02

Google Kubernetes Engine Networking

Google Kubernetes Engine Networking

01 Kubernetes networking

02 Kubernetes Services

03 Ingress

04 Container-native load balancing

05 Network policies in GKE

Google Kubernetes Engine Networking

01 Kubernetes networking

02 Kubernetes Services

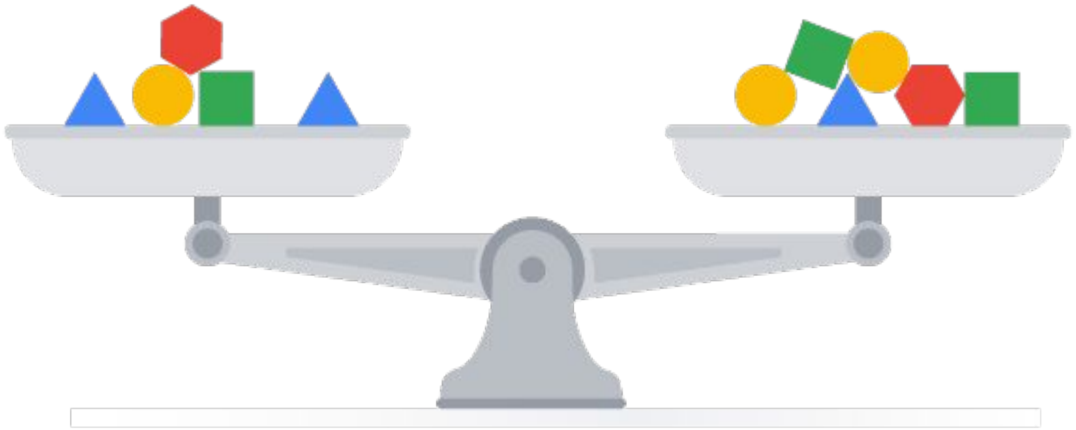
03 Ingress

04 Container-native load balancing

05 Network policies in GKE

Pod networking

Pods, containers, and nodes all communicate using IP addresses and ports.

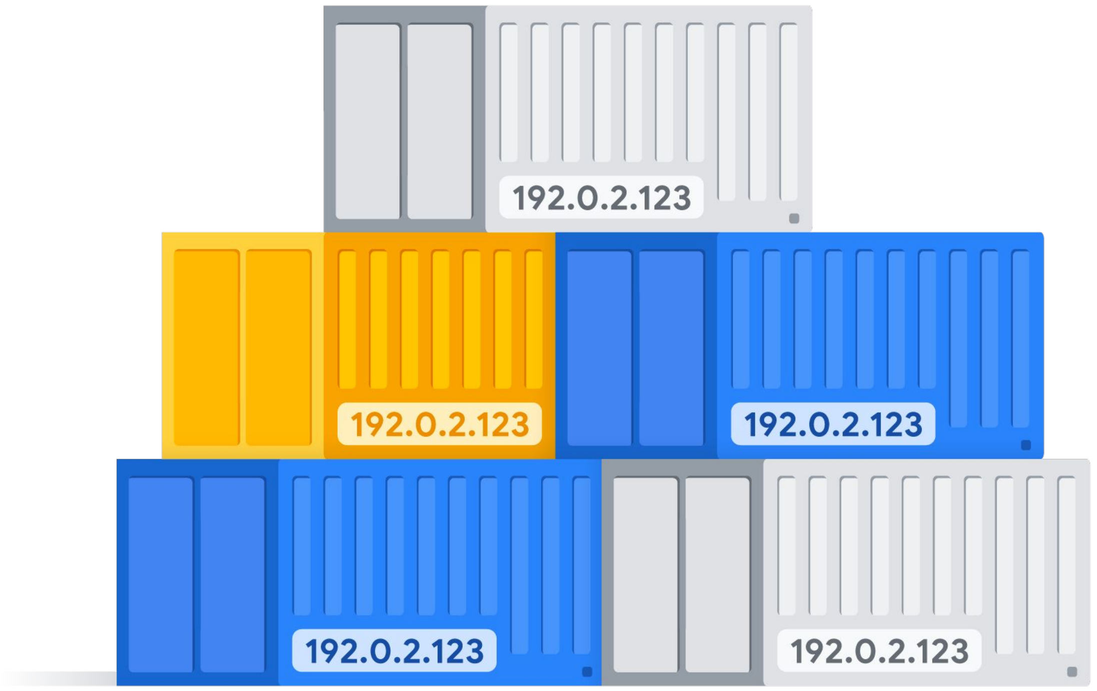


Kubernetes provides different types of load balancing to direct traffic to the correct Pods.

Pods share storage and networking

Pod

192.0.2.123



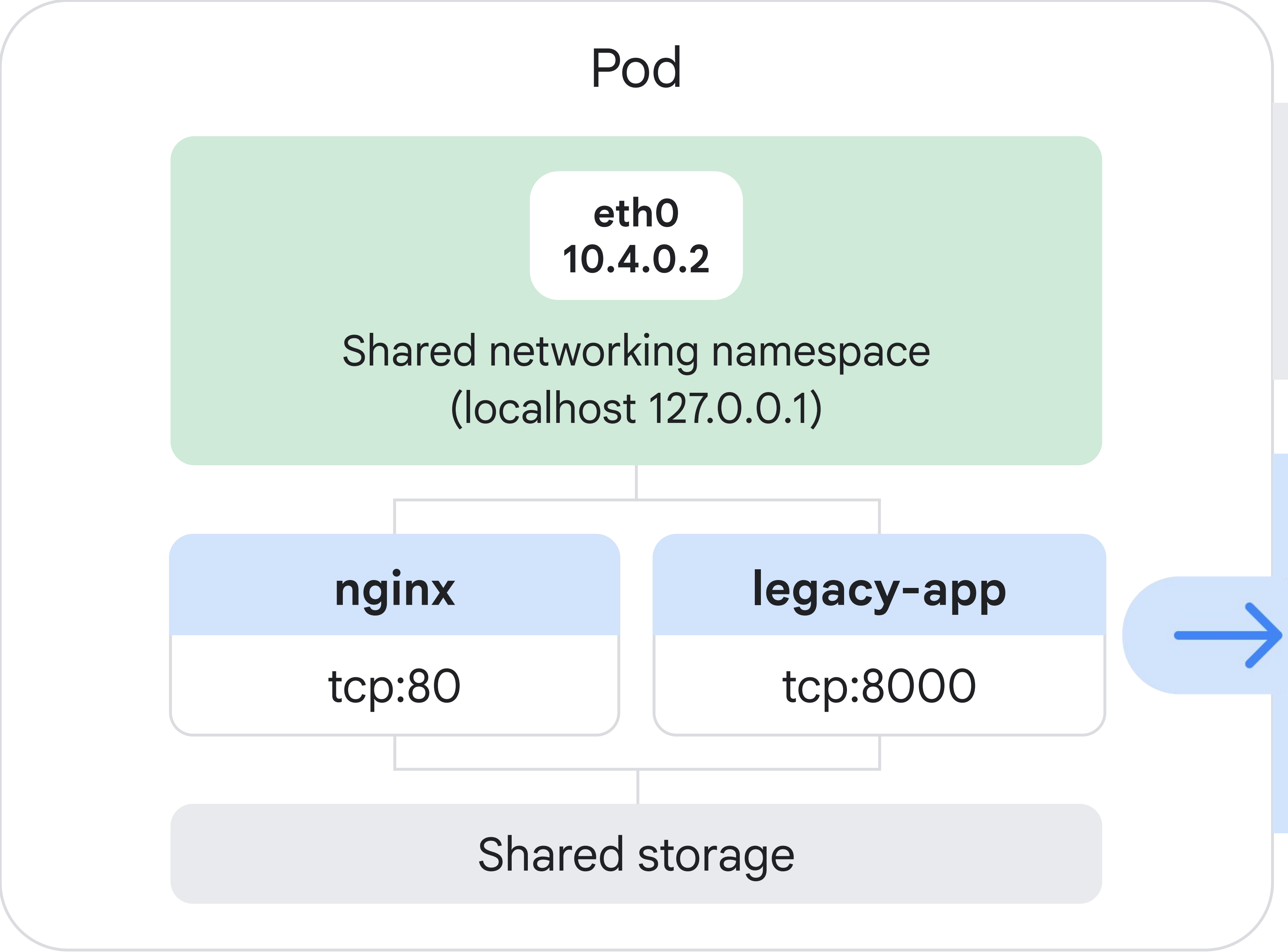
192.0.2.123

Each Pod is assigned a single IP address.

Containers within a Pod share the same network namespace, including that IP address.

Based on the "IP-per-pod" model of Kubernetes.

Namespaces isolate resources within a cluster



Example:
A legacy application using nginx as a reverse-proxy for client access.

Sharing a networking namespace gives the containers the appearance of being on the same machine.

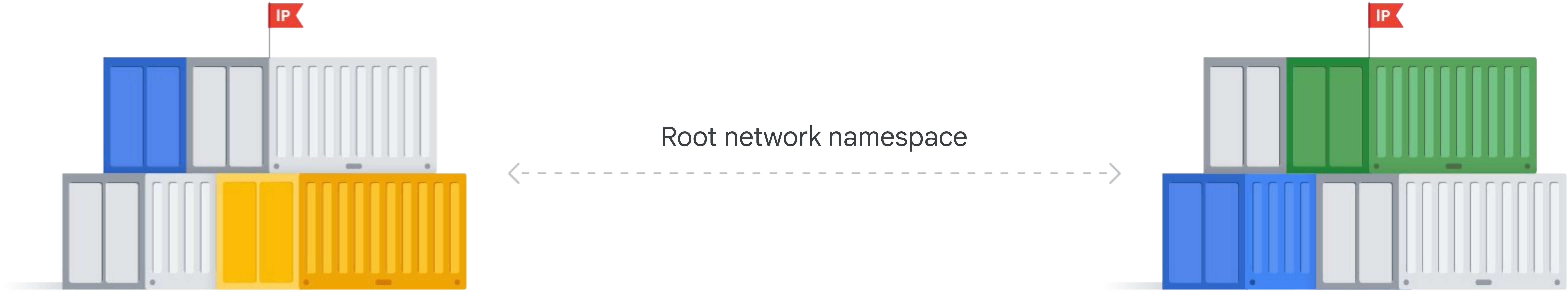
This works well for a single Pod.

How Pods talk to other Pods

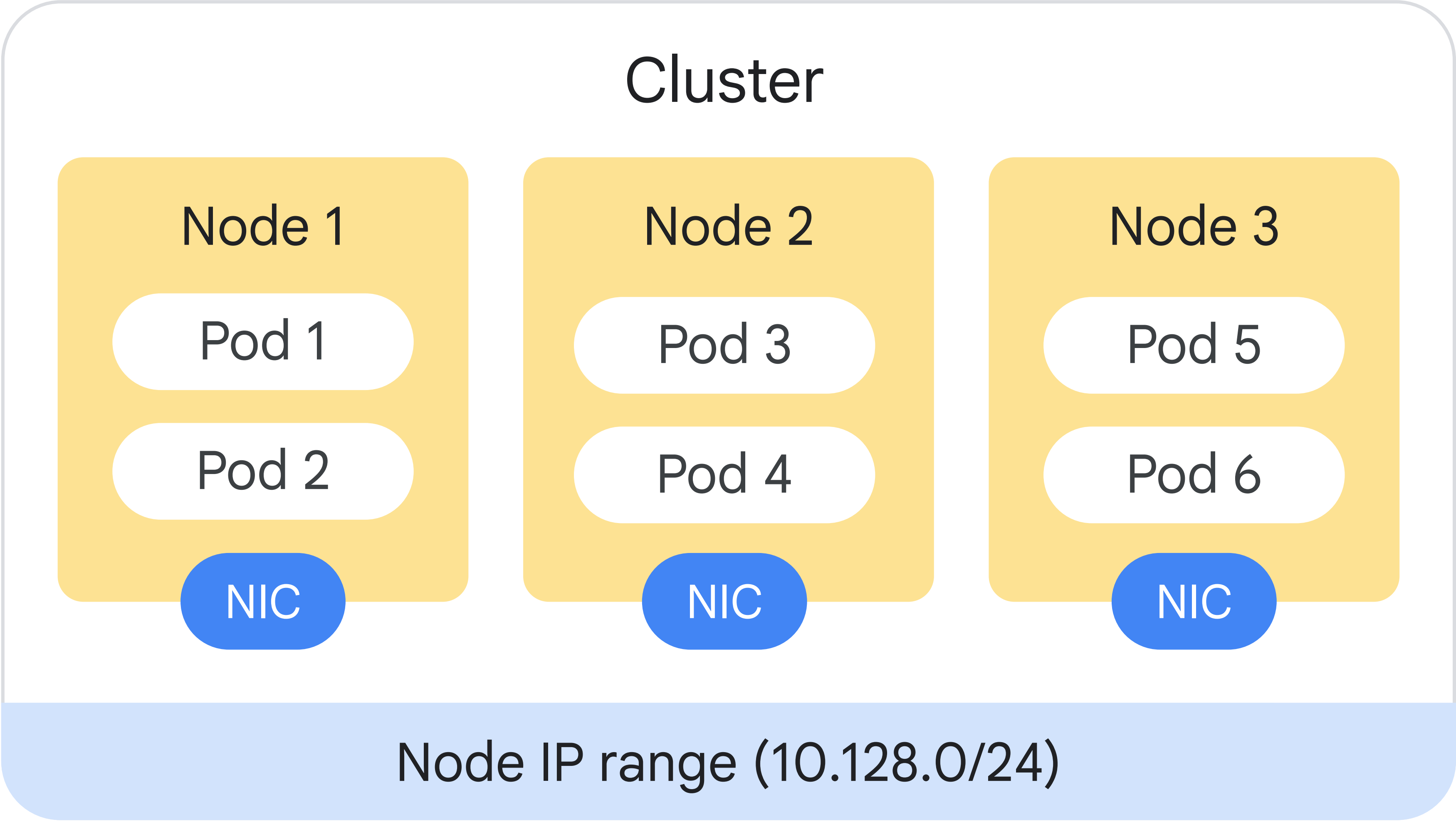
Each Pod has a unique IP address, just like a host on the network.

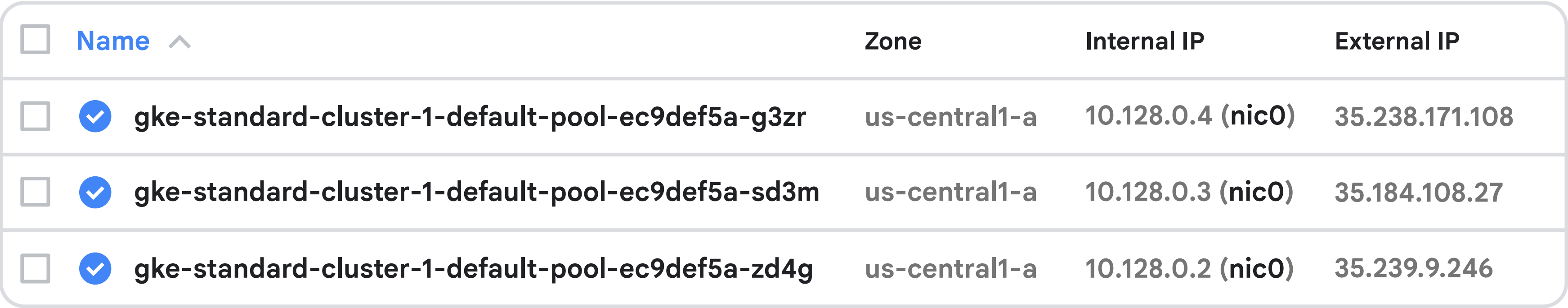
On a node, Pods are connected to each other through the node's root network namespace.

The root network namespace is connected to the node's primary network interface card (NIC).



Nodes source Pod IPs from VPC address ranges





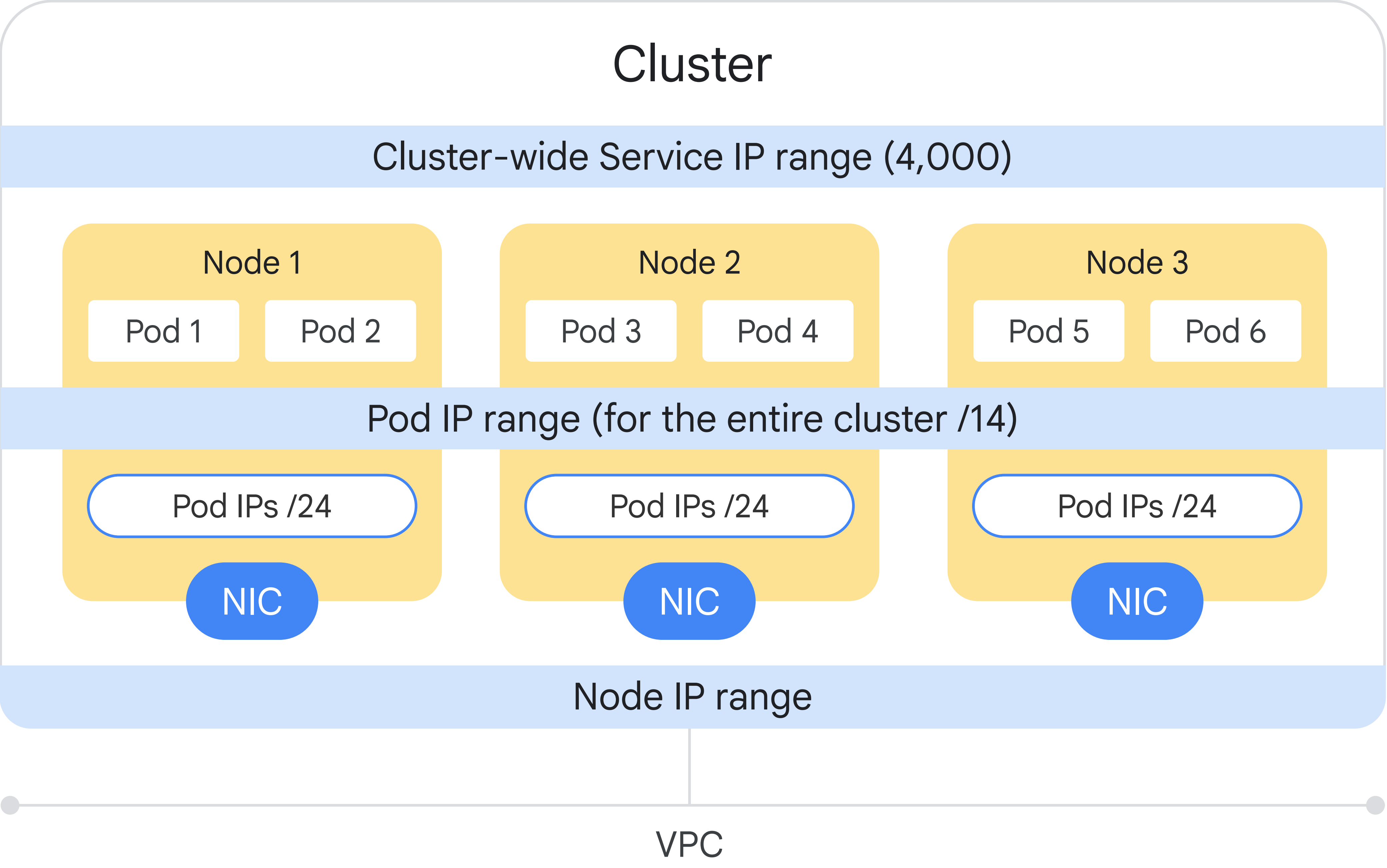
□	Name ^	Zone	Internal IP	External IP
□	✓ gke-standard-cluster-1-default-pool-ec9def5a-g3zr	us-central1-a	10.128.0.4 (nic0)	35.238.171.108
□	✓ gke-standard-cluster-1-default-pool-ec9def5a-sd3m	us-central1-a	10.128.0.3 (nic0)	35.184.108.27
□	✓ gke-standard-cluster-1-default-pool-ec9def5a-zd4g	us-central1-a	10.128.0.2 (nic0)	35.239.9.246

VPCs are logically isolated networks that provide connectivity for resources deployed within Google Cloud.

A VPC can be composed of many different IP subnets in regions around the world.

When you deploy a GKE cluster, you can select a VPC along with a region or zone.

GKE cluster nodes: Managed compute instances

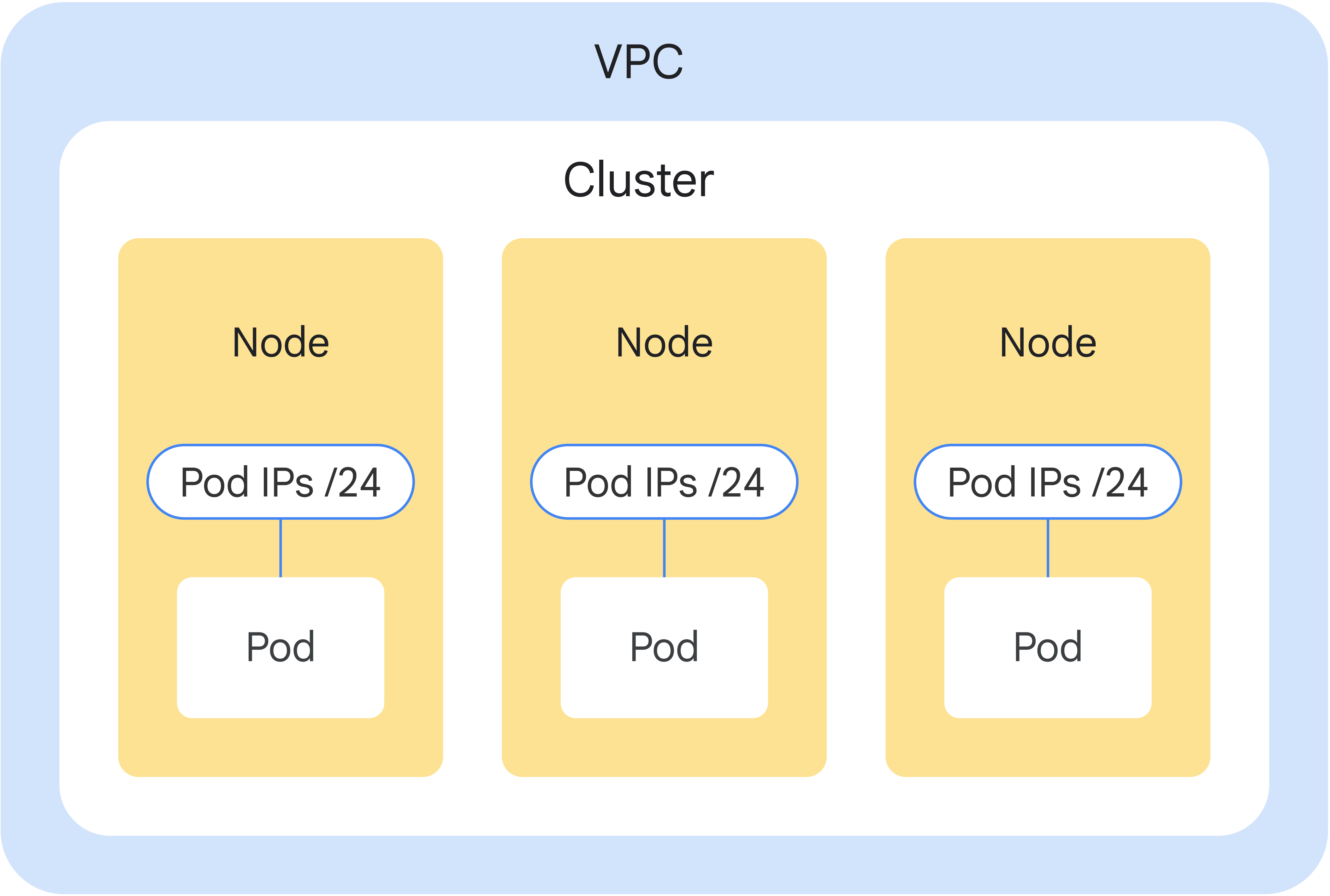


Alias IPs can configure additional secondary IP addresses or IP ranges on Compute Engine VM instances.

VPC-Native GKE clusters automatically create an alias IP range to reserve approximately 4,000 IP addresses for cluster-wide services.

VPC-Native GKE clusters also create a separate alias IP range for your Pods.

IP ranges allow GKE to divide IP space

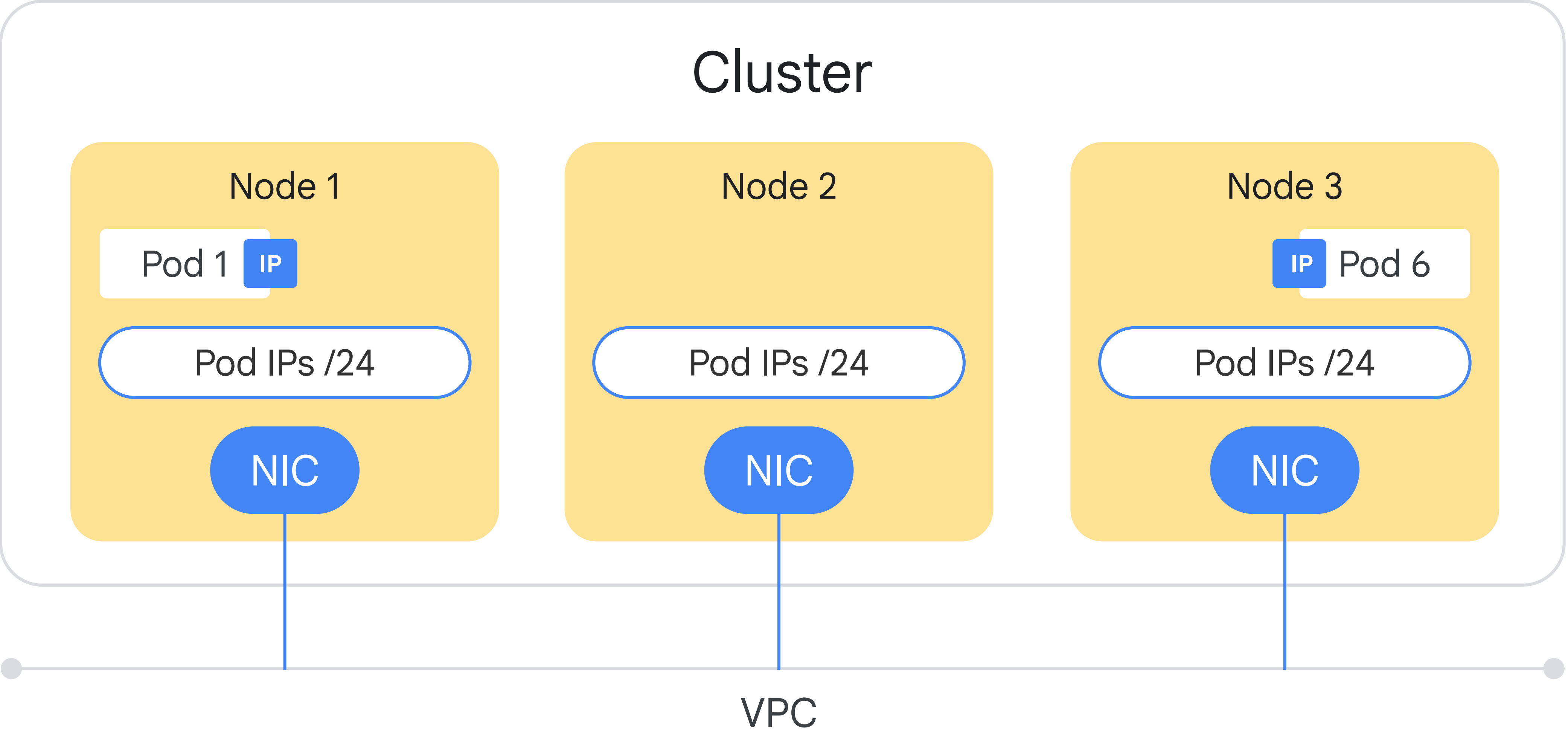


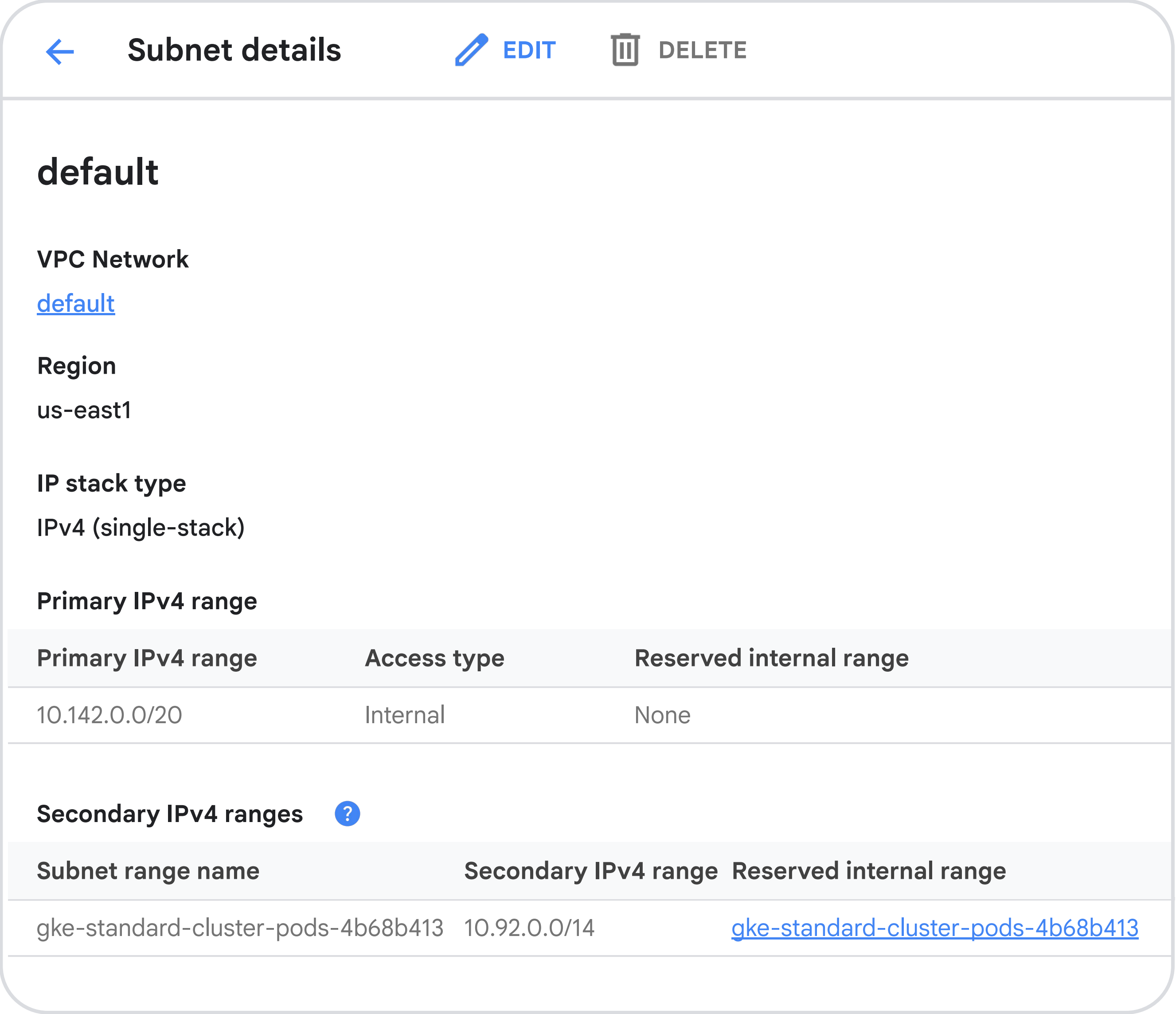
Large default Pod IP range (/14) enables flexible IP distribution across nodes.

Each node receives a smaller IP block (/24) for Pod allocation.

Design supports scaling both nodes and Pods per node.

Nodes assign unique IP addresses to Pods





Subnet details

default

VPC Network
default

Region
us-east1

IP stack type
IPv4 (single-stack)

Primary IPv4 range

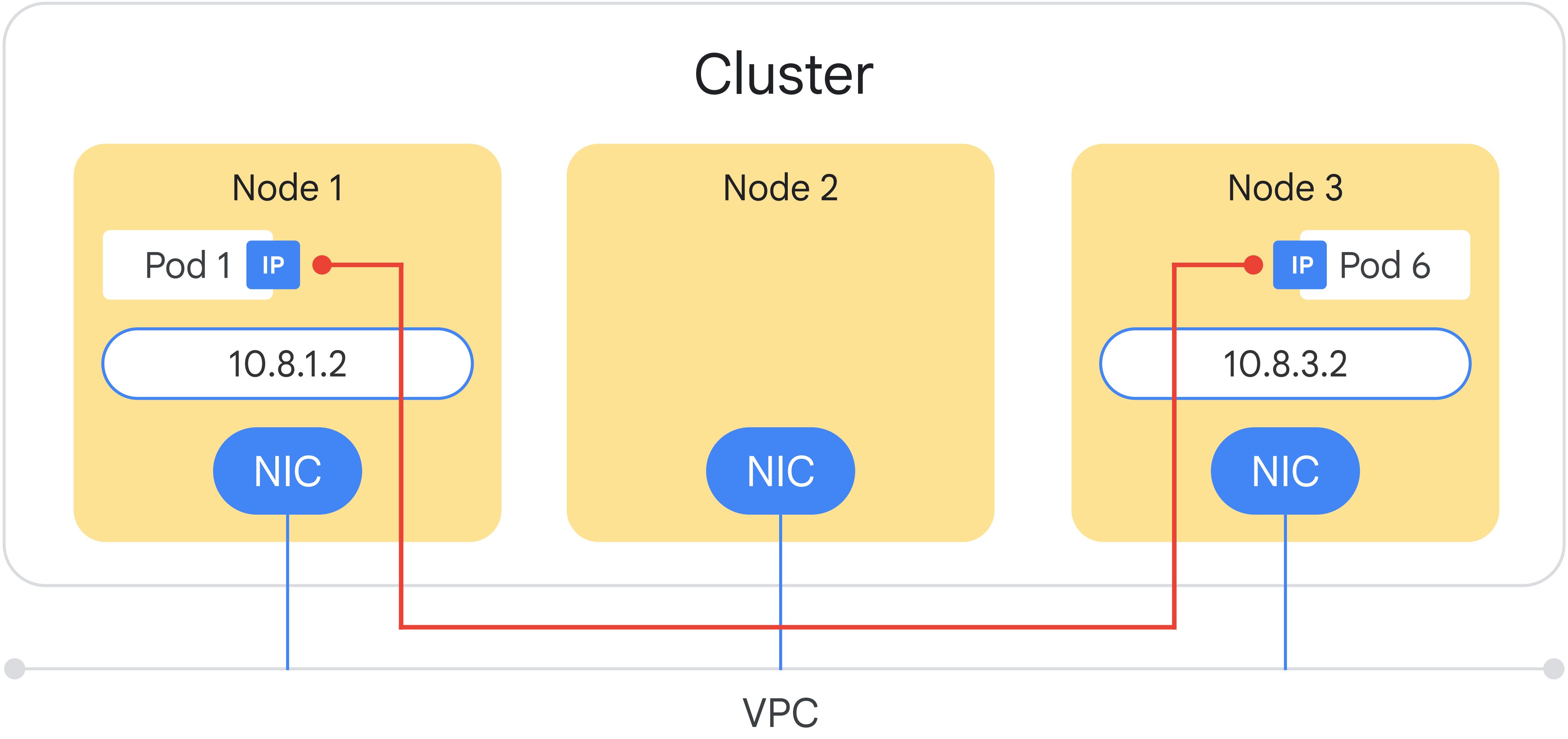
Primary IPv4 range	Access type	Reserved internal range
10.142.0.0/20	Internal	None

Secondary IPv4 ranges ?

Subnet range name	Secondary IPv4 range	Reserved internal range
gke-standard-cluster-pods-4b68b413	10.92.0.0/14	gke-standard-cluster-pods-4b68b413

GKE automatically configures your VPC to recognize this range of IP addresses as an authorized secondary subnet of IP addresses

Nodes maintain separate Pod IP address spaces



Pods can directly connect to each other by using their native IP addresses.

Pod IP addresses are natively routable by VPC Network Peering.

Traffic from clusters is routed or peered inside Google Cloud, but becomes NAT translated at the node IP address if it has to exit Google Cloud.

Google Kubernetes Engine Networking

01 Kubernetes networking

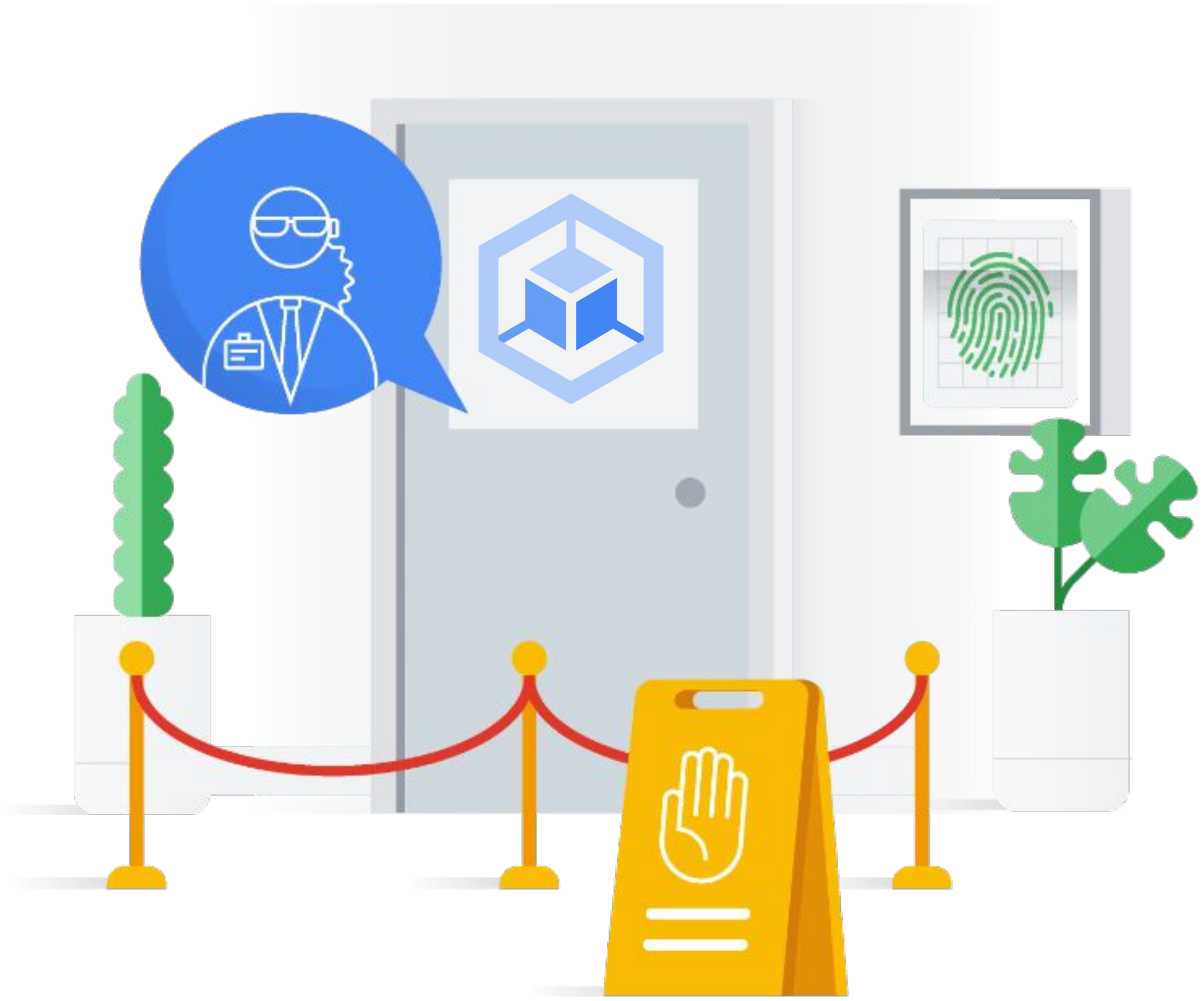
02 Kubernetes Services

03 Ingress

04 Container-native load balancing

05 Network policies in GKE

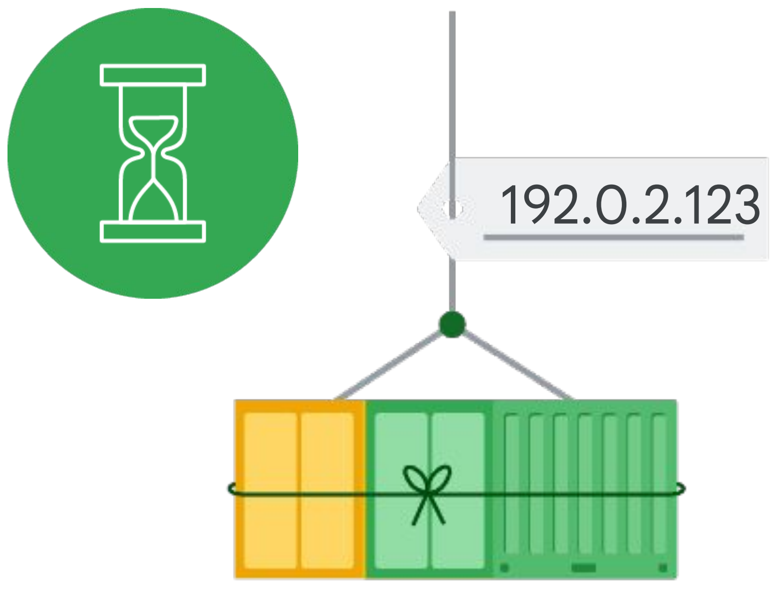
What is a Service?



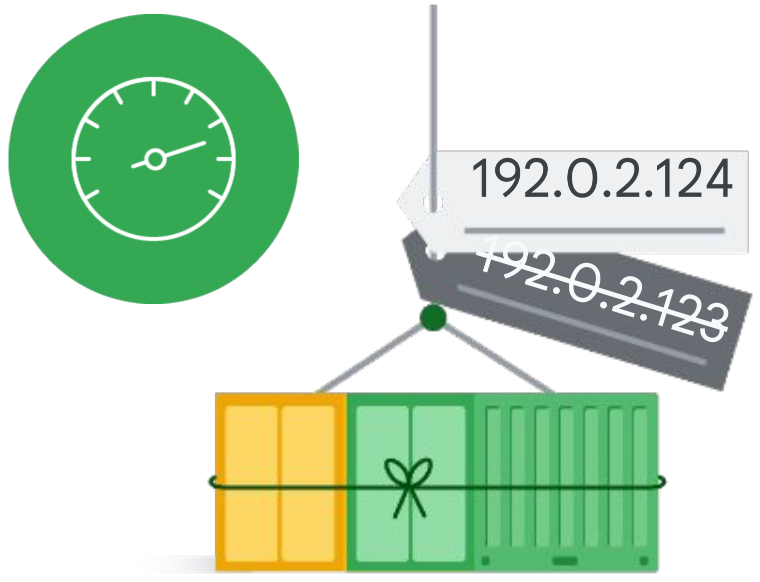
- ✓ It's a logical abstraction that defines a set of Pods and a single IP address for accessing them.
- ✓ It's how the outside world accesses the cluster. Think of it like a GKE doorman or bouncer, keeping out unwanted visitors.
- ✓ It's used to provide a stable IP address and name for a Pod, because these can change frequently.

Pods and virtual machines have different life cycles

Pods are usually terminated and replaced with newer Pods after application updates.

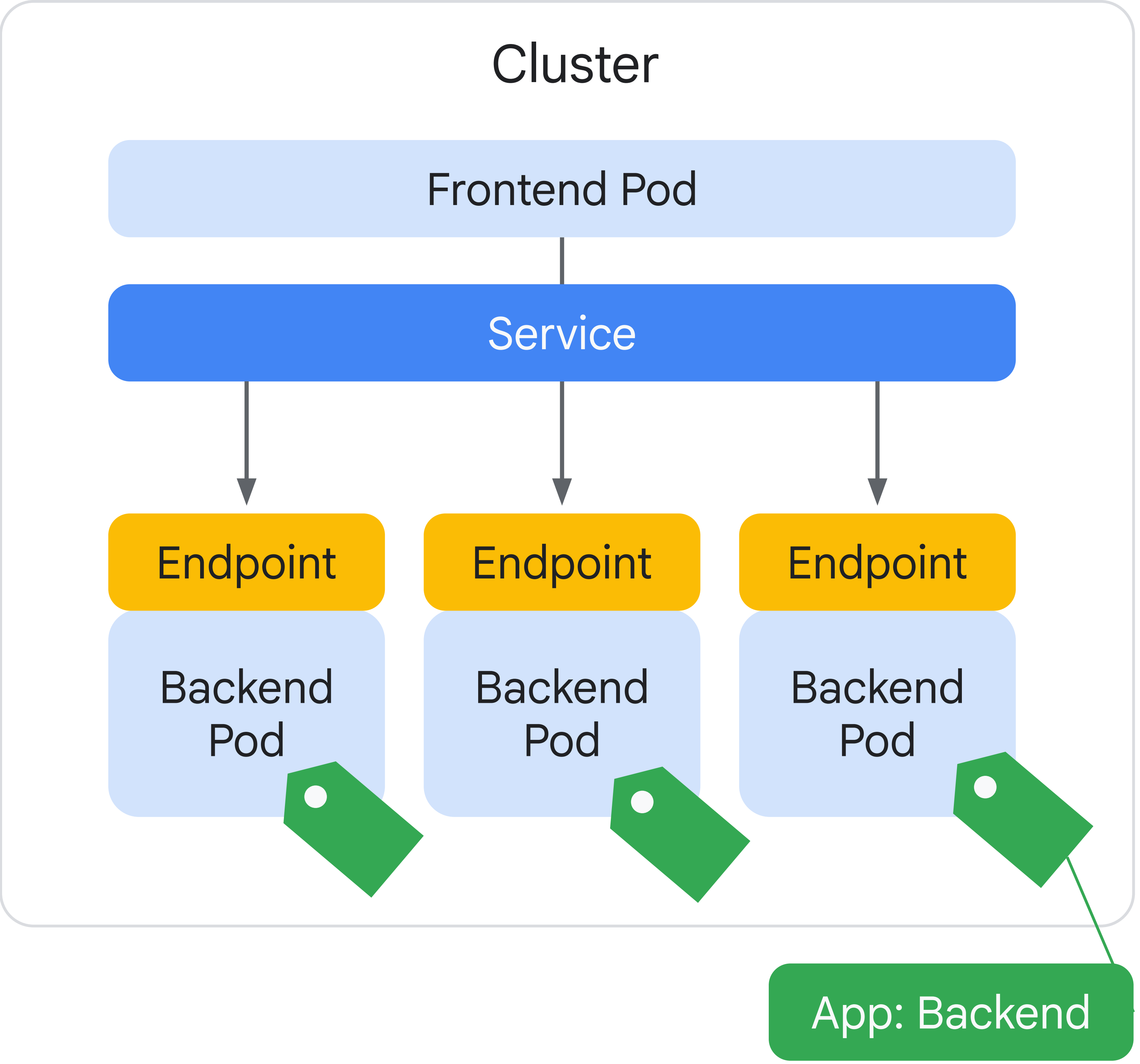


Pod IP addresses are ephemeral, which means that they’re temporary.



If a Pod deployment is rescheduled, the Pod gets assigned a new IP address.

Endpoints



- ✓ Kubernetes services create dynamic IP address collections called **endpoints.**
- ✓ Endpoints link to Pods matching the Service's labels.
- ✓ Services receive a fixed virtual IP address upon creation.
- ✓ Service virtual IPs are durable, unlike Pod IPs.

Ways to search for and locate a Service in GKE

By environment variables

By DNS

By Service type

Locate a Service by environment variables

By environment variables

- ✓ Kubelet adds a set of environment variables for each active Service.
- ✓ The Pod can access the Service by using the environment variables.
- ✗ Not the most robust mechanism for discovery.

  Changes made to a Service *after* Pods have been started *will not* be visible to the Pods that are already running.

An example of environment variables

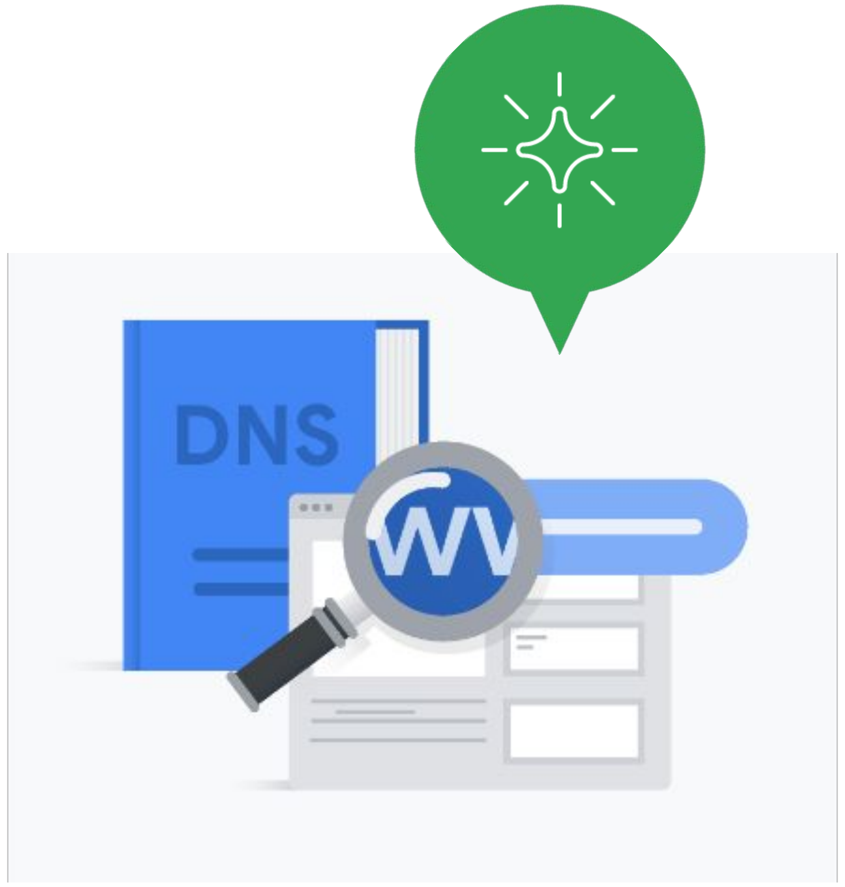
```
DEMO_SERVICE_HOST=10.70.0.11
DEMO_SERVICE_PORT=6379
DEMO_PORT=TCP://10.70/0/11:6379
DEMO_PORT_6379_TCP=tcp://10.70.0.11:6379
DEMO_PORT_6379_TCP_PROTO=tcp
DEMO_PORT_6379_TCP_PORT=6379
DEMO_PORT_6379_TCP_ADDR=10.70.0.11
```

Example of environment variables for a Service named **demo.**

```
demo.my-project
```

```
_http._tcp.demo.my-project
```

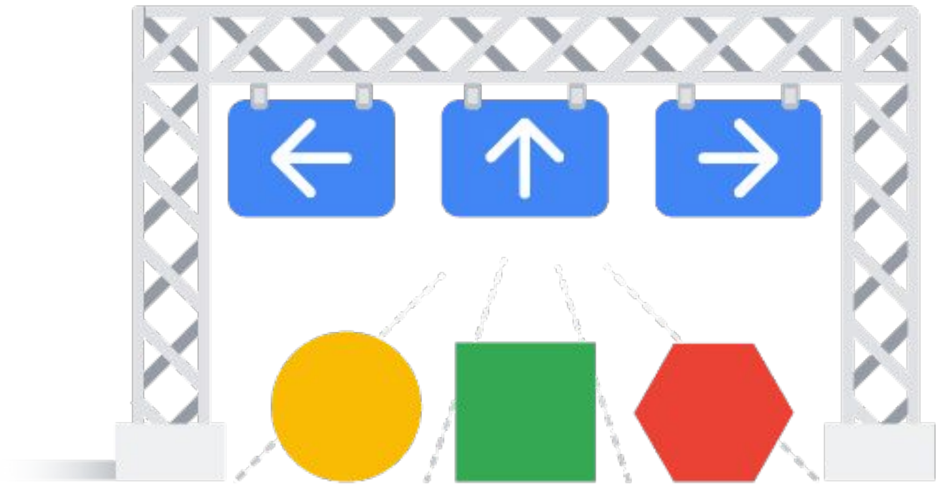
Locate a Service using a DNS server



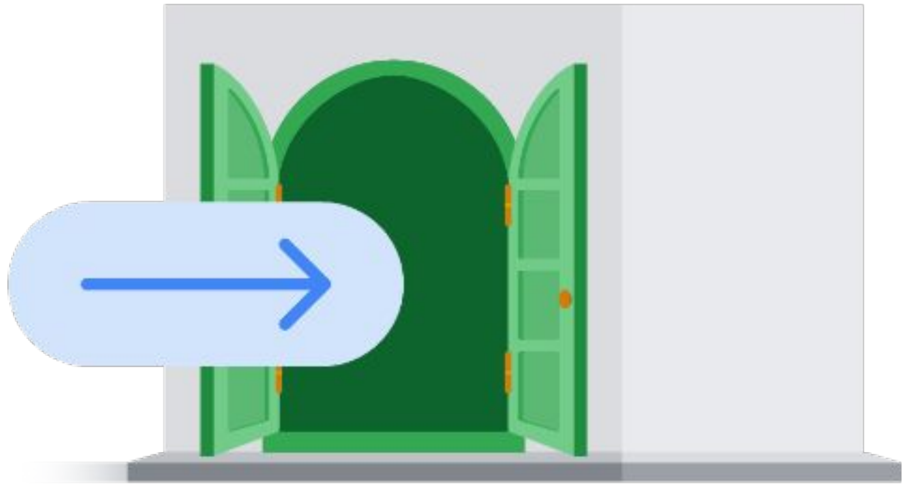
By DNS

- ✓ GKE includes pre-installed DNS.
- ✓ DNS monitors the API server for new Services.
- ✓ Kube-dns auto-generates DNS records for new Services.
- ✓ Client pods' DNS search includes their namespace and the cluster's default domain.

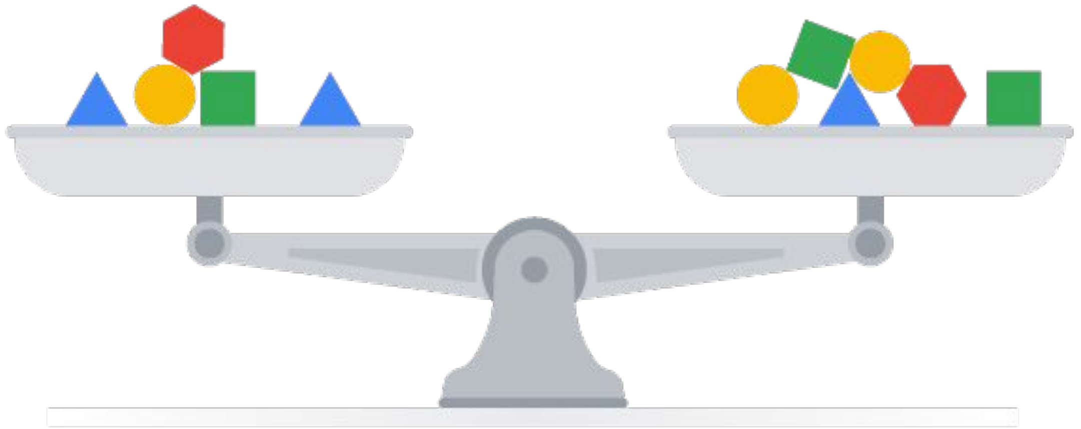
Find a Service by changing the Service type



ClusterIP Service

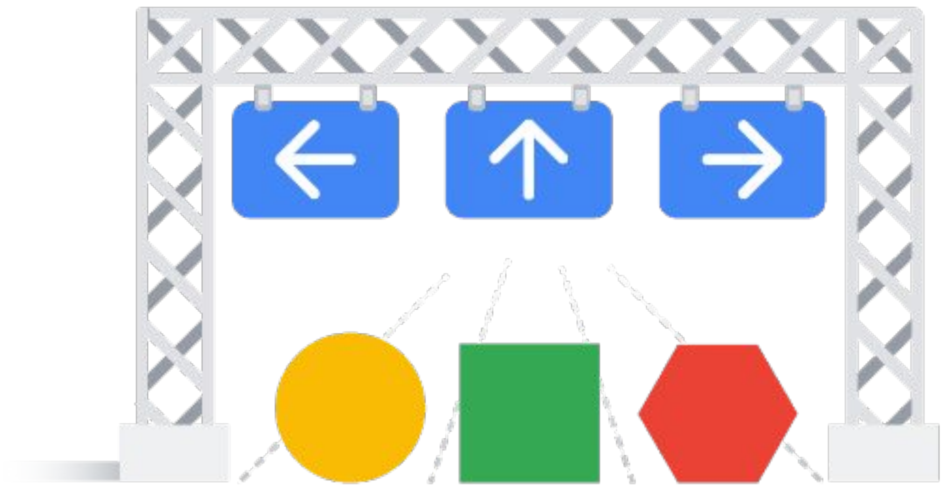


NodePort



LoadBalancer

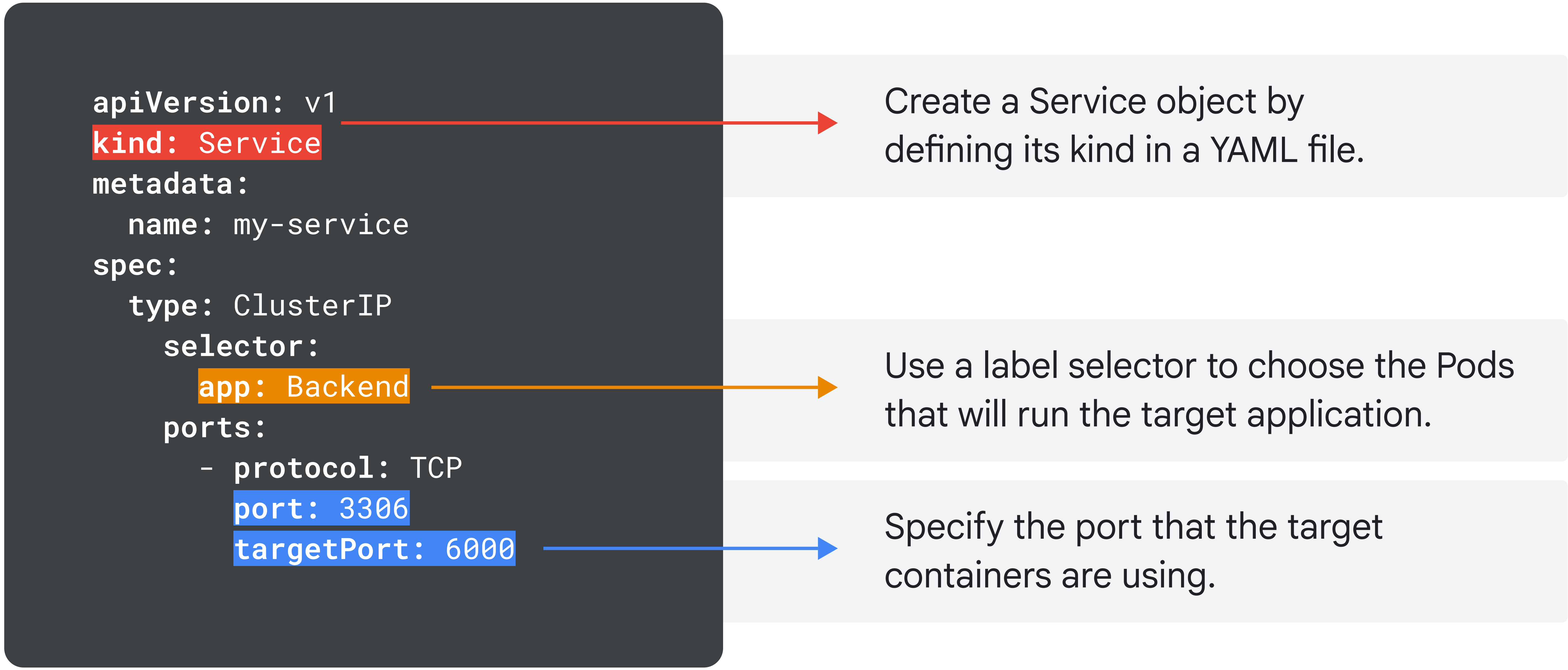
ClusterIP Service



ClusterIP Service

- ✓ Has a static IP address.
- ✓ Operates as a traffic distributor within the cluster.
- ✓ Not accessible by resources outside the cluster.
- ✓ Other Pods use it to communicate with the Service.

Creating a ClusterIP Service



```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
    selector:
      app: Backend
    ports:
      - protocol: TCP
        port: 3306
        targetPort: 6000
```

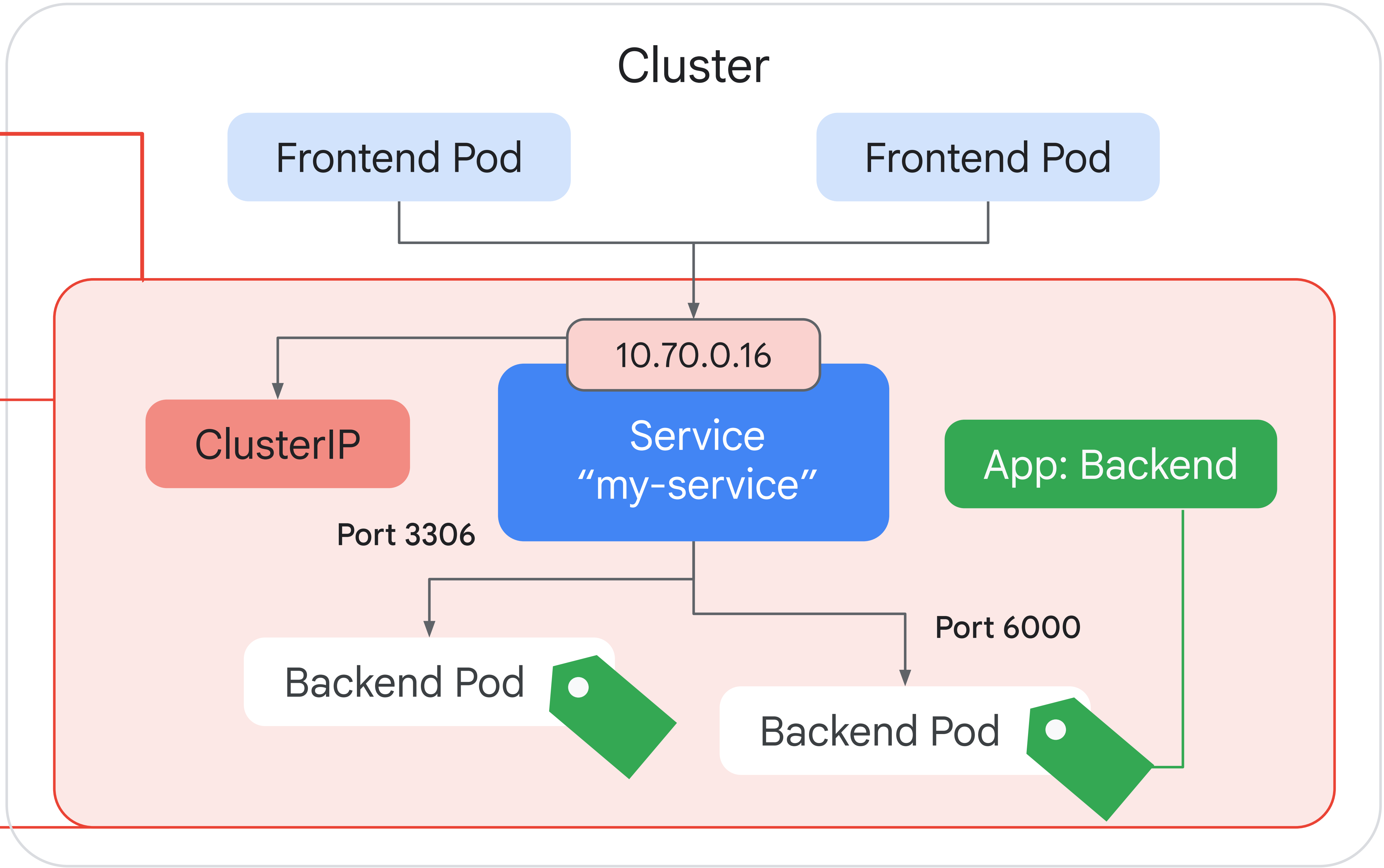
Create a Service object by defining its kind in a YAML file.

Use a label selector to choose the Pods that will run the target application.

Specify the port that the target containers are using.

The cluster control plane

Cluster control plane



Assigns a virtual IP address (ClusterIP)

The ClusterIP won’t change throughout the lifespan of the Service.

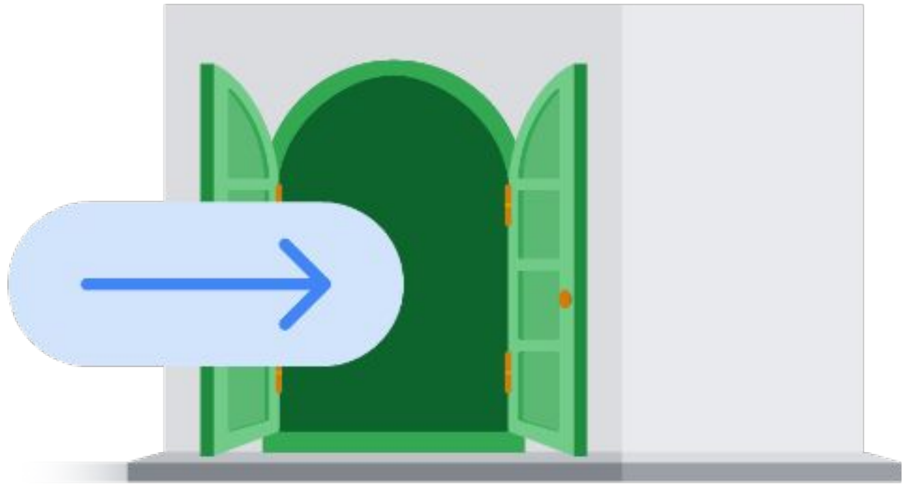
Selects Pods to include in the Service’s endpoints

NodePort Service

✓ ClusterIP Services are for internal communication, while NodePort services are used for external communication.

✓ A NodePort Service includes an underlying ClusterIP service.

✓ A ClusterIP Service is automatically created within a NodePort service for internal traffic distribution.

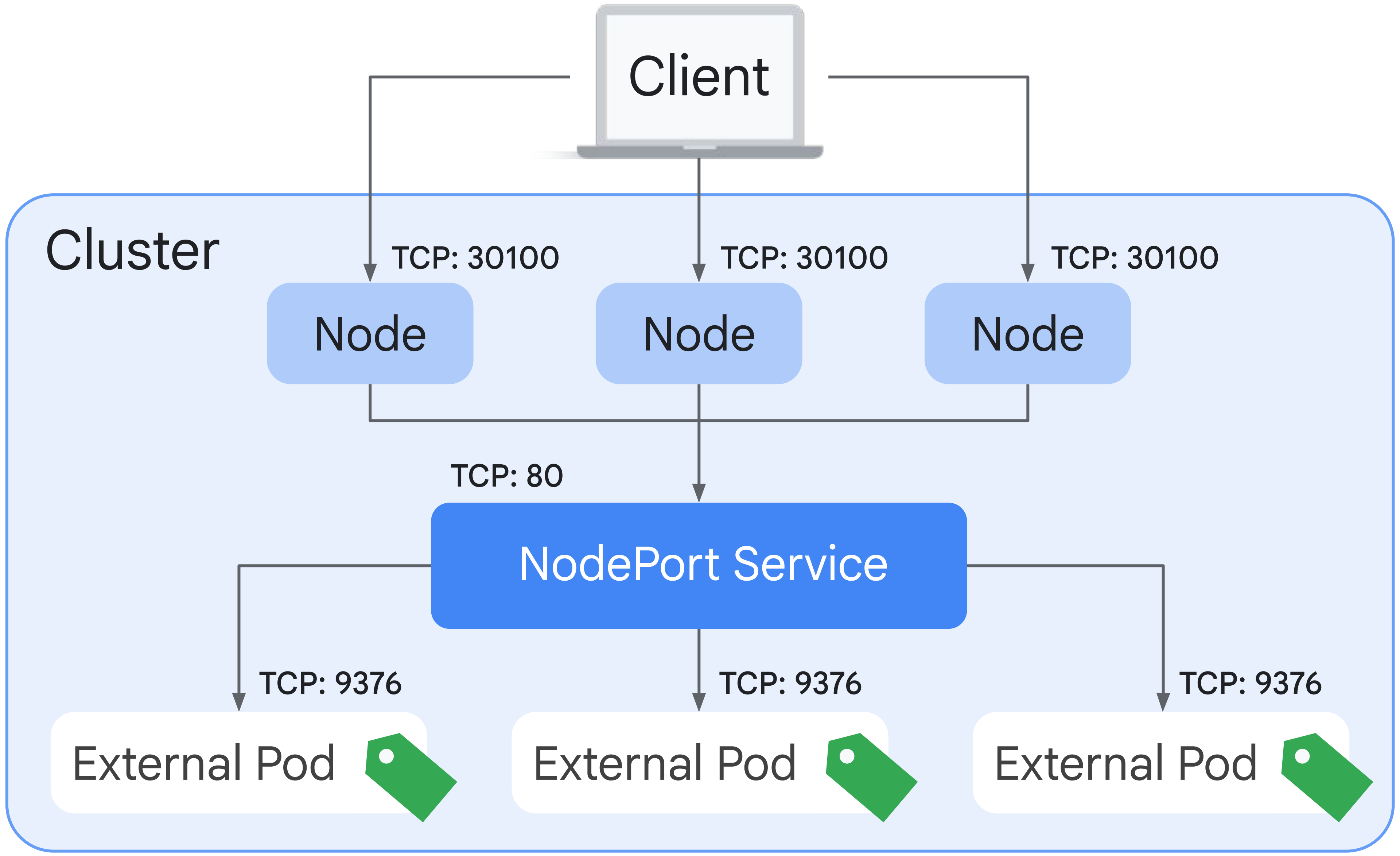


NodePort

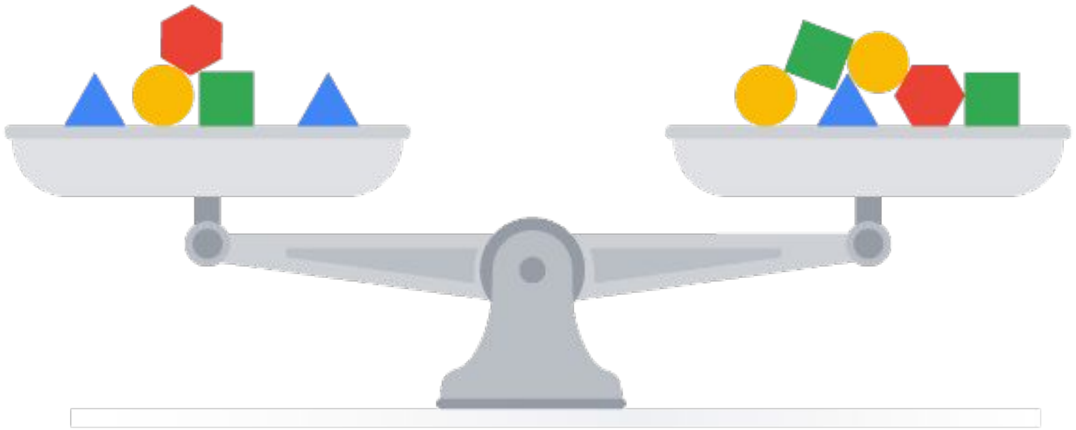
NodePort Service

Example

- There's a Service that can be reached from outside of the cluster by using the IP address of any node and the corresponding NodePort number.
- For this to work, incoming traffic is directed to a Service on port 80, then forwarded to a target Pod on port 9376.



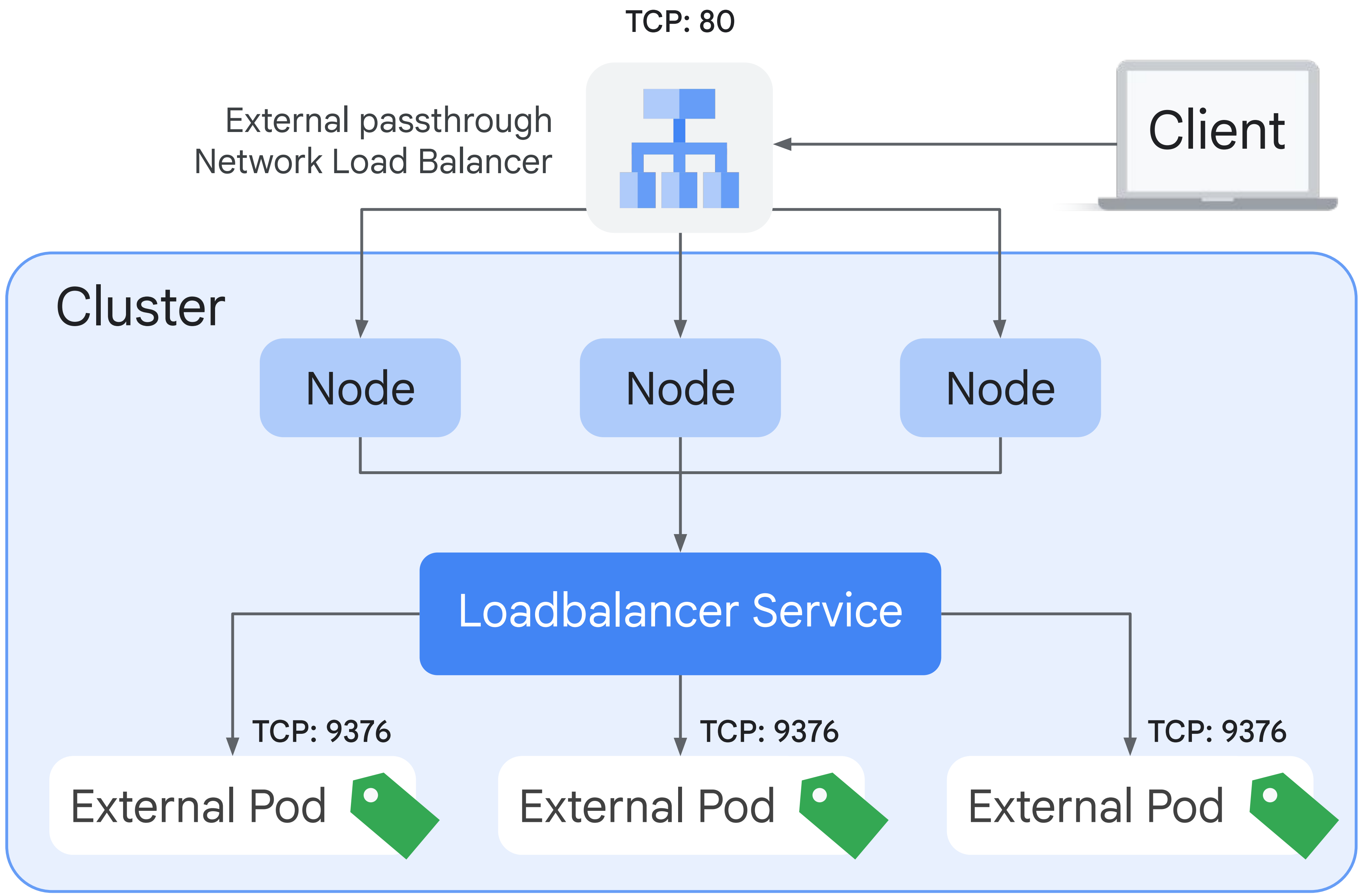
LoadBalancer Service



LoadBalancer

- ✓ Builds on the ClusterIP Service.
- ✓ Can be used to expose a Service to resources outside the cluster.
- ✓ Implemented using Google Cloud’s passthrough Network Load Balancer.

How does a LoadBalancer Service work?



GKE automatically creates an external passthrough Network Load Balancer when a LoadBalancer Service is deployed.

Client traffic is sent to the load balancer's external IP, which forwards it to the service nodes.

Nodes forward traffic to the internal LoadBalancer Service, which distributes it to a Pod.

Creating a LoadBalancer Service

```
[...]
spec:
  type: LoadBalancer
  selector:
      app: external
    ports:
      - protocol: TCP
      port: 80
        targetPort: 9376
```

Specify the type: **LoadBalancer.**

- Google Cloud assigns a static load balancer IP address.
- GKE automatically creates the LoadBalancer Service.
- GKE creates the appropriate load balancer (external or internal).

Google Kubernetes Engine Networking

01 Kubernetes networking

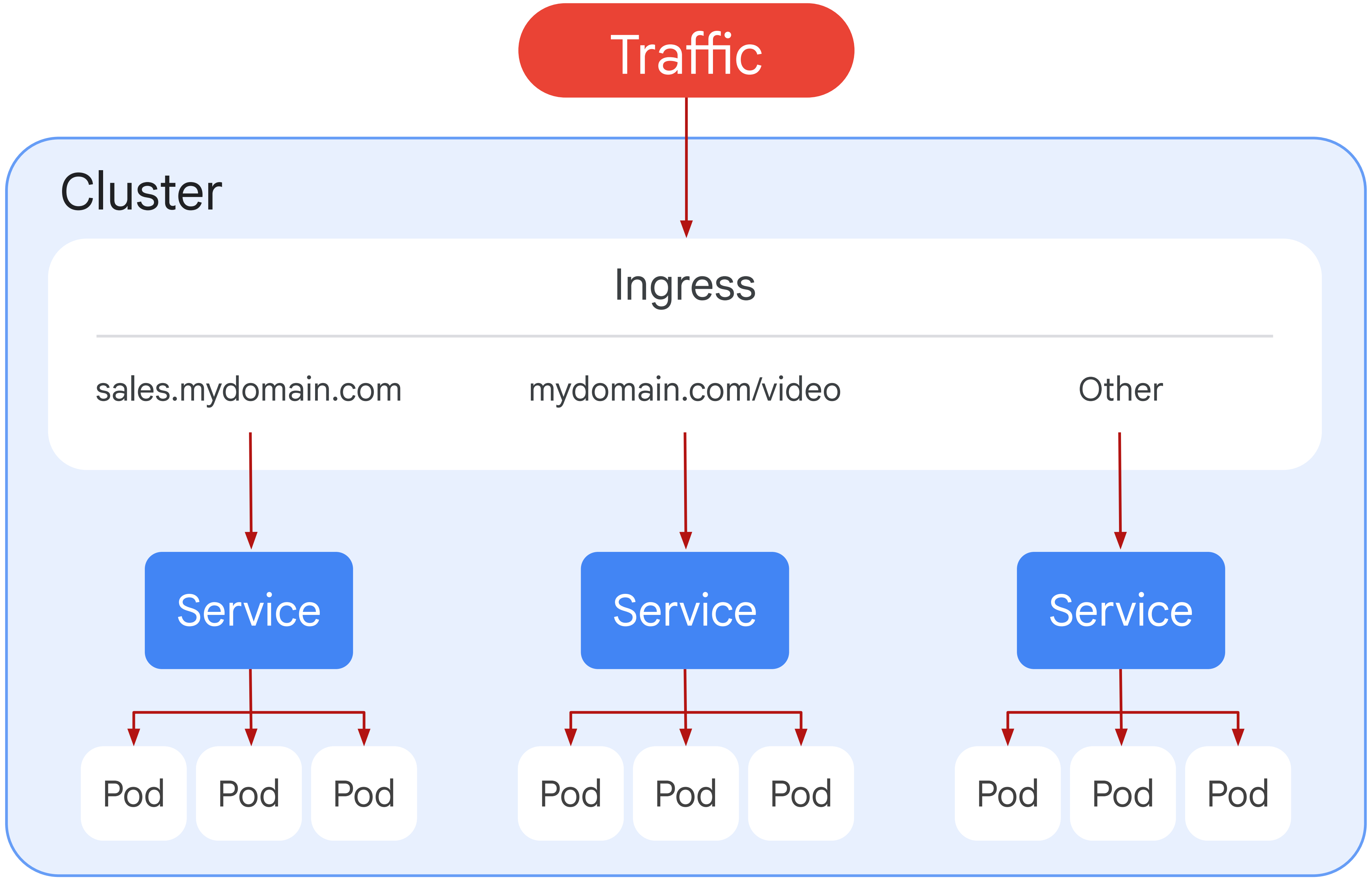
02 Kubernetes Services

03 Ingress

04 Container-native load balancing

05 Network policies in GKE

Ingress resource

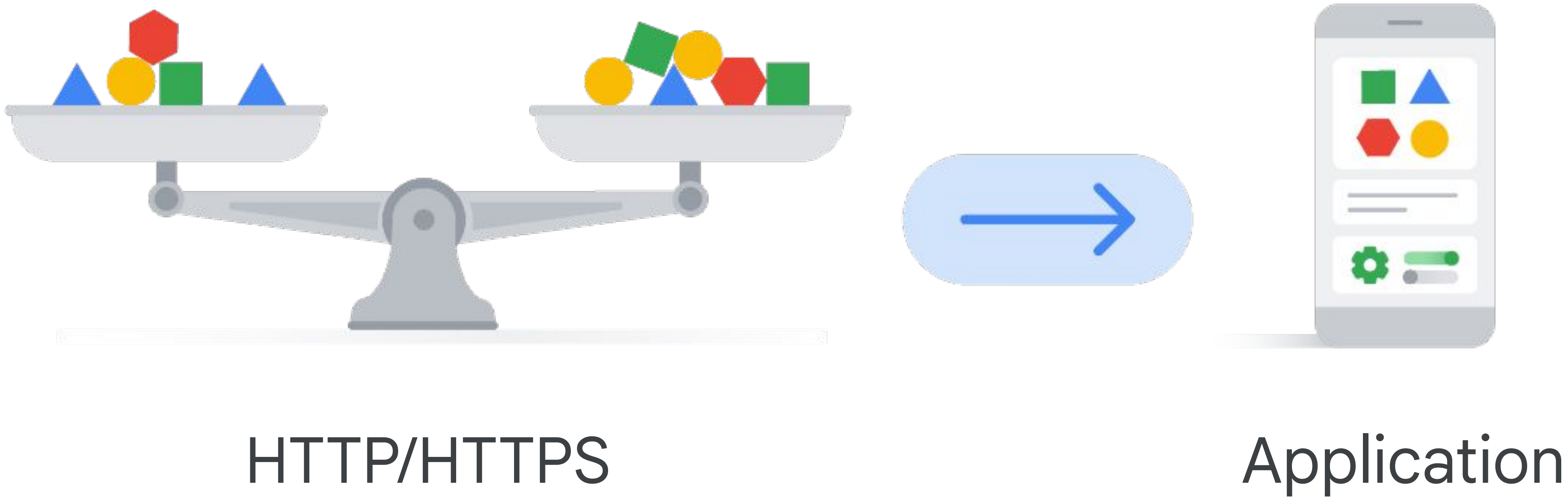


The Ingress resource operates one layer higher than Services.

Think of it as **a Service for Services.**

It's a collection of rules that direct external inbound connections to a set of Services within the cluster.

Kubernetes Ingress uses Cloud Load Balancing

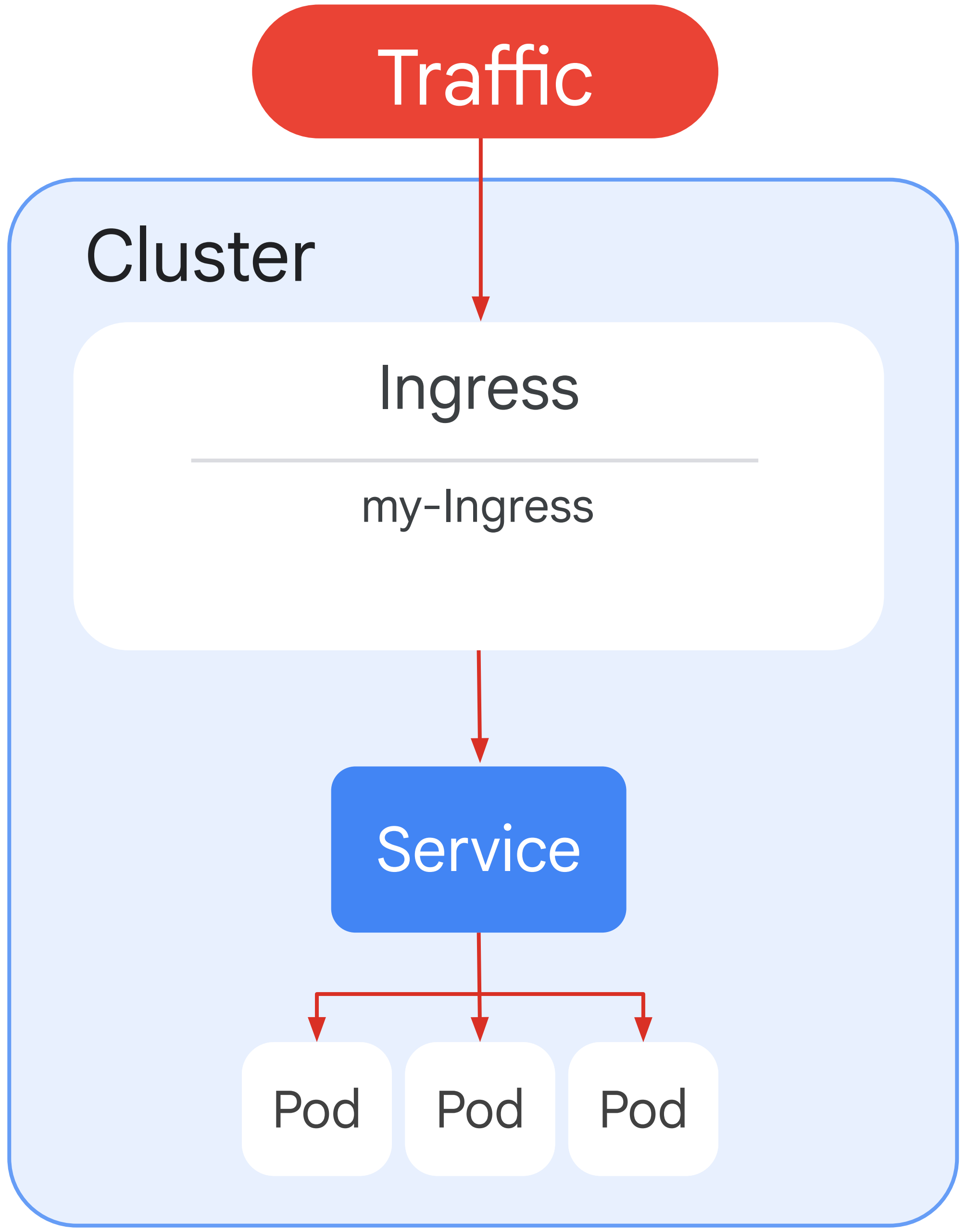


When an Ingress resource is created in the cluster, GKE creates an Application Load Balancer and configures it to route traffic to the application.

Ingress can deliver traffic to either **NodePort Services** or **LoadBalancer Services.**

Creating an Ingress

Example 1



```
apiVersion:
networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-Ingress
spec:
  backend:
    serviceName: demo
    servicePort: 80
```

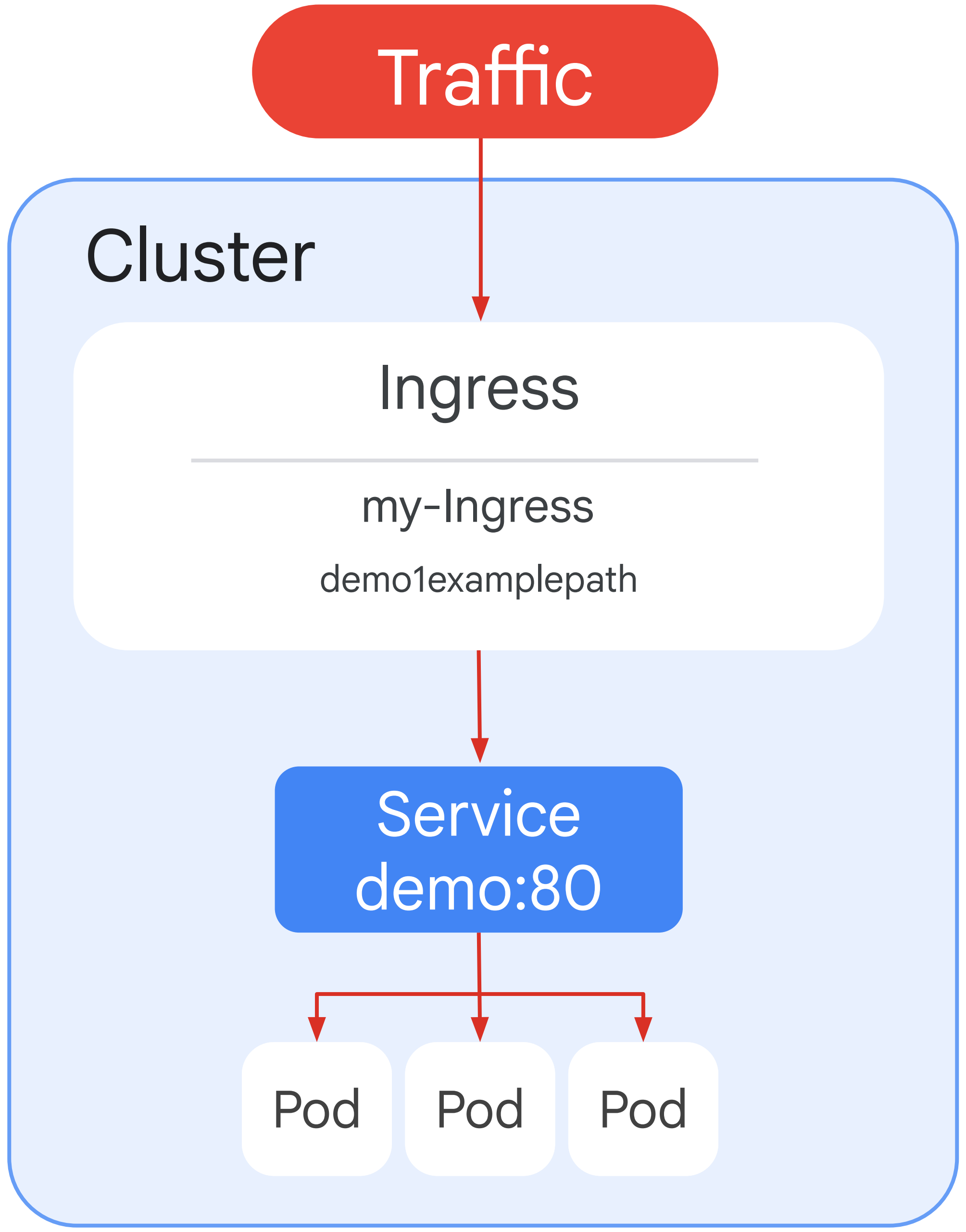
The Ingress controller creates Application Load Balancer.

The backend Service is selected by name and port.

- Routes traffic to specified service/port.

Creating an Ingress

Example 2



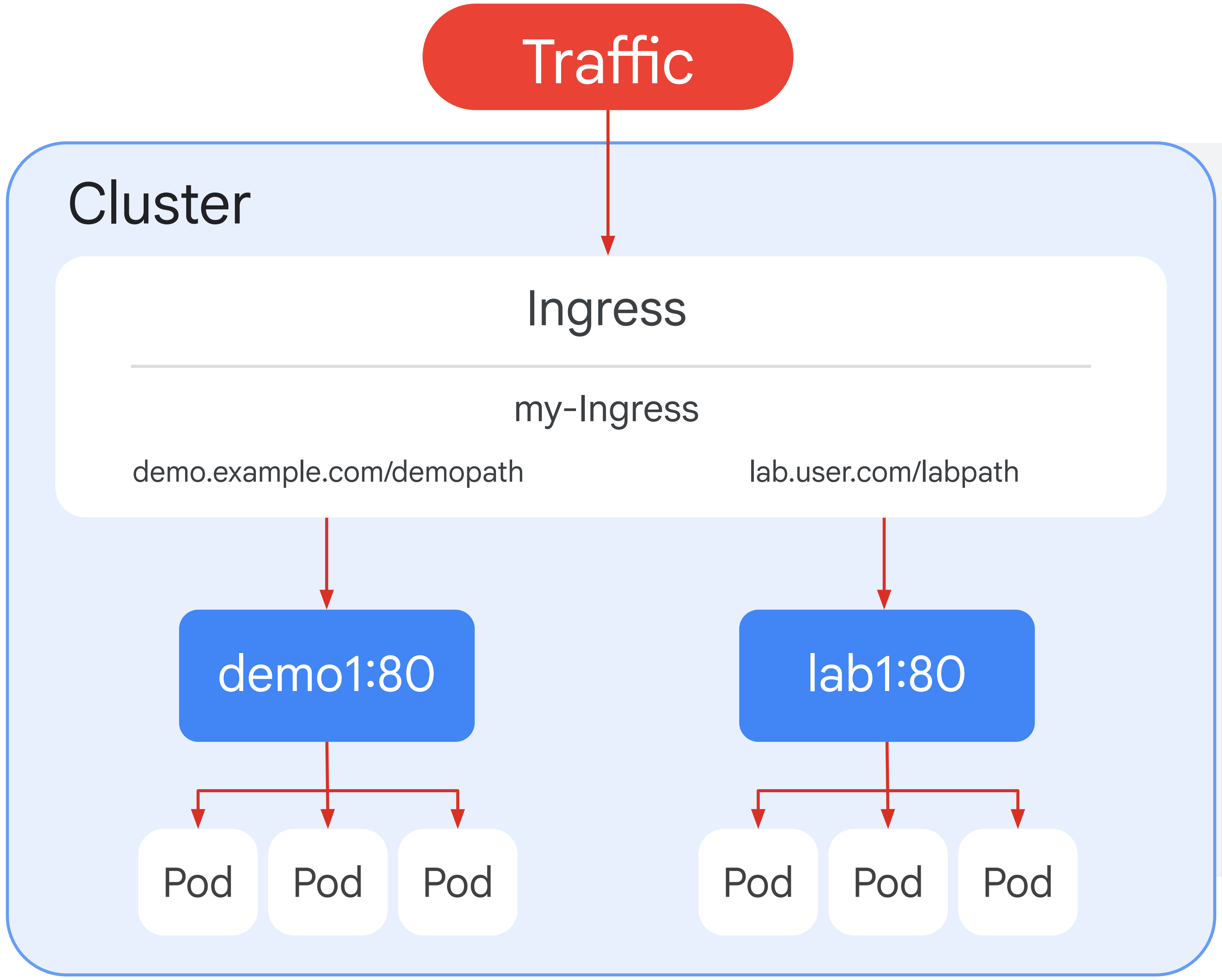
```
apiVersion:
networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-Ingress
spec:
  Rules:
  - host: demo1.example.com
    backend:
      serviceName: demo
      servicePort: 80
```

Inside this Ingress, the specifications are rules.

- GKE only supports HTTP rules, and each rule takes the same name as the host.
- The host name can be further filtered based on the path which will have a Service backend that defines the Service's name and port.

Creating an Ingress

Example 3



The traffic will be redirected from the Application Load Balancer, based on the host names, to their respective backend Services.

For example, the load balancer will route traffic for demo.example.com/demopath to the Service named demo1 on port 80.

Creating an Ingress

Example 3 (continued)

```
apiVersion:
networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-Ingress
spec:
  rules:
  - host: demo.example.com
    http:
      paths:
      - path: /demopath
        backend:
          serviceName: demo1
          servicePort: 80
```

```
- path: /labpath
  backend:
    serviceName: demo2
    servicePort: 80
```

This example considers rules based on the URL path.

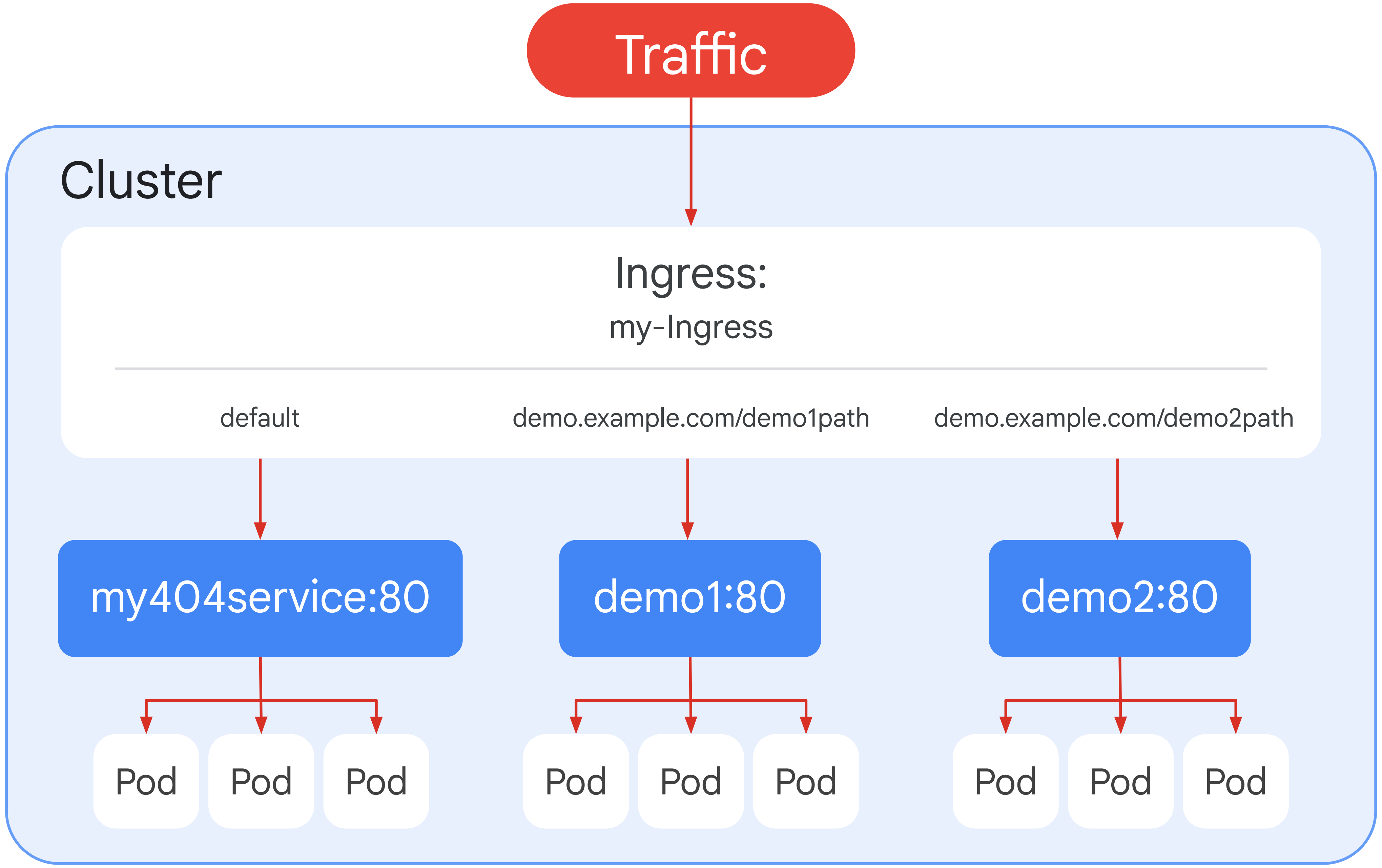
- Under Spec, a path defined as **/demopath** will be directed to the backend Service named **demo1**.
- Similarly, **/labpath** will be directed to its backend Service **demo2**.

What happens to the traffic that doesn’t match any rules?

Traffic with no matching rules is sent to the default backend.

Specify a default backend in the Ingress manifest to handle unmatched traffic.

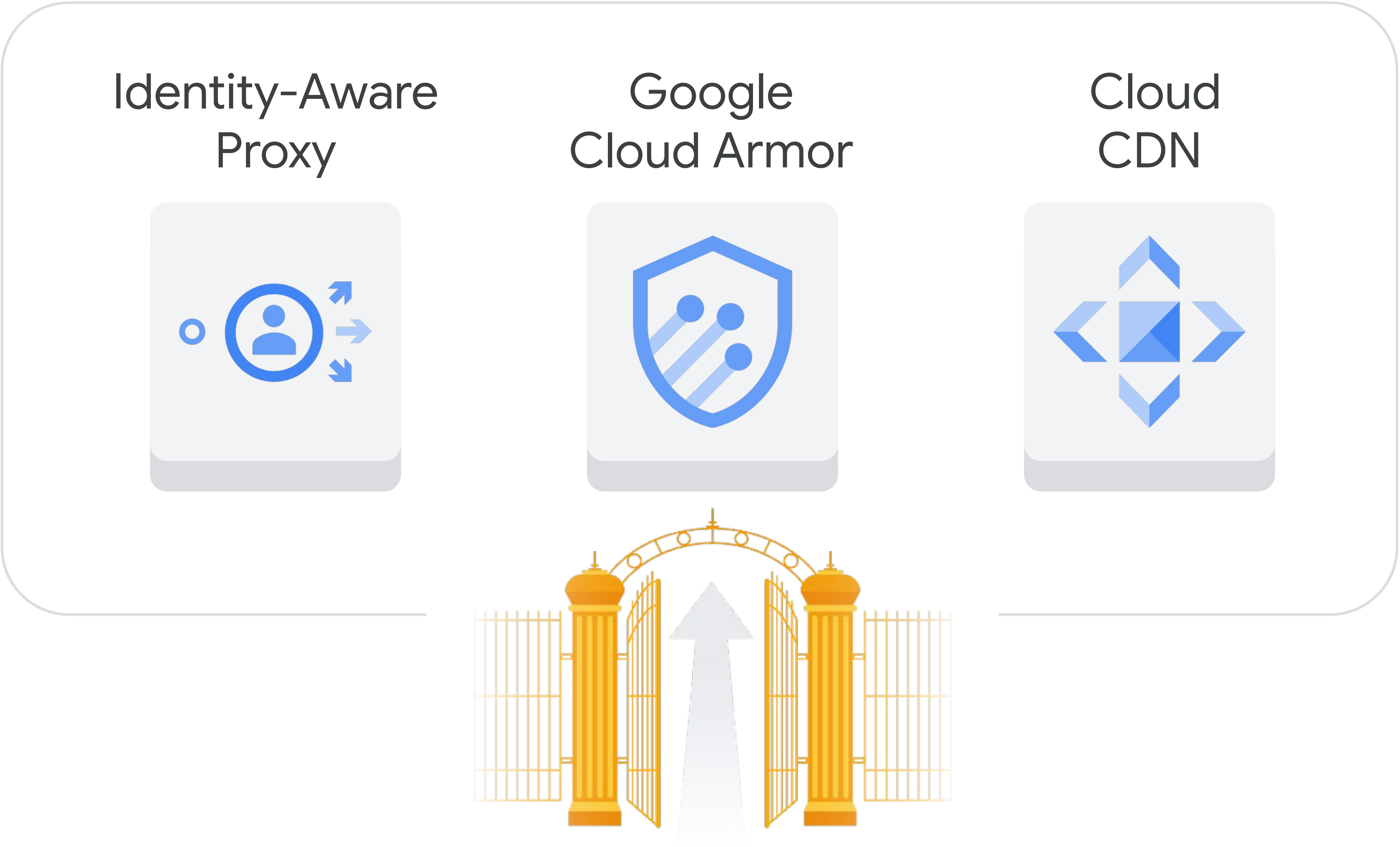
If no default backend is specified, GKE provides one that returns a 404 error.



Updating an Ingress

`$ kubectl edit ingress [NAME]`	Updates the Ingress manifest.
`$ kubectl replace [FILE]`	Replaces the Ingress object manifest file entirely.

Native Ingress support for Google Cloud services



Identity-Aware Proxy

Google Cloud Armor

Cloud CDN

Identity-Aware Proxy

Identity-Aware Proxy

Provides granular access control at the application level.

Authenticated users can have HTTPS access to the applications within a cluster without any VPN setup.

Google Cloud Armor

Google Cloud Armor

Protects against DDoS and web attacks.

Allows IP allow/deny lists and predefined rules.

Customizable security rules for varied threats.

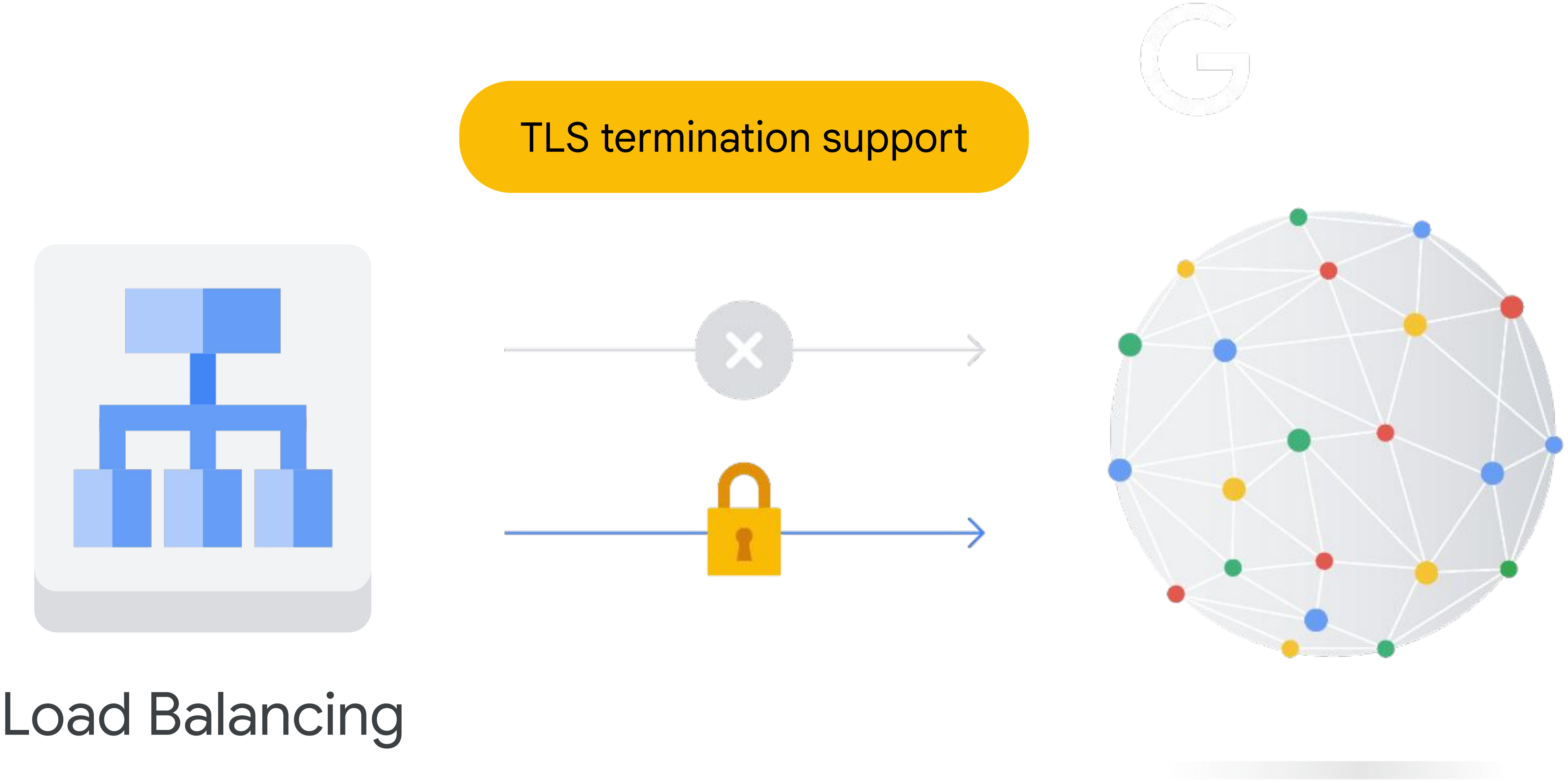
Cloud CDN

Cloud CDN

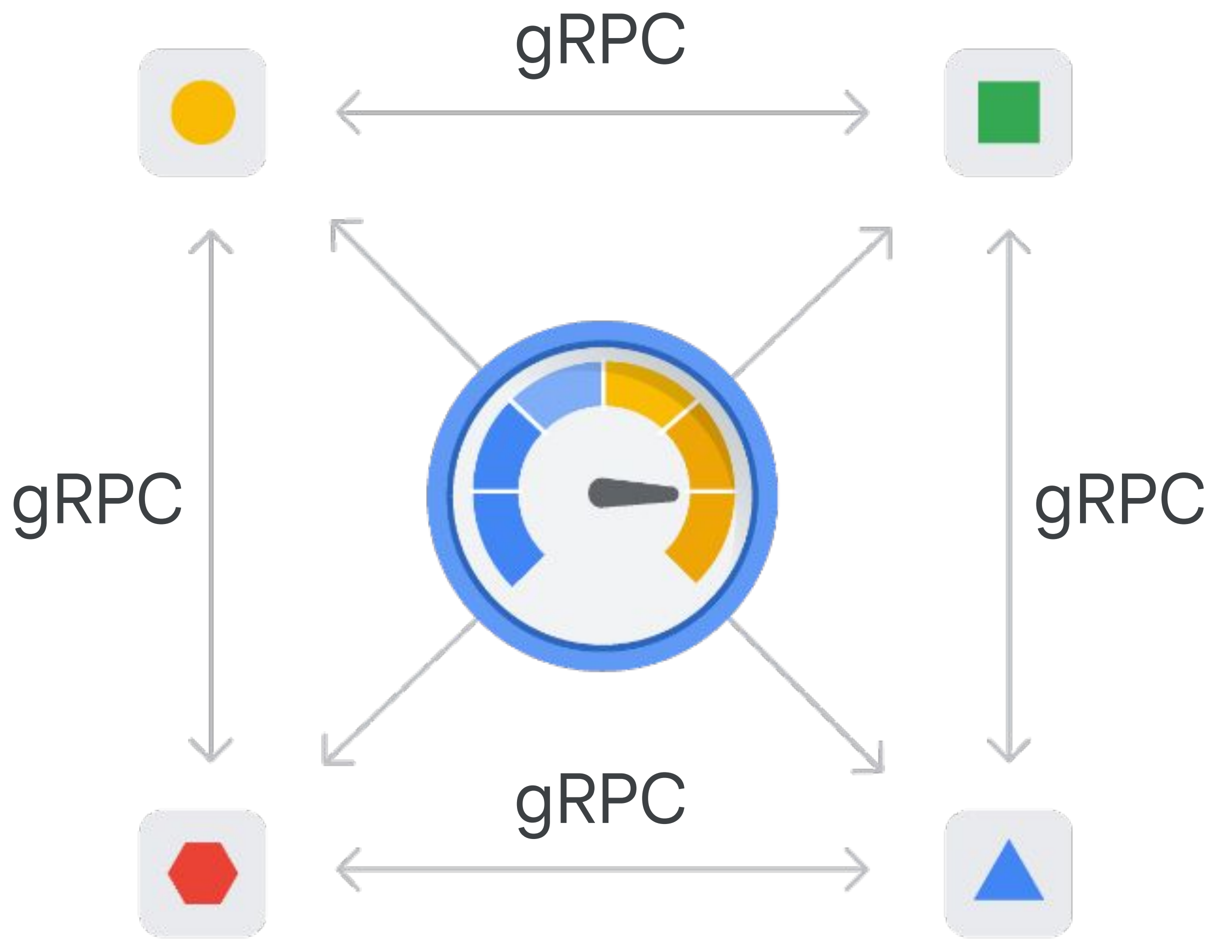
Allows an application’s content to be brought closer to its users by using more than 100 Edge points of presence.

Settings can be configured using **BackendConfig**, a custom resource used by the Ingress controller to define configuration for these Services.

Ingress gains security features from underlying Google Cloud resources



Ingress has support for HTTP/2, HTTP/1.0, and HTTP/1.1



Microservices must communicate efficiently using a high-performance remote procedure call system.

gRPC can be used along with HTTP/2 to create performant, low-latency, scalable microservices within the cluster.

Global load balancing with Ingress

A single standard Ingress resource can be used to load balance traffic globally to multiple clusters across multiple regions.

This also supports location-based load balancing, called **geo-balancing** across multiple regions, which improves the availability of the cluster.

Google Kubernetes Engine Networking

01 Kubernetes networking

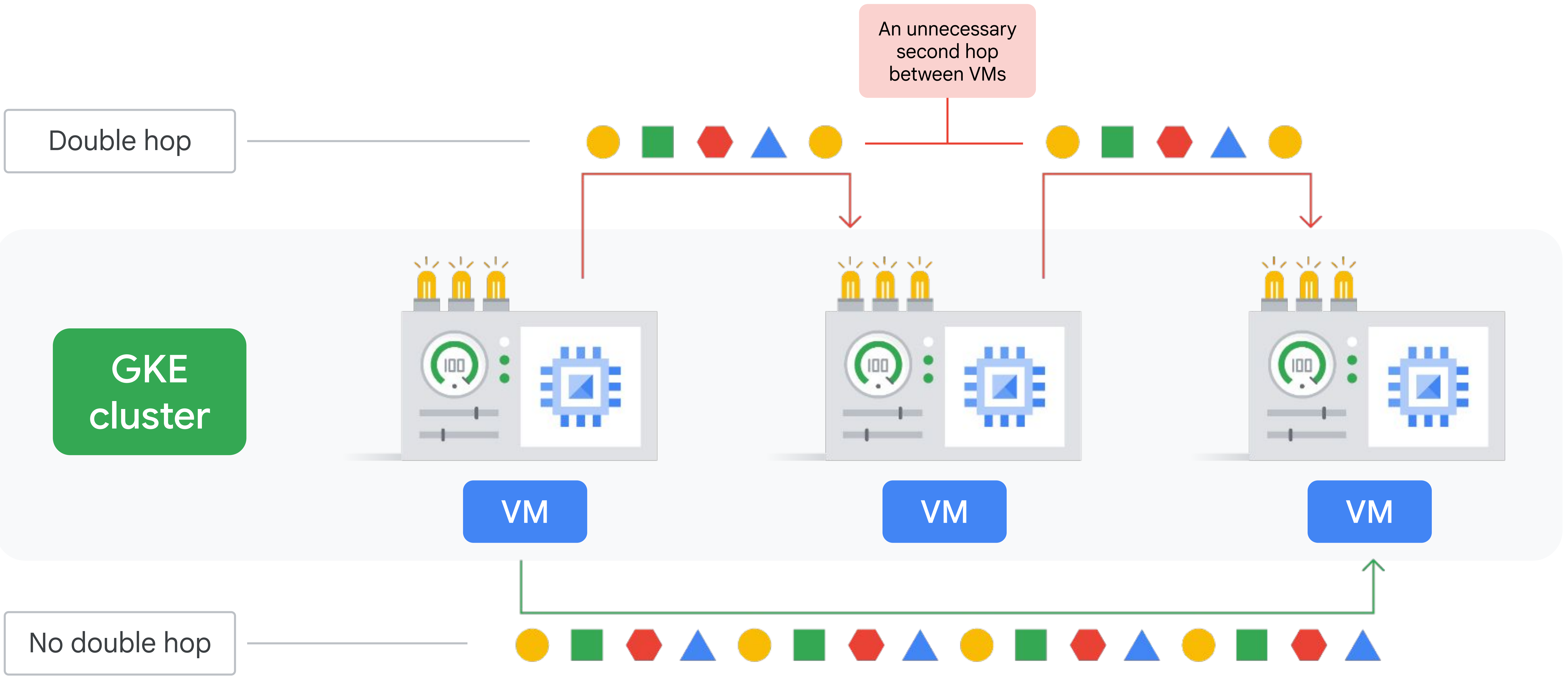
02 Kubernetes services

03 Ingress

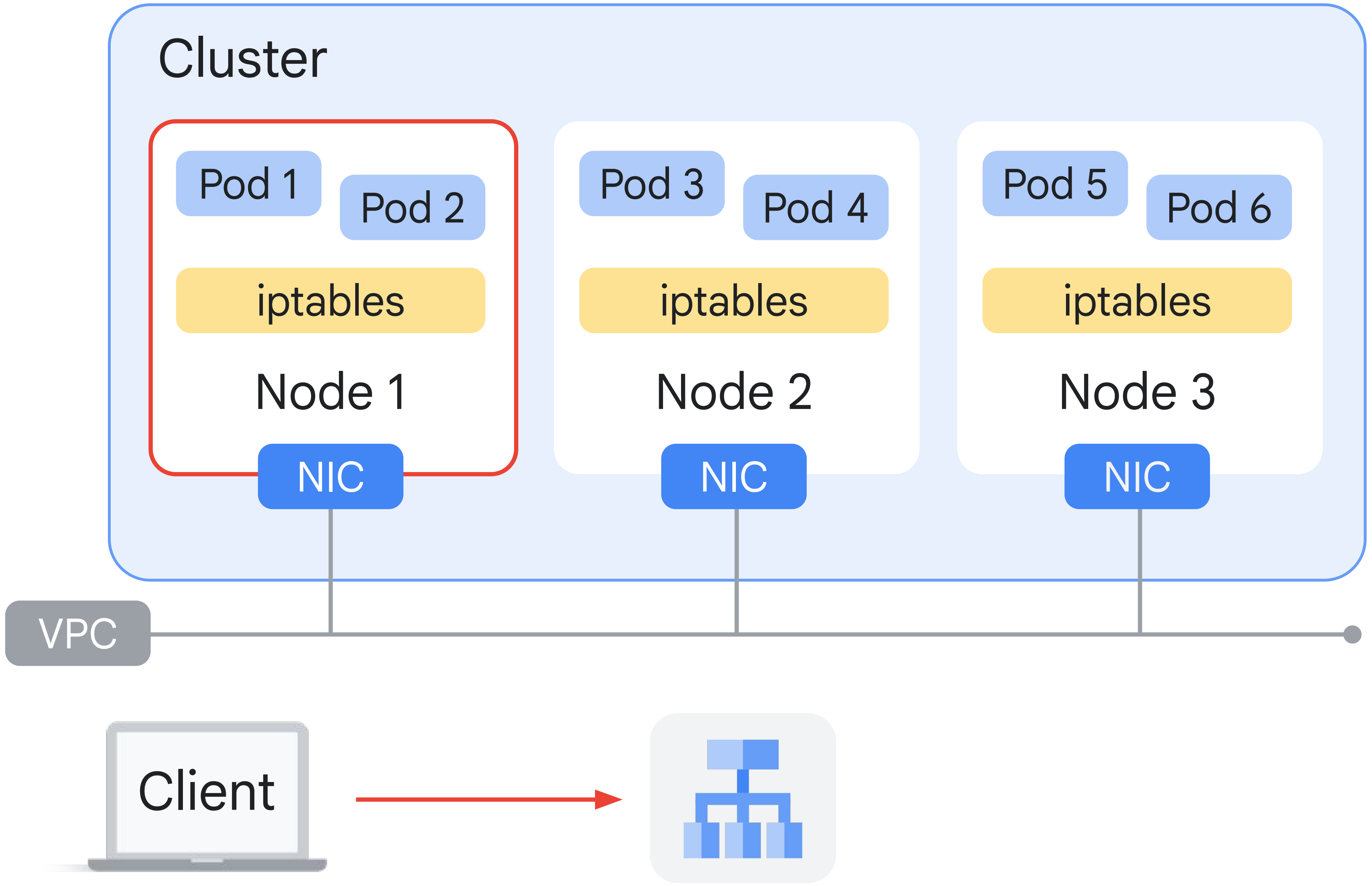
04 Container-native load balancing

05 Network policies in GKE

The double-hop dilemma



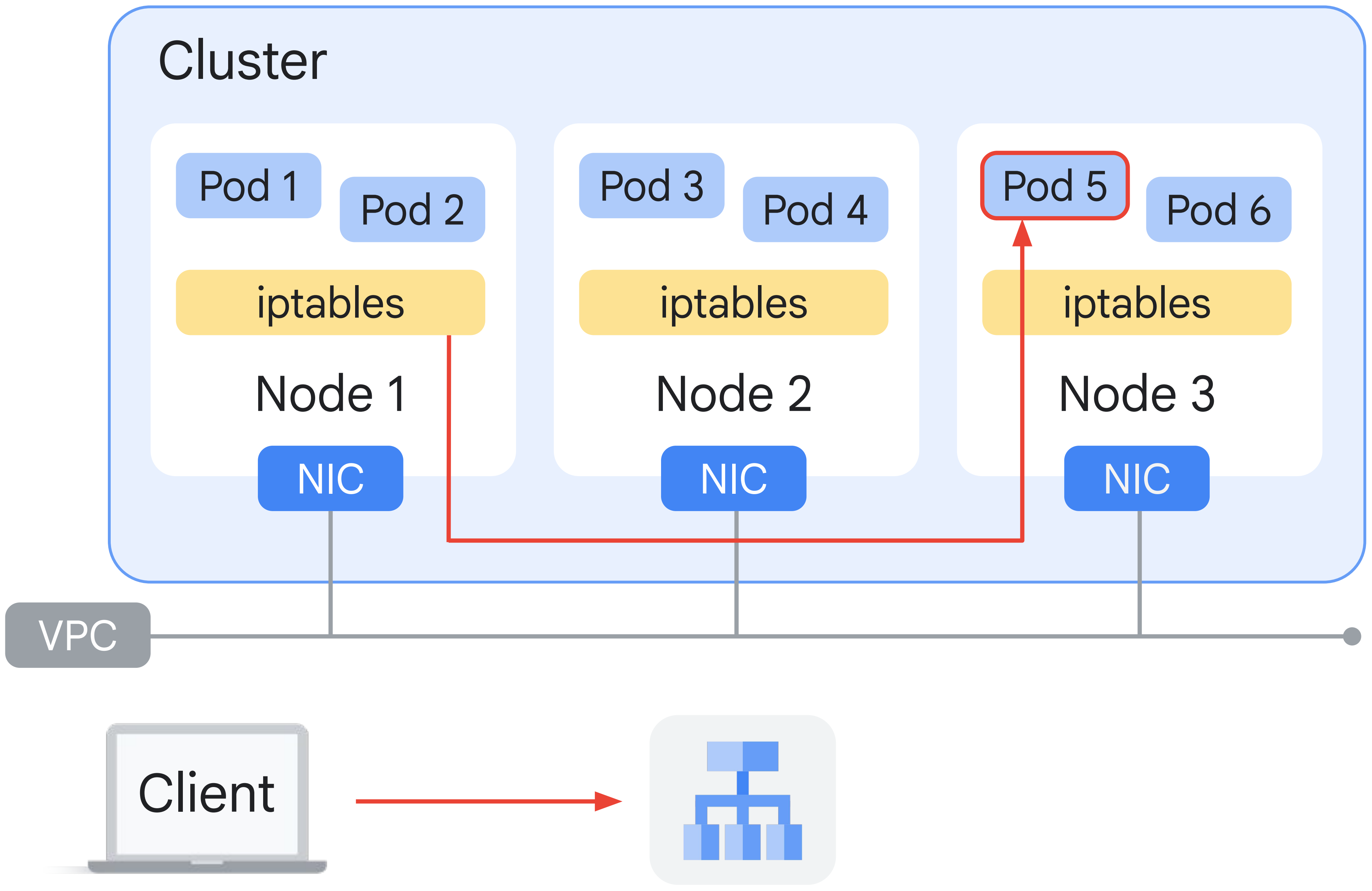
Traffic distribution without a container-native load balancer



The Network Load Balancer chooses a random node in the cluster and forwards the traffic to it.

In this example, there are three possible Nodes to choose from and **Node 1** is chosen.

Traffic distribution without a container-native load balancer

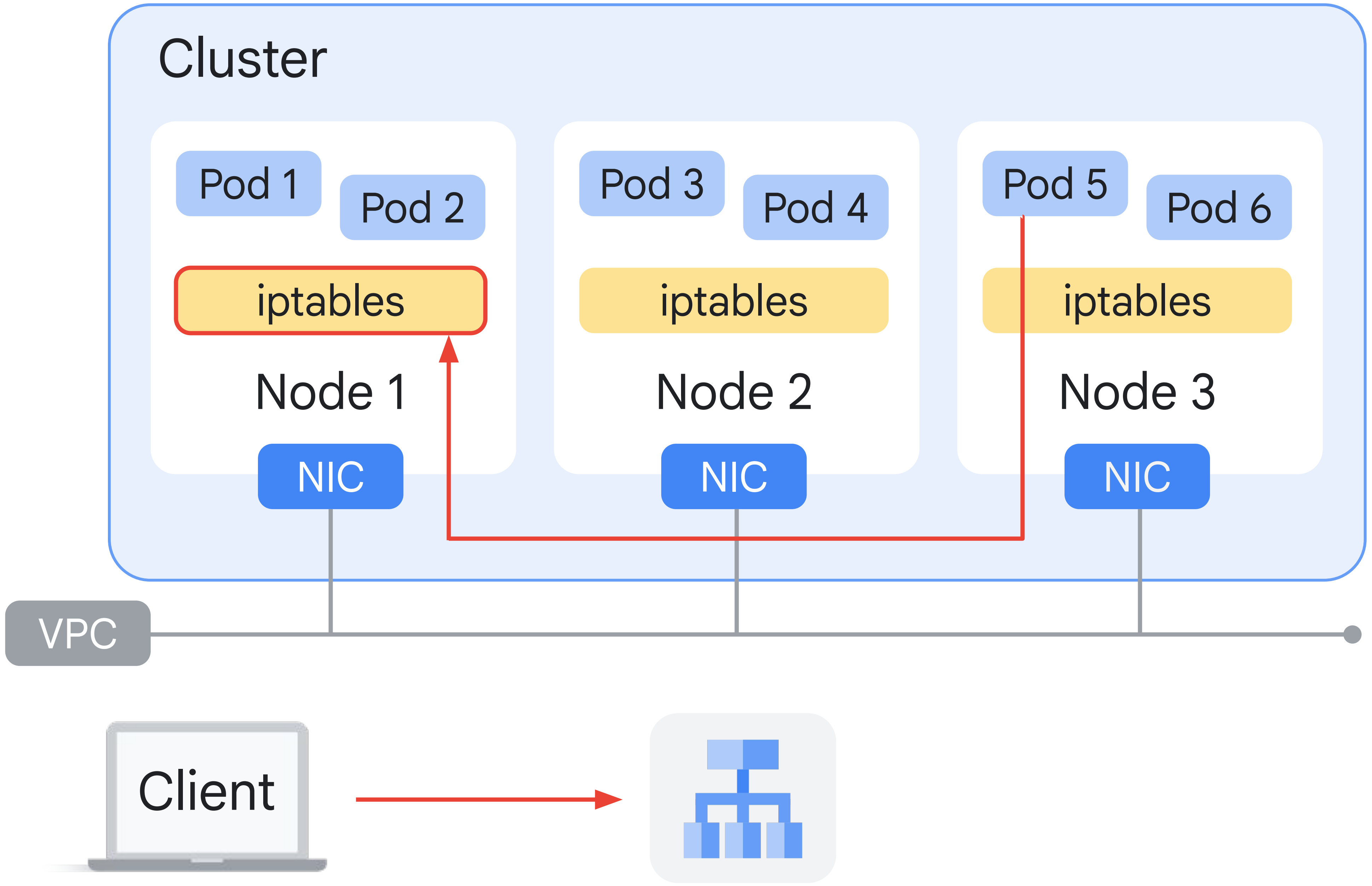


The initial node will use kube-proxy to select a Pod at random to handle the incoming traffic.

Node 1 chooses **Pod 5**, which isn't on this node.

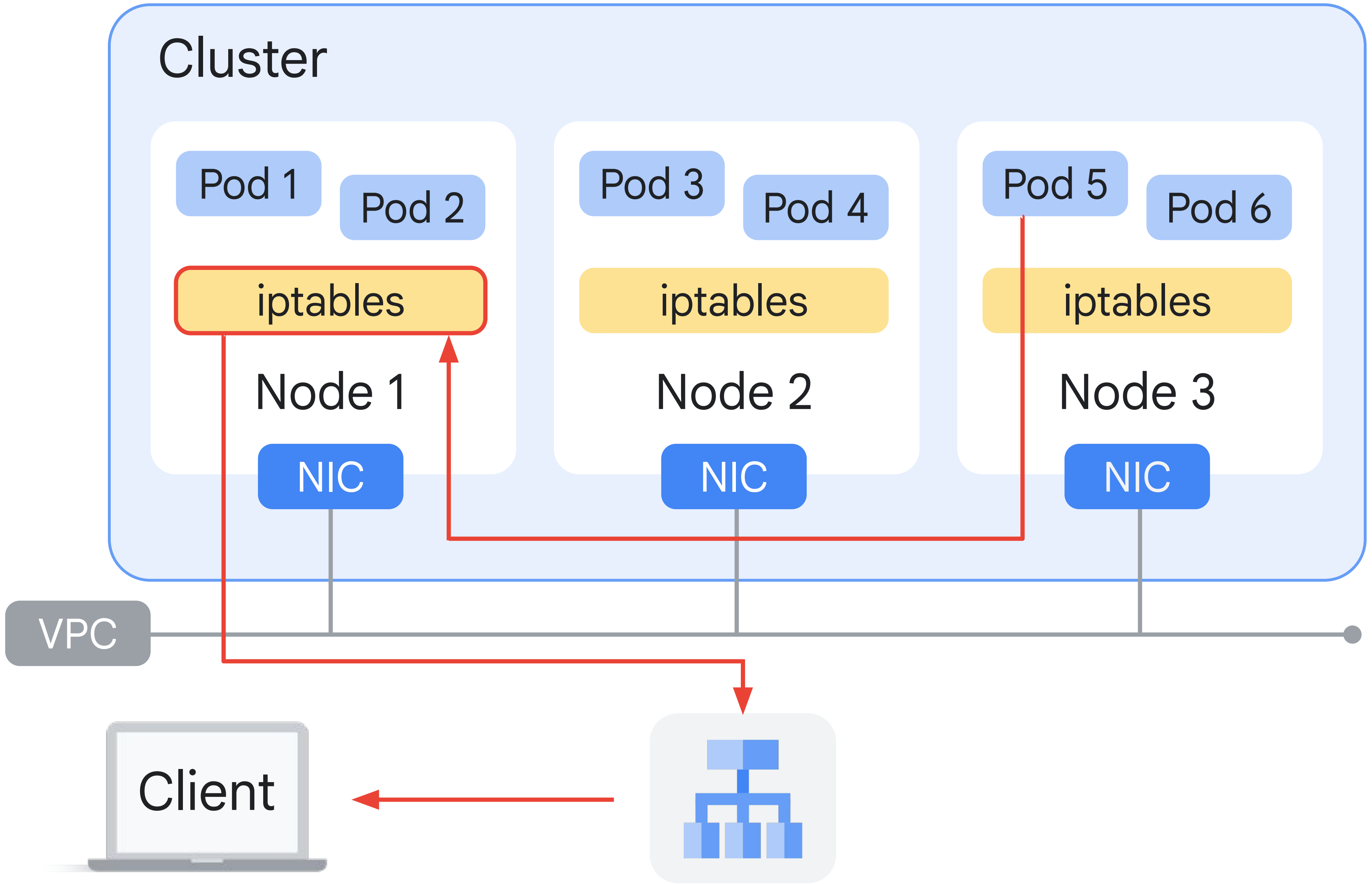
This means that **Node 1** will forward the traffic to **Pod 5** on **Node 3**.

Traffic distribution without a container-native load balancer



Pod 5 then directs its responses back through **Node 1**, which is when the double-hop happens.

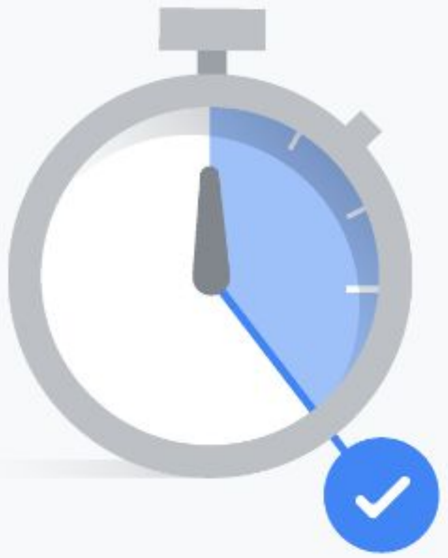
Traffic distribution without a container-native load balancer



Node 1 then forwards the traffic back to the Network Load Balancer, which sends it back to the client.

A double-hop is not optimal for load balancing.

What’s more important?



Lowest possible latency

Most even cluster load balancing

Prioritizing low latency

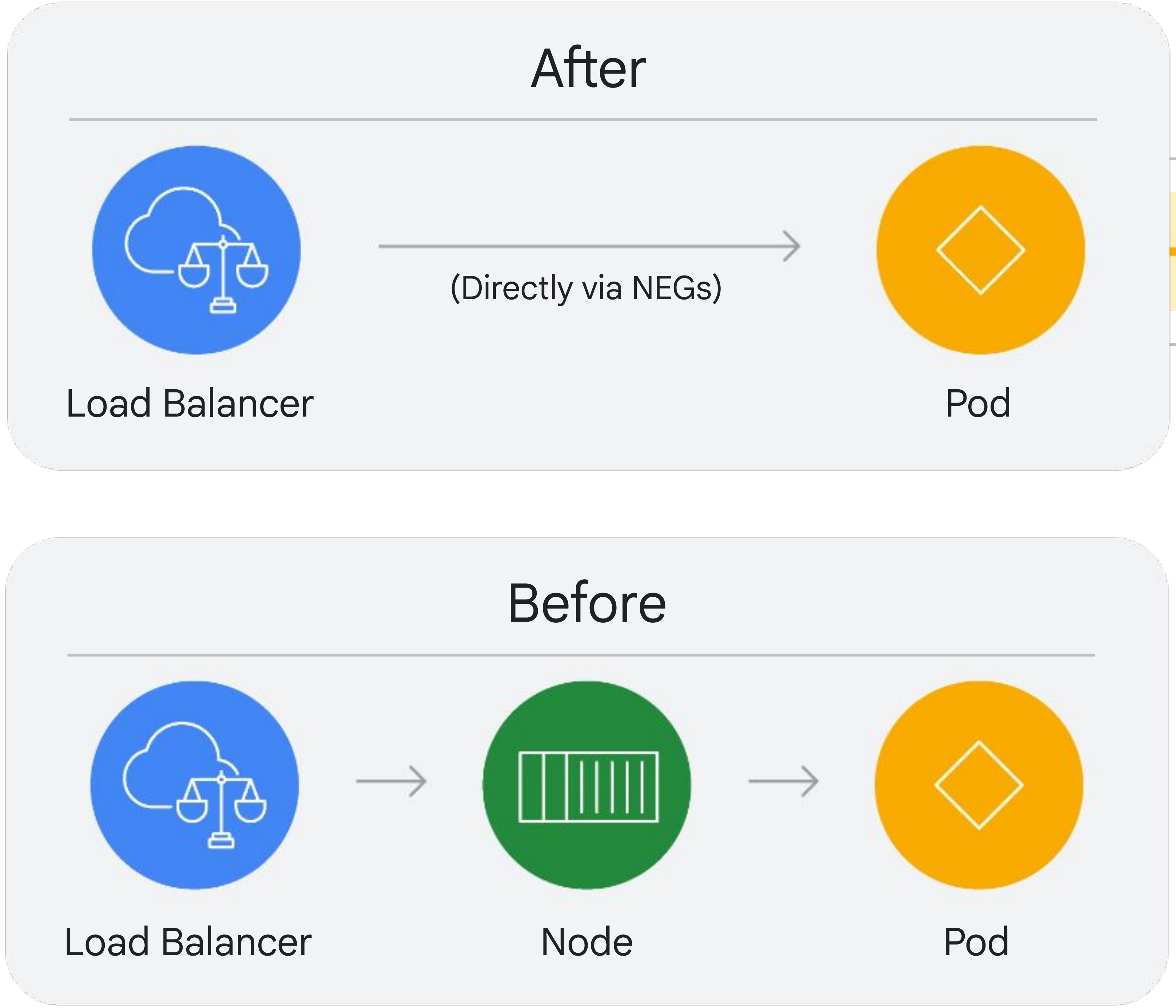
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
   externalTrafficPolicy: Local
  selector:
     app: external
  ports:
     - protocol: TCP
       port: 80
       targetPort: 9376
```

Configure the **LoadBalancer Service** to force the kube-proxy to choose a Pod local to the node that received the client traffic.

Set the **externalTrafficPolicy** field to “Local.”

This eliminates the double-hop to another node as the kube-proxy will always choose a Pod on the receiving node.

Container-native Load Balancing



The load balancer directs traffic to the Pods directly instead of to the nodes.

This method requires:

- GKE clusters to operate in VPC-native mode.
- A data model called network endpoint groups (NEGs), which represent IP-to-port pairs.

What’s the best choice? It depends.

Lowest possible latency

Most even cluster load balancing

“Local” external-traffic policy may cause other issues because:

- It imposes constraints on the mechanisms that balance Pod traffic internally.
- The Application Load Balancer forwards traffic via nodes with no awareness of the state of the Pods themselves.

Container-native load balancing benefits

✓ Pods can be specified directly as endpoints for Google Cloud load balancers.

✓ Features, such as traffic shaping and advanced algorithms, are supported.

✓ Direct visibility to the Pods and more accurate health checks.

✓ Time it takes traffic to travel from the client to the load balancer can be measured.

✓ Fewer network hops in the path, which optimizes the data path.

✓ Support for Google Cloud networking services.

Support for Google Cloud networking services

Google Cloud Armor

Cloud CDN

Identity-Aware Proxy

Google Kubernetes Engine Networking

01 Kubernetes networking

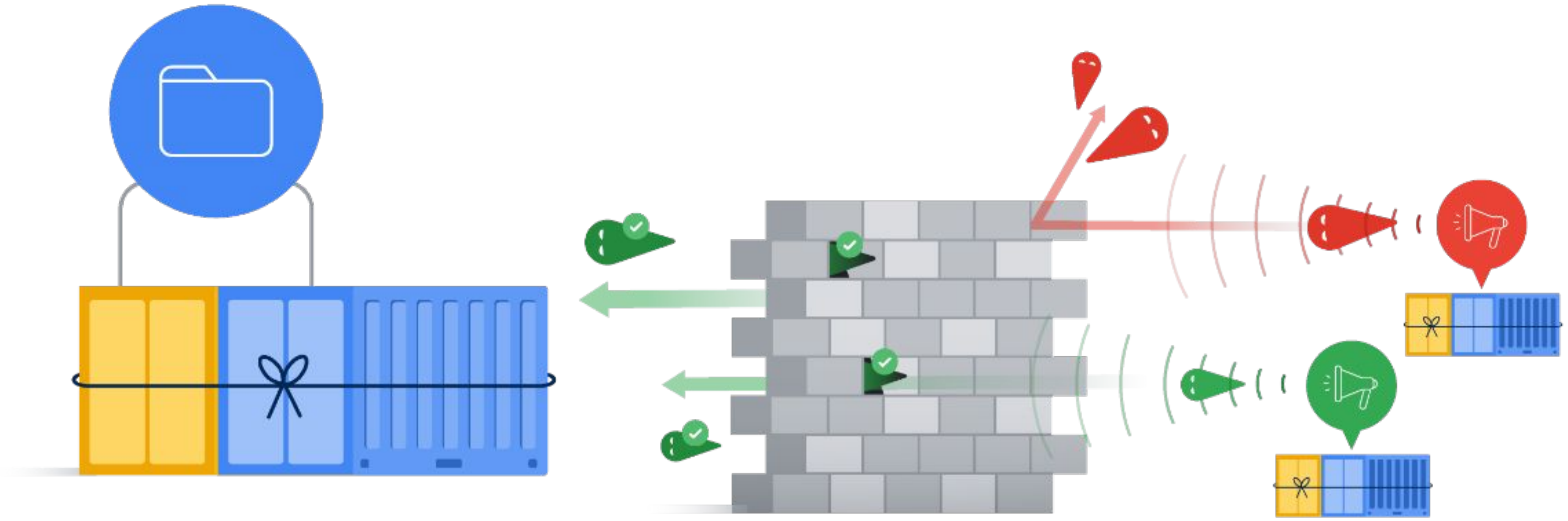
02 Kubernetes Services

03 Ingress

04 Container-native load balancing

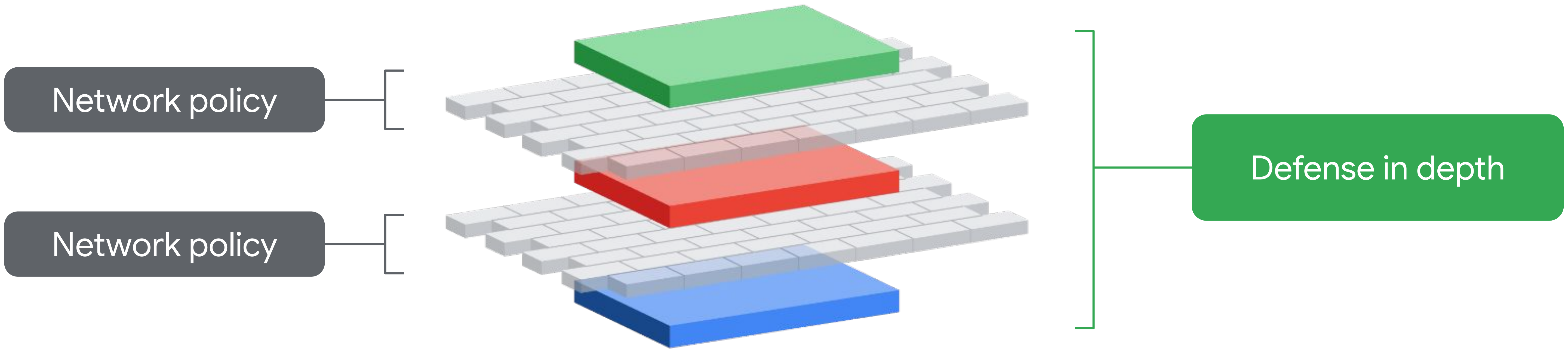
05 Network policies in GKE

Restricting access to Pods with network policies

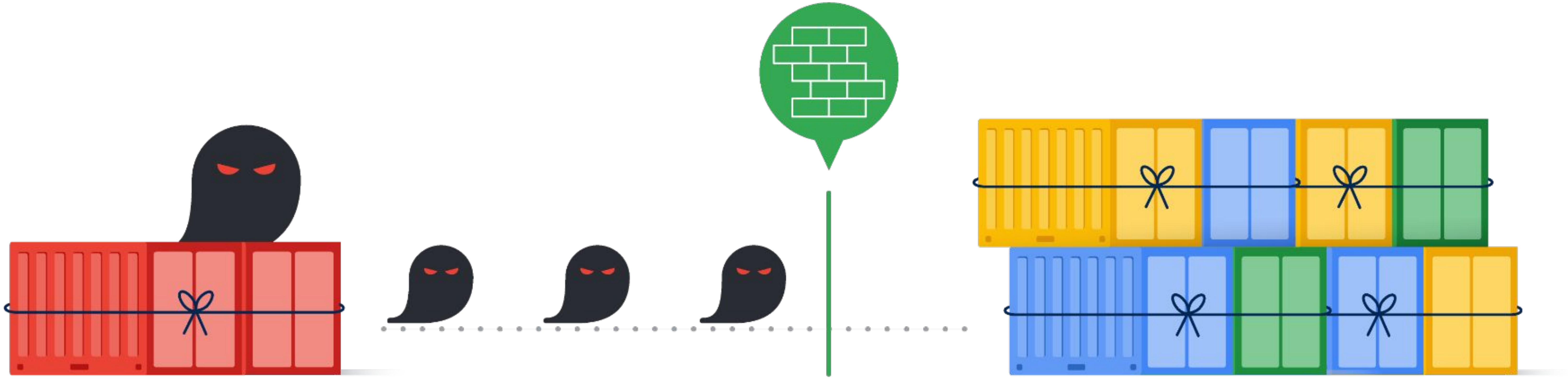


A **network policy** is a set of firewall rules applied at the Pod level that restrict access to other Pods and Services inside the cluster.

Network policies can be used to restrict access at each stack level



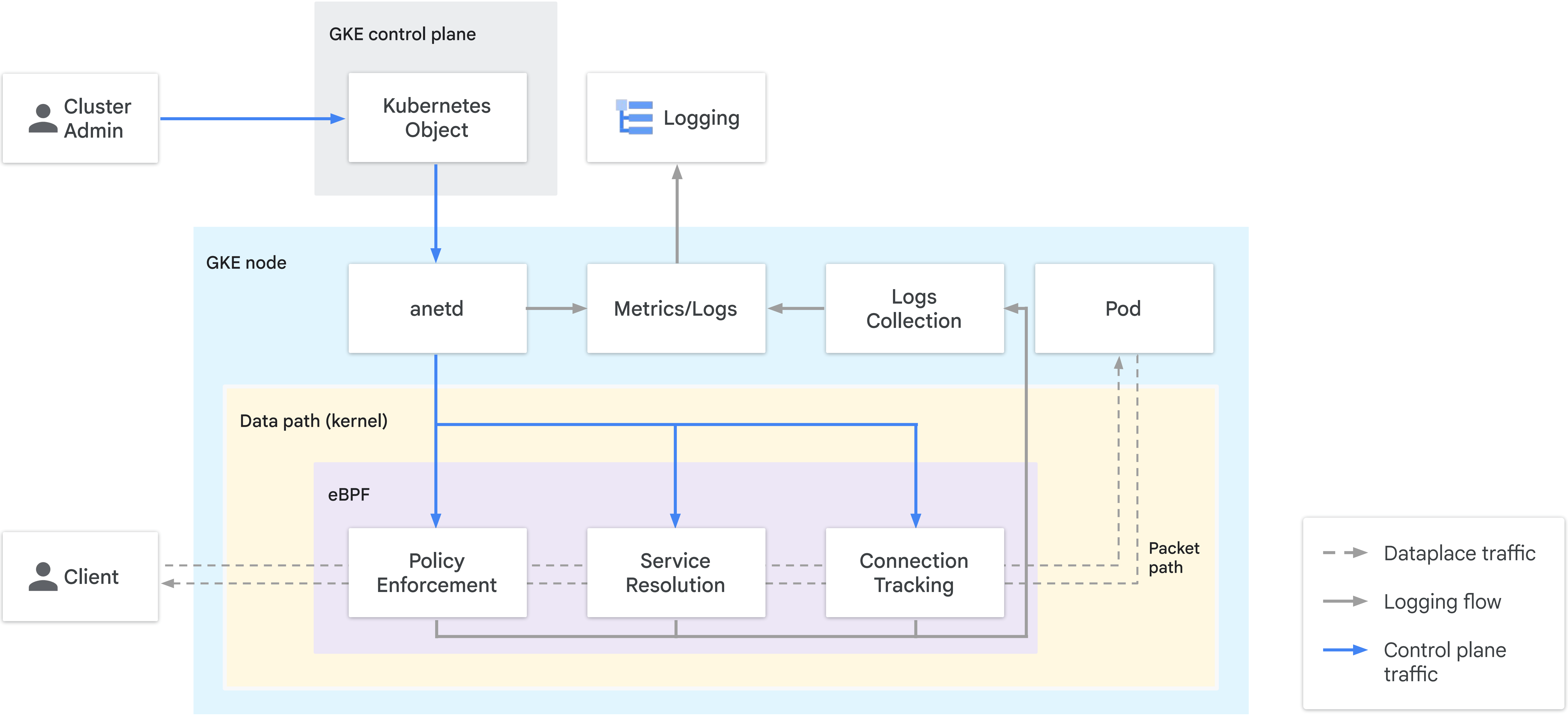
Why consider network policies?



Imagine an attacker compromised one of your Pods by exploiting a security vulnerability in it.

Network policies let you lock down network traffic to only the pathways you want traffic to travel.

Enabling network policies with GKE Dataplane V2



Define the actual network policy

Defining the network policy is necessary after enabling network policy enforcement.

Enabling network policy enforcement may require **increasing cluster size** due to increased resource consumption.

NetworkPolicy, podSelector, and policyTypes

```
apiVersion:
networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: demo-network-policy
 namespace: default
spec:
 podSelector:
  matchLabels:
   role: demo-app
 policyTypes:
 - Ingress
 - Egress
```

```
ingress:
- from:
 - ipBlock:
   cidr: 172.17.0.0/16
   except:
   - 172.17.1.0/24
 - namespaceSelector:
   matchLabels:
    project: myproject
 - podSelector:
   matchLabels:
    role: frontend
 ports:
 - protocol: TCP
  port: 6379
```

Network Policies are written in YAML files and have the kind **NetworkPolicy**.

The **podSelector** lets you select a set of Pods based on labels.

policyTypes indicates whether ingress, egress, or both traffic restrictions will be applied.

The Ingress section of the policy

```
apiVersion:
networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
  namespace: default
spec:
  podSelector:
   matchLabels:
     role: demo-app
  policyTypes:
  - Ingress
  - Egress
```

```
ingress:
- from:
  - ipBlock:
      cidr: 172.17.0.0/16
      except:
      - 172.17.1.0/24
  - namespaceSelector:
      matchLabels:
        project: myproject
  - podSelector:
      matchLabels:
        role: frontend
  ports:
  - protocol: TCP
   port: 6379
```

In the **Ingress** section of the policy, there are two main sections: from and ports.

- The from section can be from three sources:
  - ipBlock
  - namespaceSelector
  - podSelector
- The ports section states what ports ingress will be accepted from.

The Egress section of the policy

```
apiVersion:
networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
  namespace: default
spec:
  podSelector:
   matchLabels:
     role: demo-app
  policyTypes:
  - Ingress
  - Egress
```

```
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/24
  ports:
  - protocol: TCP
    port: 5978
```

In the **Egress** section of the policy, there are two main sections: **to** and **ports.**

In this example, traffic destined for network 10.0.0.0/24 on TCP port 5978 will be permitted to egress from the demo-app Pods.

Disabling a network policy

Disable a network policy for a cluster

```
gcloud container clusters update [NAME] \
--no-enable-network-policy
```

✓ Google Cloud console

Step 1: Disable the network policy for Nodes.

Step 2: Disable the network policy for the control plane.

If no network policies exist, all traffic between Pods in the namespace is allowed.

Default policies for Ingress and Egress

```
metadata:
  name: default-deny
spec:
   podSelector: {}
   policyTypes:
   - Ingress
```

```
metadata:
  name: allow-all
spec:
   podSelector: {}
   policyTypes:
   - Ingress
   ingress:
    - {}
```

```
metadata:
  name: default-deny
spec:
   podSelector: {}
   policyTypes:
   - Ingress
   - Egress
```

```
metadata:
  name: default-deny
spec:
   podSelector: {}
   policyTypes:
   - Egress
```

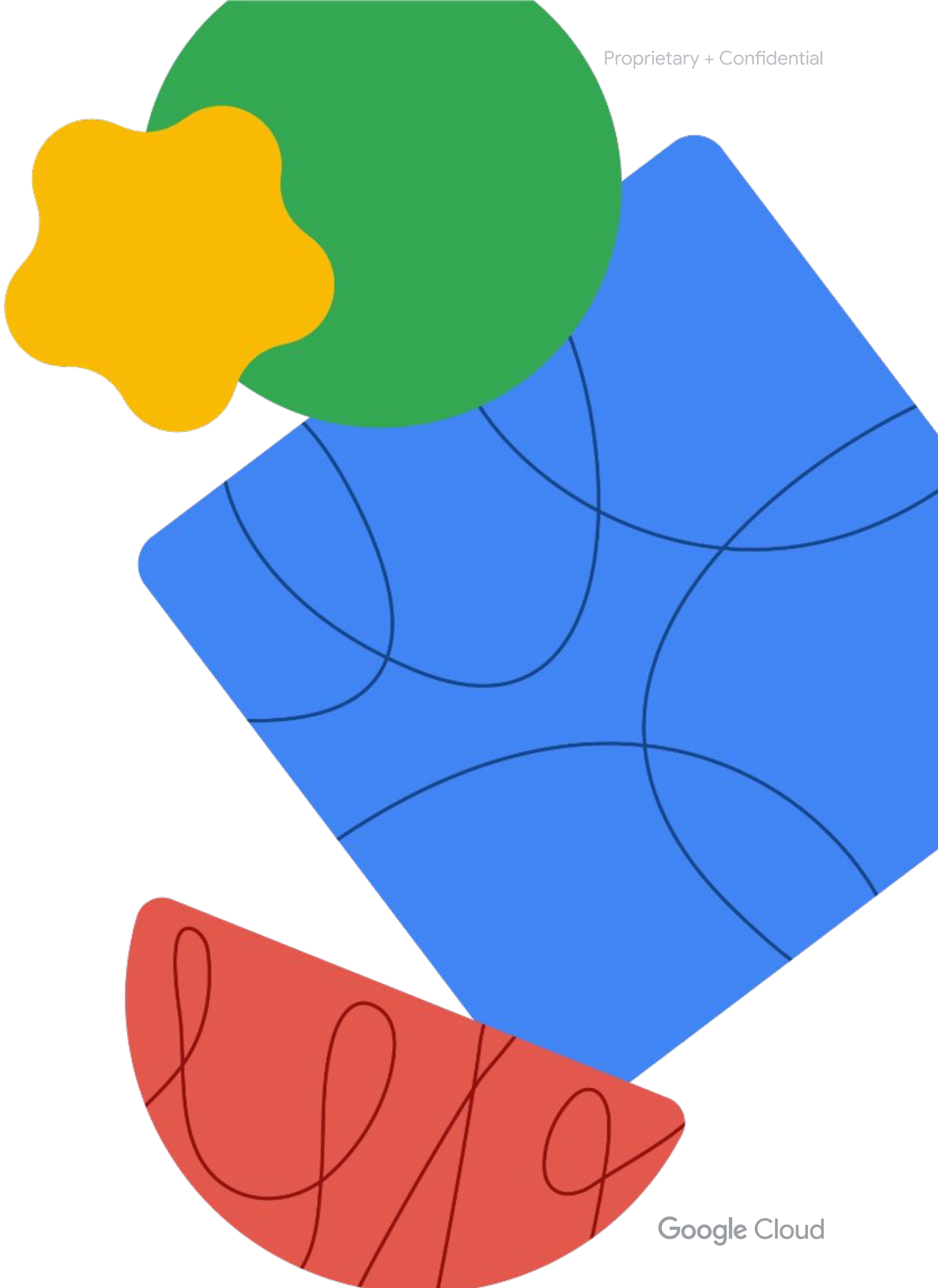
default-deny blocks all incoming or outgoing traffic respectively

```
metadata:
  name: allow-all
spec:
   podSelector: {}
   policyTypes:
   - Egress
   ingress:
    - {}
```

allow-all allows all traffic in either the Ingress or Egress direction

Quiz questions

Let’s pause for a quick check in.



Quiz | Question 1

Question

Your Pod has been rescheduled, and the original IP address that was assigned to the Pod is no longer accessible. What is the reason for this?

A. The new Pod IP address is blocked by a firewall.

B. The old Pod IP address is blocked by a firewall.

C. The Pod IP range for the cluster is exhausted.

D. The new Pod has received a different IP address.

Quiz | Question 1

Answer

Your Pod has been rescheduled, and the original IP address that was assigned to the Pod is no longer accessible. What is the reason for this?

A. The new Pod IP address is blocked by a firewall.

B. The old Pod IP address is blocked by a firewall.

C. The Pod IP range for the cluster is exhausted.

[✓ D. The new Pod has received a different IP address.]

Quiz | Question 2

Question

You have updated your application and deployed a new Pod. How can you ensure consistent network access to the Pod throughout its lifecycle?

A. Deploy a Kubernetes Service with a selector that locates the application's Pods.

B. Register the fully qualified domain name of the application's Pod in DNS.

C. Add the fully qualified domain name of the application's Pod to your local hostfile.

D. Add metadata annotations to the Pod manifest that define a persistent DNS name.

Quiz | Question 2

Answer

You have updated your application and deployed a new Pod. How can you ensure consistent network access to the Pod throughout its lifecycle?

A. Deploy a Kubernetes Service with a selector that locates the application's Pods. ✓

B. Register the fully qualified domain name of the application's Pod in DNS.

C. Add the fully qualified domain name of the application's Pod to your local hostfile.

D. Add metadata annotations to the Pod manifest that define a persistent DNS name.

Quiz | Question 3

Question

You are designing a GKE solution. One of your requirements is that network traffic load balancing should be directed to Pods instead of balanced across nodes. How can you enable this for your environment?

A. Configure or migrate your cluster to VPC-Native Mode and deploy a container-native load balancer.

B. Set the externalTrafficPolicy field to local in the YAML manifest for your external services.

C. Configure all external access for your application using Ingress resources rather than services.

D. Configure affinity and anti-affinity rules that ensure your application's Pods are distributed across nodes.

Quiz | Question 3

Answer

You are designing a GKE solution. One of your requirements is that network traffic load balancing should be directed to Pods instead of balanced across nodes. How can you enable this for your environment?

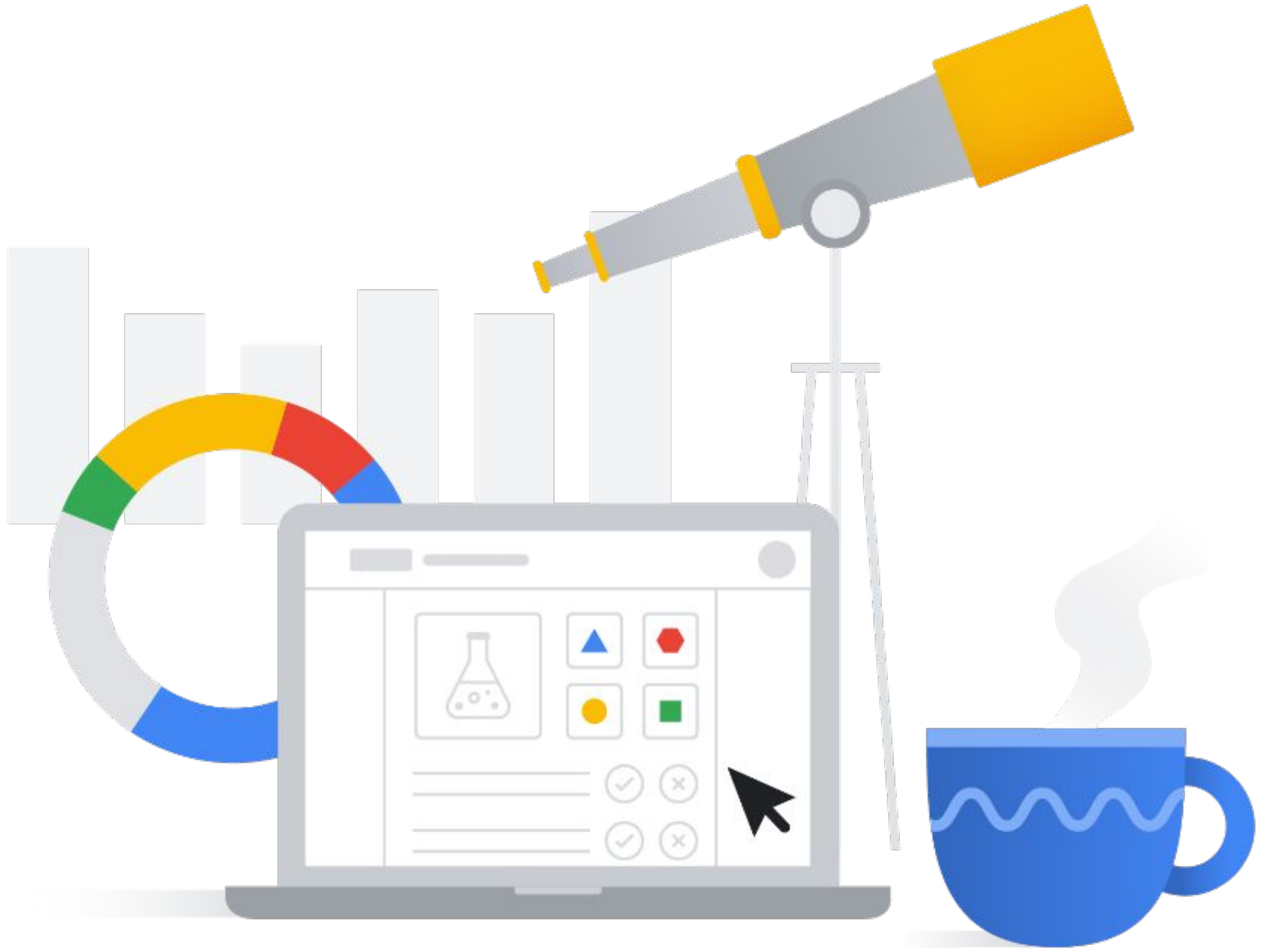
A. Configure or migrate your cluster to VPC-Native Mode and deploy a container-native load balancer. ✓

B. Set the externalTrafficPolicy field to local in the YAML manifest for your external services.

C. Configure all external access for your application using Ingress resources rather than services.

D. Configure affinity and anti-affinity rules that ensure your application's Pods are distributed across nodes.

Lab: Configuring Google Kubernetes Engine Networking



01 Configure a cluster for authorized network control plane access.

02 Configure a Cluster network policy.