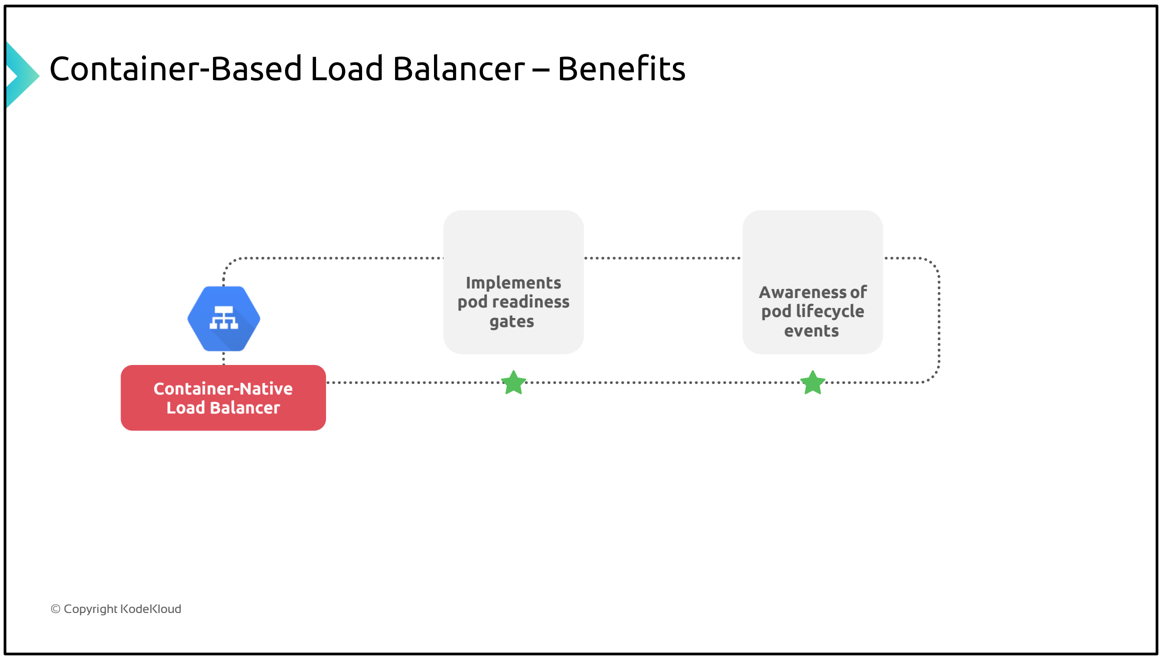


Container-Based Load Balancer

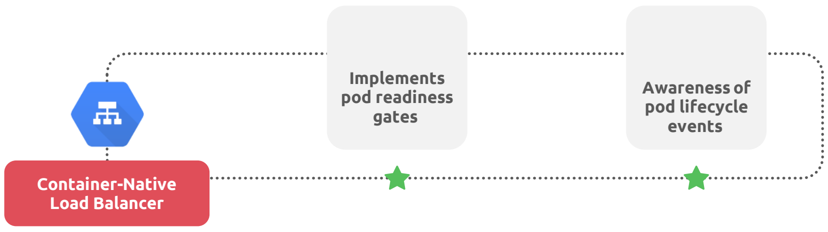


In traditional load balancing using Instance Groups, Compute Engine load balancers send traffic to the IP addresses of virtual machines (VMs) as backends. When running containers on VMs, this introduces several limitations. There are two hops of load balancing involved: first from the load balancer to the VM NodePort, and then through kube-proxy routing to the Pod IP, which may reside on a different VM. This additional hopping adds latency, complexity, and can result in suboptimal traffic balancing. The Compute Engine load balancer has no direct visibility to Pods.

Container-native load balancing overcomes the limitations of **Instance Group-based load balancing**. It routes traffic directly from the load balancer to the Pod IP without traversing the VM IP and kube-proxy networking.

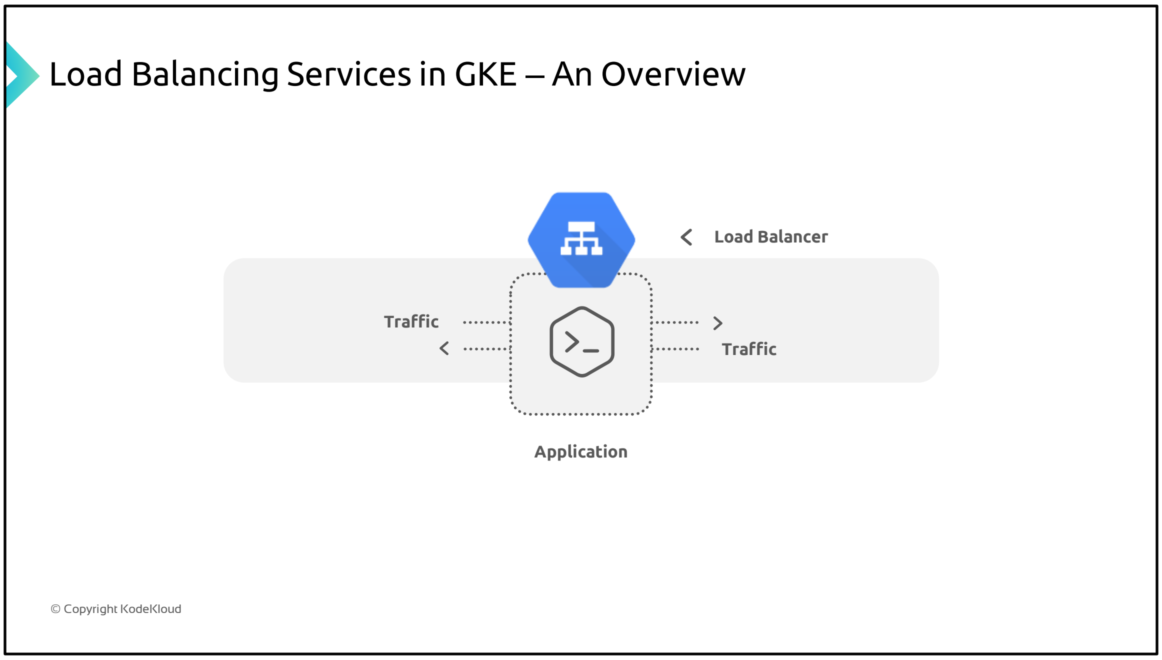


Container-Based Load Balancer – Benefits

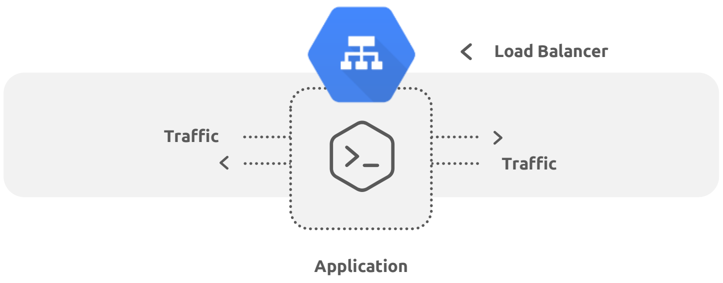

Container-Native Load Balancer
Implements pod readiness gates
Awareness of pod lifecycle events

With NEGs, **Pod readiness checks are implemented** to determine Pod health from the perspective of the load balancer, enabling better traffic stability. This way the load balancer becomes aware of **Pod lifecycle events** resulting in more performant and stable networking.

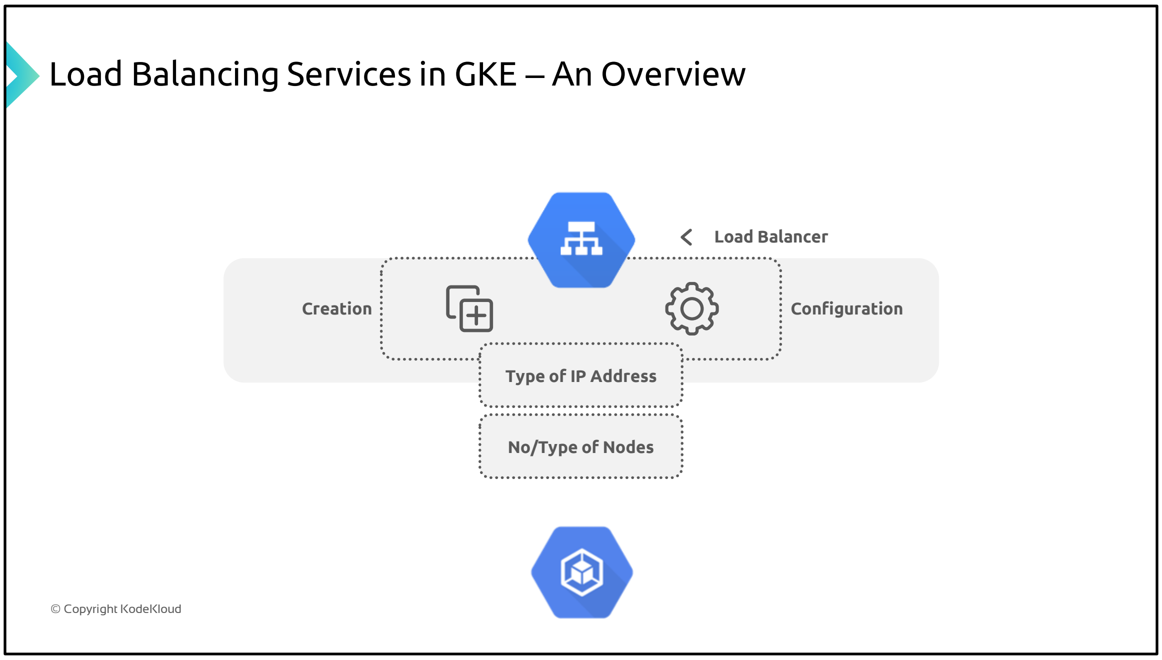
Load Balancing GKE Traffic – Service



Load Balancing Services in GKE – An Overview



In Google Kubernetes Engine (GKE), a LoadBalancer Service is a Kubernetes resource that allows you to expose your application externally and distribute incoming traffic across a set of backend pods.



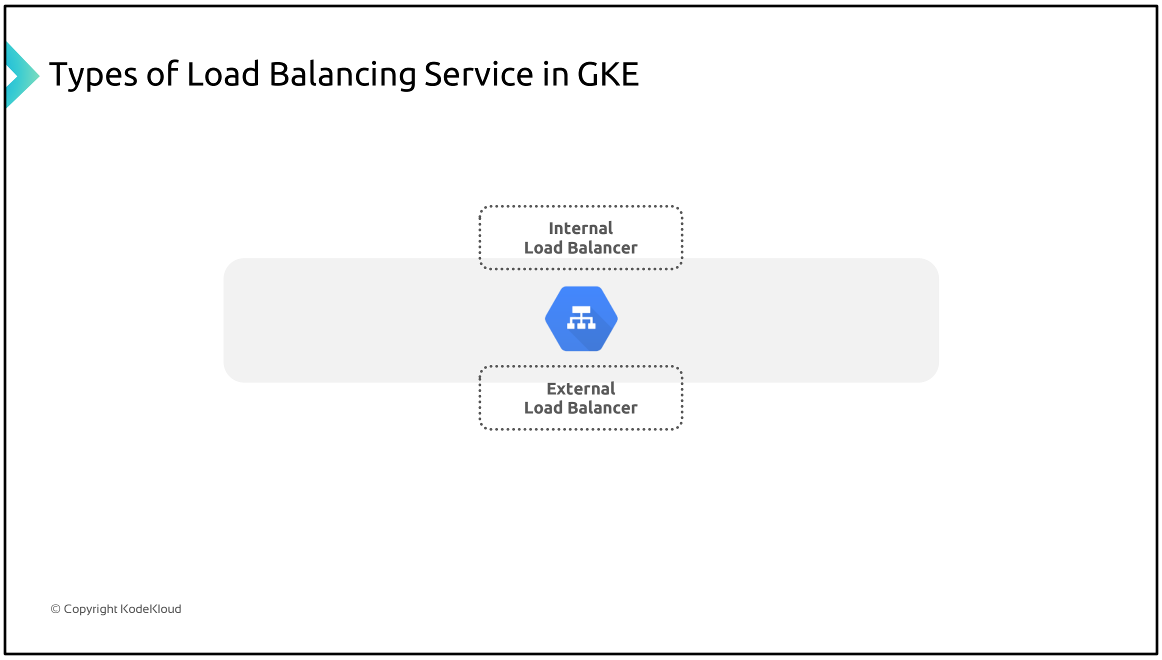
Load Balancing Services in GKE – An Overview

GKE manages the creation and configuration of Google Cloud load balancers based on the parameters specified in the LoadBalancer Service manifest.

When you create a LoadBalancer Service, GKE configures a Google Cloud pass-through load balancer whose characteristics depend on parameters of your Service manifest. For example, you can specify the type of IP address for the load balancer (internal or external), the number and type of nodes the load balancer supports, and whether to enable GKE subsetting.

https://kodekloud.com/

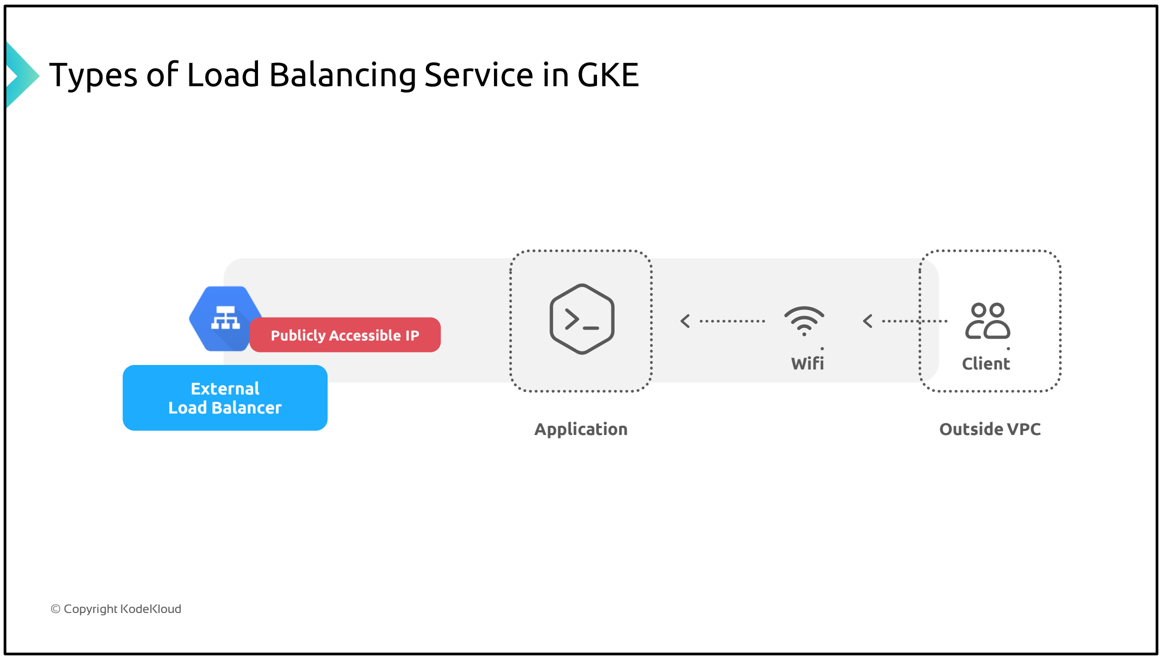
https://kodekloud.com/courses/gke-google-kubernetes-engine/



Types of Load Balancing Service in GKE

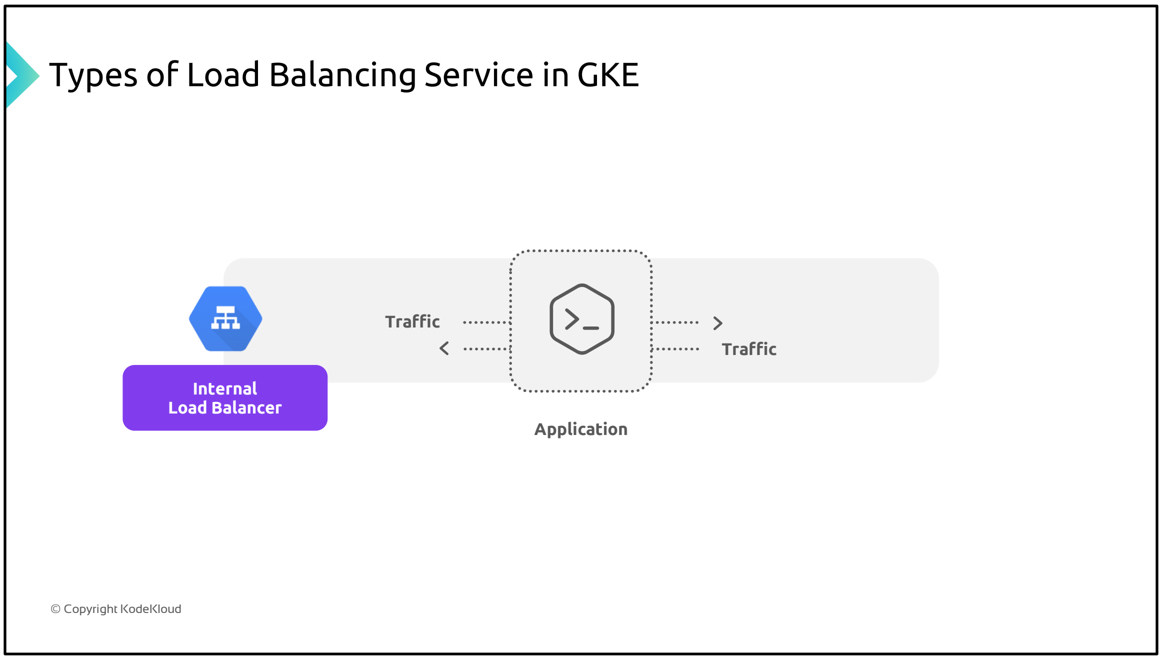
Internal
Load Balancer
External
Load Balancer

When creating a LoadBalancer Service in GKE, you have the option to choose between an internal load balancer and an external load balancer based on the location of your clients and their accessibility.



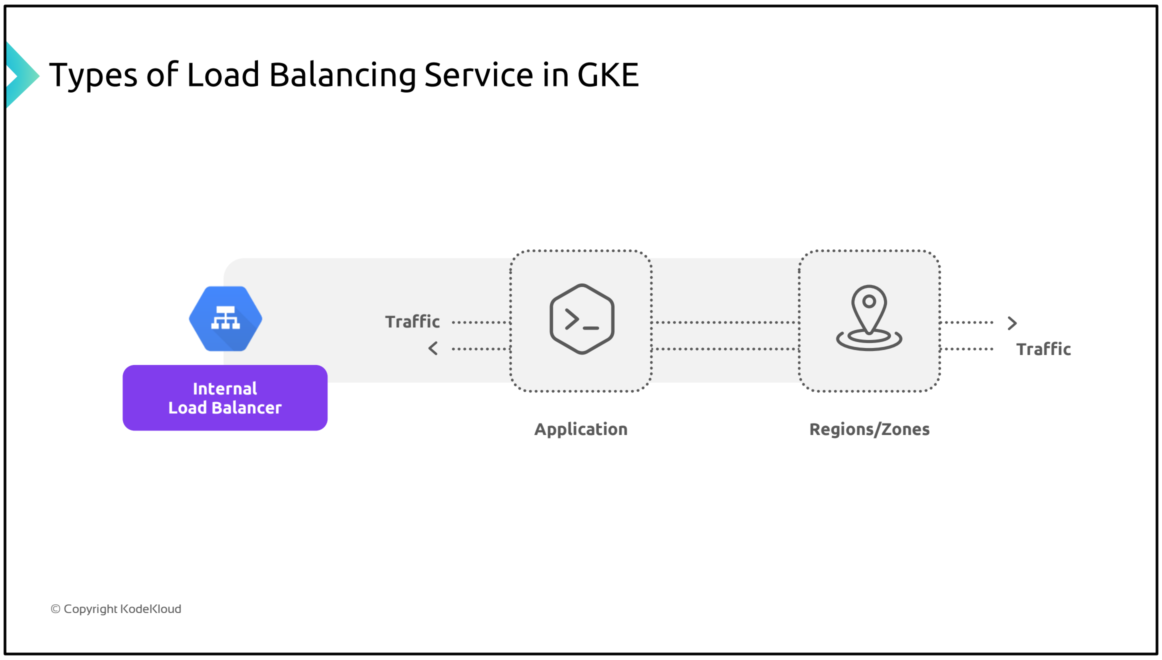
Types of Load Balancing Service in GKE

External LoadBalancer: This type of Service creates an external load balancer with a publicly accessible IP address. It routes traffic from external clients to the Pods in your GKE cluster. It is suitable for applications that need to be accessed from the internet. Use an external load balancer if your clients are located outside of your VPC network and need to access the Service from the internet or other networks.



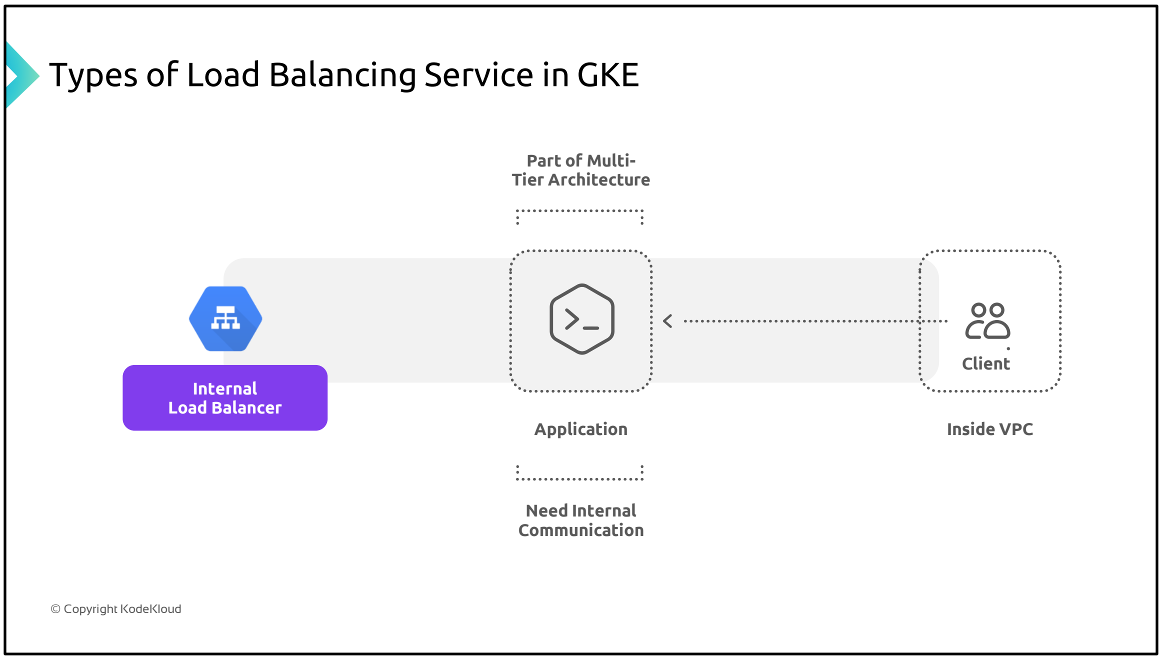
Types of Load Balancing Service in GKE

Internal LoadBalancer: This type of Service creates an internal load balancer with an IP address from the internal network range. It is used to route traffic within the VPC network OR...



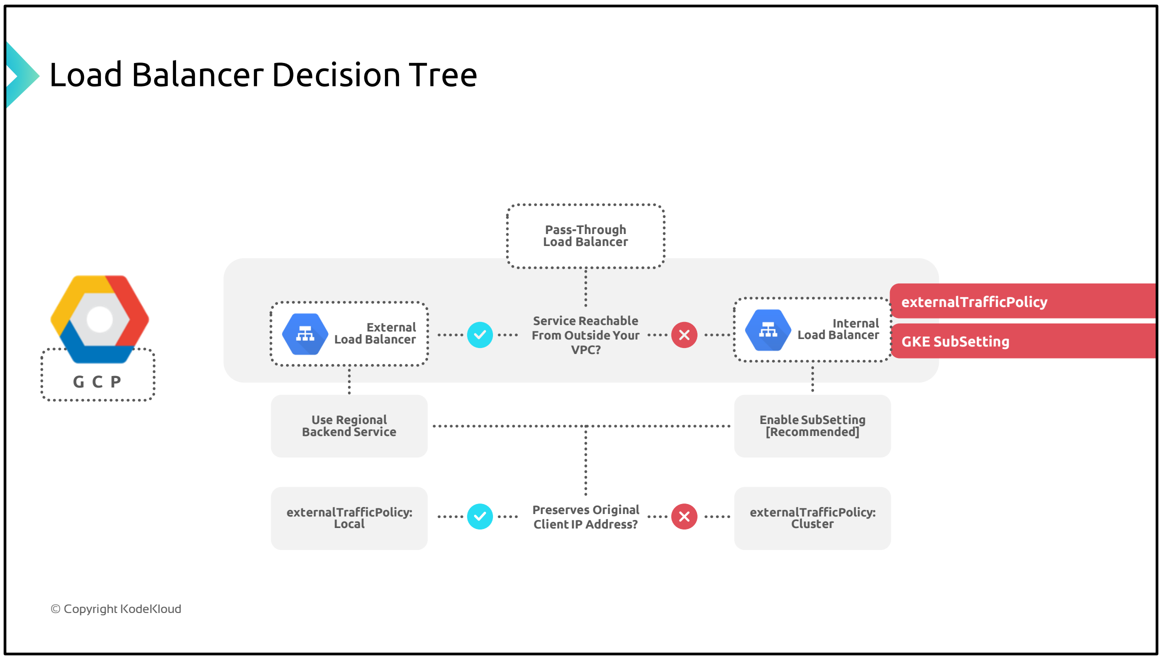
Types of Load Balancing Service in GKE

Internal LoadBalancer:
…across different regions or zones in a private manner.



Types of Load Balancing Service in GKE

Internal LoadBalancer:
It is suitable for applications that need internal communication or are part of a multi-tier architecture. Use an internal load balancer if your clients are located within the same VPC network or in a network connected to the cluster's VPC network.



Load Balancer Decision Tree

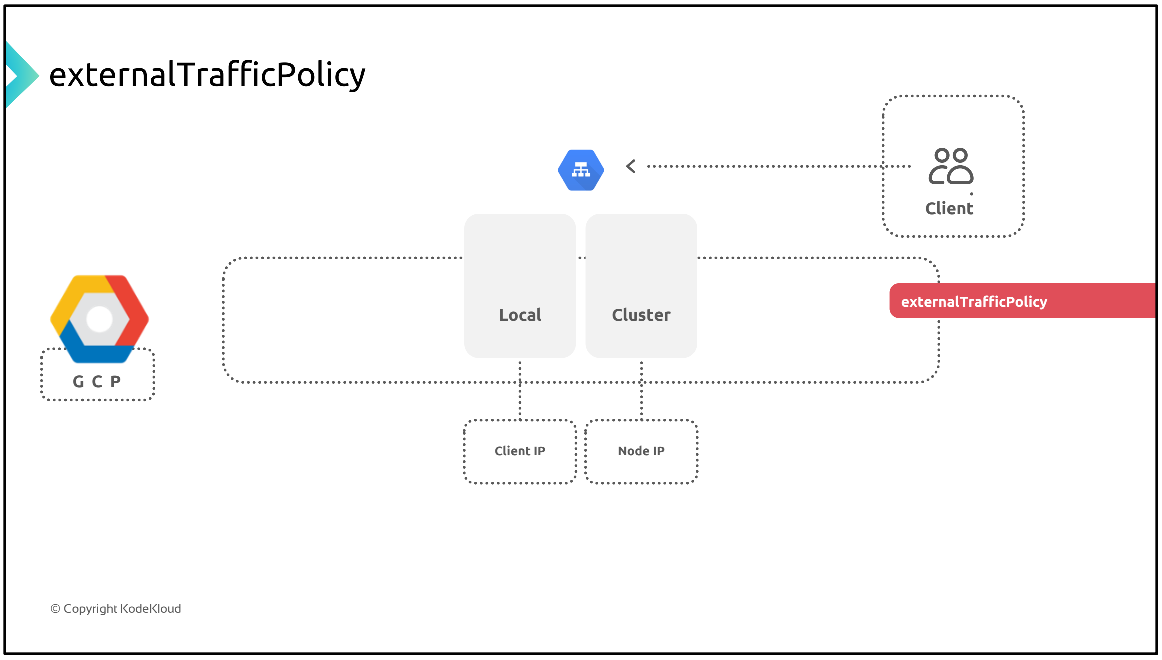
Pass-Through Load Balancer
G C P
External Load Balancer
Service Reachable From Outside Your VPC?
Internal Load Balancer
externalTrafficPolicy
GKE SubSetting
Use Regional Backend Service
Enable SubSetting [Recommended]
externalTrafficPolicy: Local
Preserves Original Client IP Address?
externalTrafficPolicy: Cluster

Google provides a decision tree to help you choose the appropriate LoadBalancer Service configuration for your network architecture requirements. You should take into account factors such as the type of IP address (external or internal) and the number and type of nodes supported by the LoadBalancer while deciding which type of LoadBalancer suits your requirements.

So, if you need your service to be reachable from external traffic that is outside the VPC network that hosts the service, choose External load balancer; however, if the incoming traffic is from within the same network, then Internal traffic would be the right choice.

The configuration of the load balancer is determined by the parameters specified in the LoadBalancer Service manifest. These parameters include:

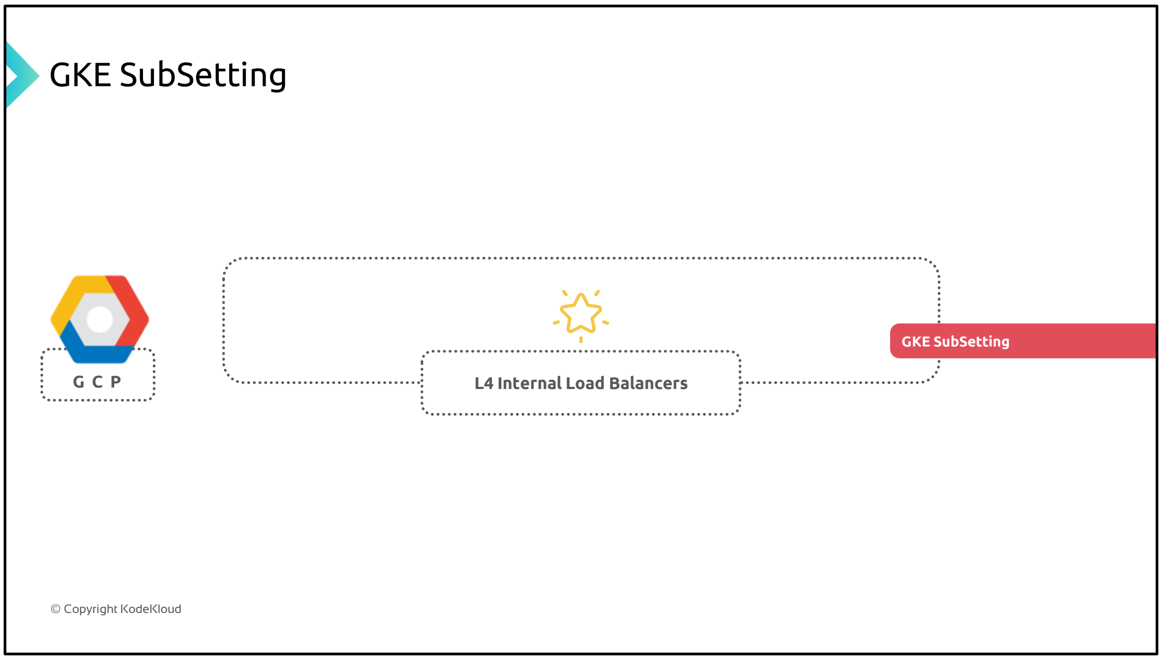
- externalTrafficPolicy:
- GKE Subsetting (for L4 internal load balancers):



The externalTrafficPolicy setting determines how the load balancer forwards incoming traffic to the backend nodes with ready and serving pods. It can be set to "Cluster", which is the default setting or "Local".

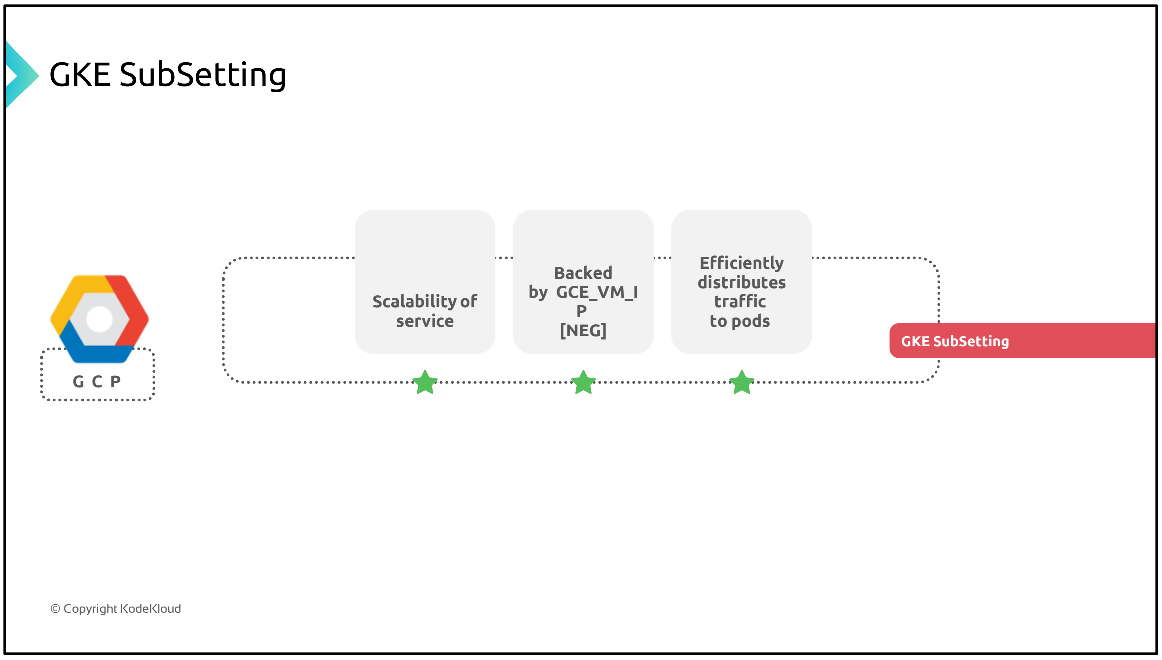
When it is set to "Cluster", the load balancer does not preserve the original client IP addresses. The node changes the source IP address of load-balanced packets to match the IP address of the node which received it from the load balancer. The node routes packets to any serving pod. The serving pod might or might not be on the same node.

Whereas when it's set to "Local", the load balancer preserves the original client IP addresses. The traffic is routed to a serving pod running on the node which received the packet from the load balancer. That node sends response packets to the original client using Direct Server Return.



GKE SubSetting

GKE subsetting is an optional but recommended feature for L4 internal load balancers. It allows you to reduce the number of instances the load balancer sends traffic to based on certain criteria, such as labels or zones. This helps in optimising network traffic and improving performance.



GKE SubSetting

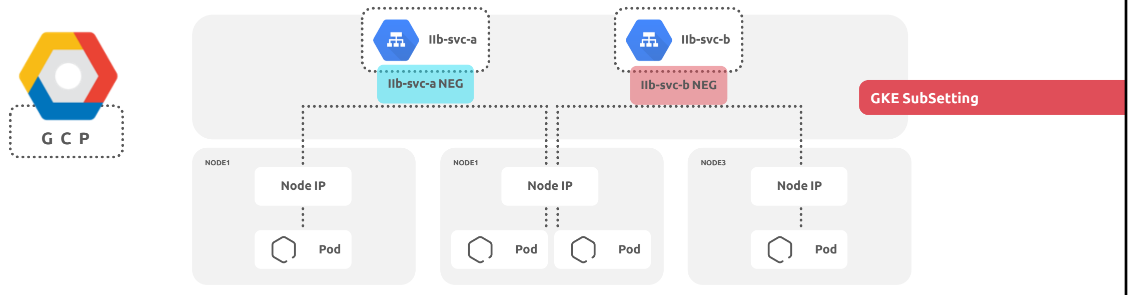
G C P
Scalability of service
Backed by GCE_VM_IP [NEG]
Efficiently distributes traffic to pods
GKE SubSetting

Without GKE subsetting, each internal LoadBalancer Service is backed by an instance group. This means that all of the pods for the service must be running on nodes that are members of the same instance group. If you have **a large cluster with many nodes, this can limit the scalability of the Service**.

With GKE subsetting, each internal **LoadBalancer Service is backed by a GCE_VM_IP** network endpoint group (NEG). An NEG is a collection of IP addresses that can be used as backends for a load balancer. When GKE subsetting is enabled, GKE creates one NEG per compute zone per internal LoadBalancer Service. The member endpoints in the NEG are the IP addresses of nodes that have at least one of the service's serving pods.

The nodes can be members of more than one load-balanced NEG. This allows GKE to **distribute traffic to pods more efficiently**, even in large clusters.

GKE SubSetting



In this example, there are two services in a zonal cluster with three nodes. With GKE subsetting enabled on the cluster, GKE creates one GCE_VM_IP network endpoint group (NEG) for each service as the backend for that particular service. Endpoints in each NEG are the nodes with the serving pods for the respective service. This way, each service gets two pods to serve the traffic and manage the load efficiently.