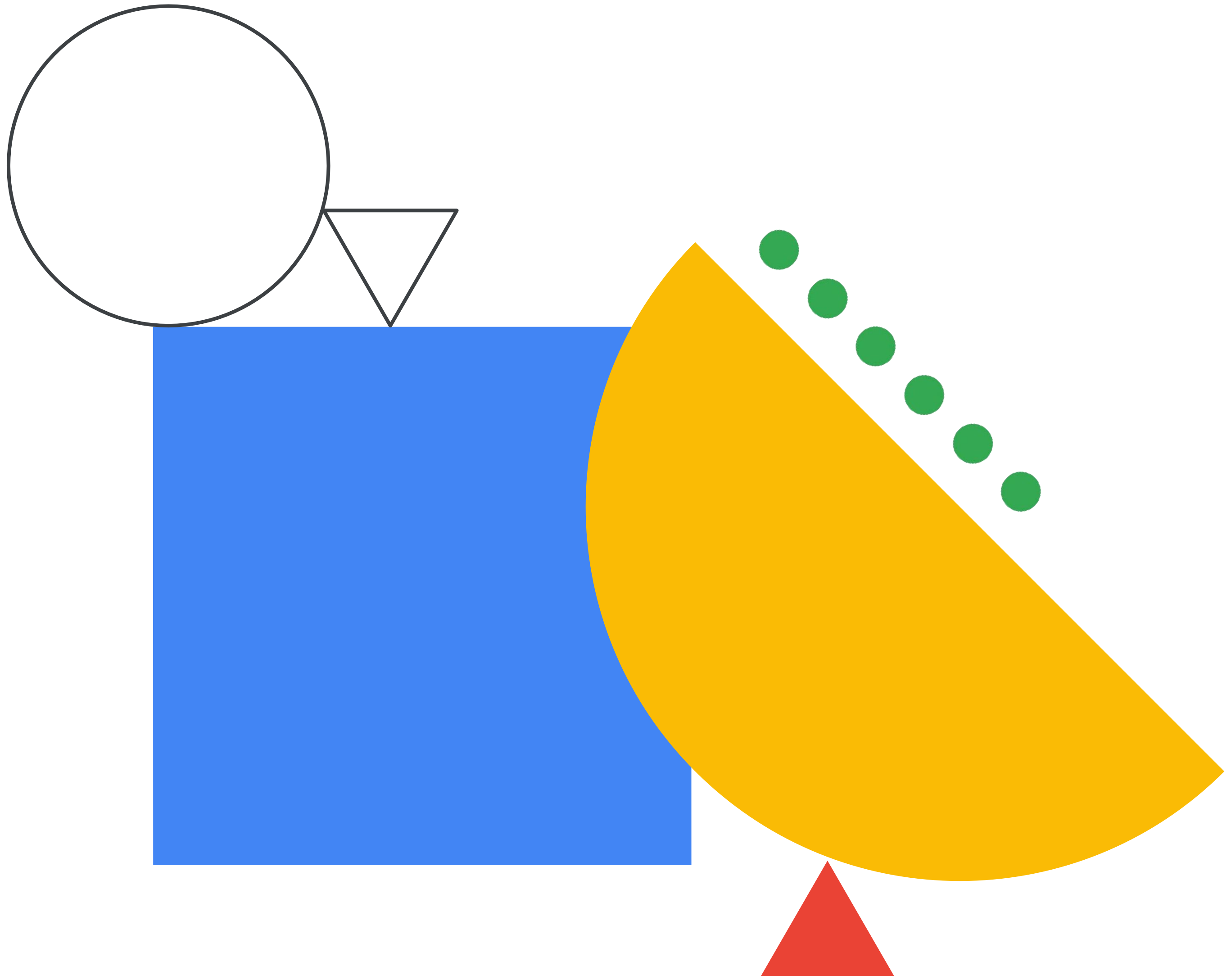
Google Cloud

Microservice Design and Architecture



The following course materials are copyright protected materials.

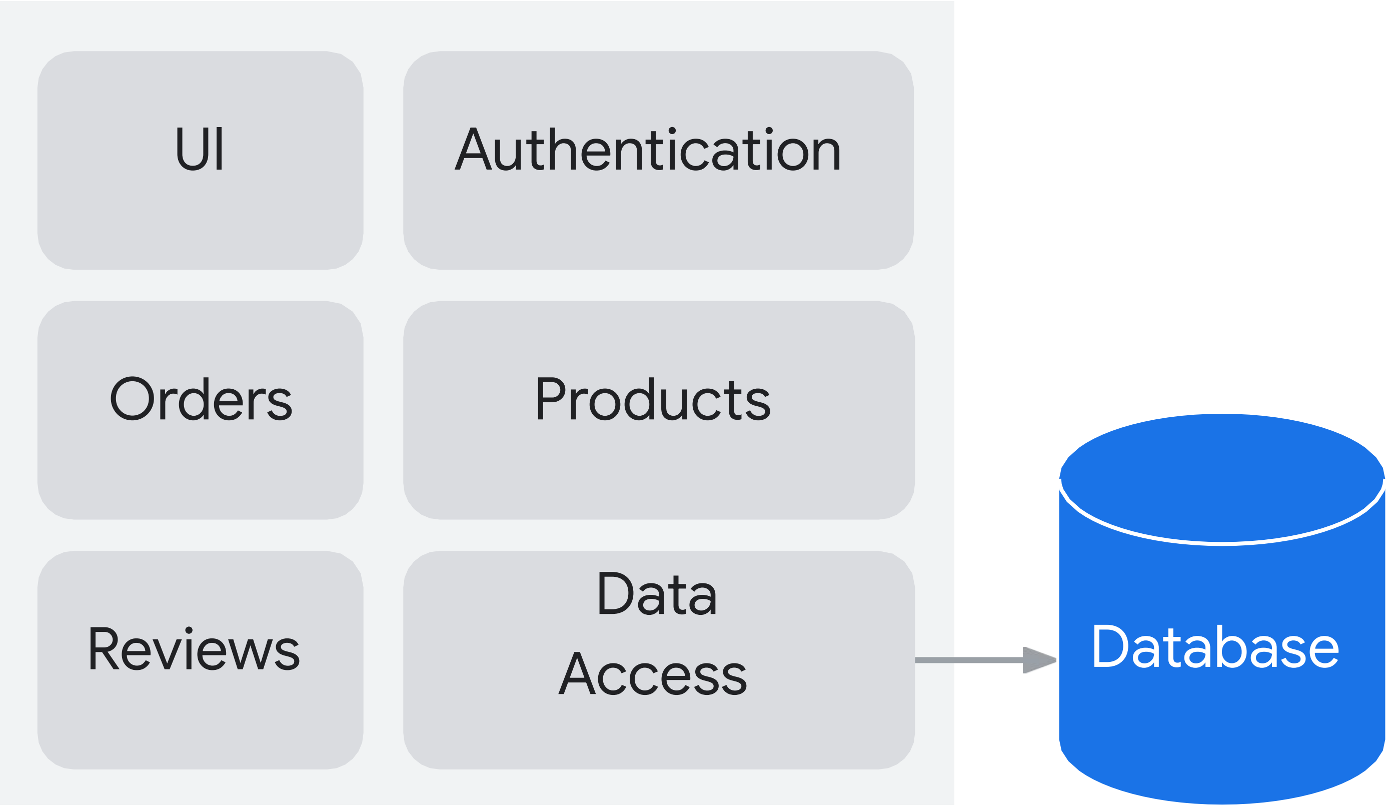
They may not be reproduced or distributed and may only be used by students attending this Google Cloud Partner Learning Services program.

01

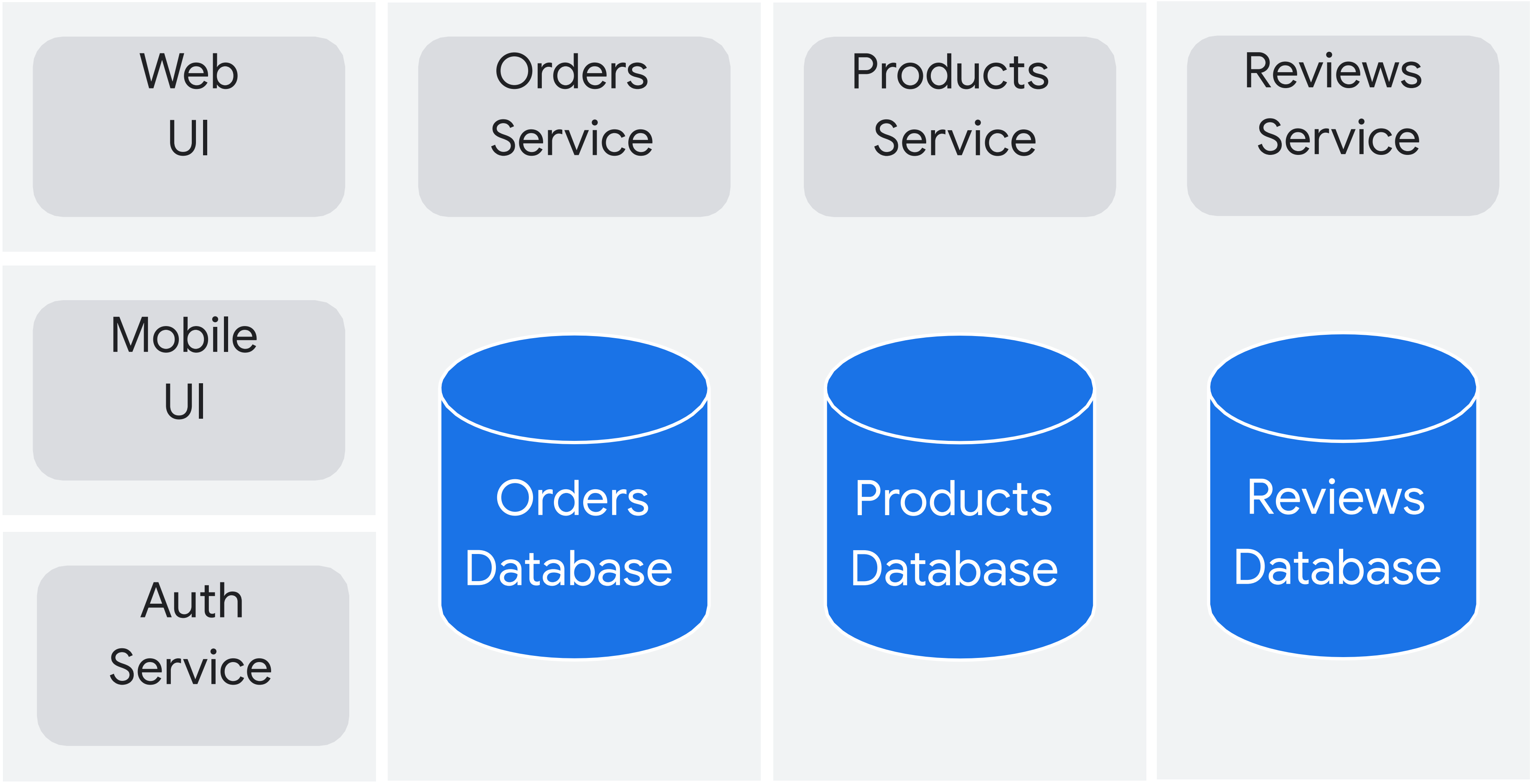
Microservices

Microservices divide a large program into multiple smaller, independent services

Monolithic applications implement all features in a single code base with a database for all data.



Microservice have multiple code bases, and each service manages its own data.



Pros and cons of microservice architectures

Pros

- Easier to develop and maintain.
- Reduced risk when deploying new versions.
- Services scale independently to optimize use of infrastructure.
- Faster to innovate and add new features.
- Can use different languages and frameworks for different services.
- Choose the runtime appropriate to each service.

Con

s

- Increased complexity when communicating between services.
- Increased latency across service boundaries.
- Services scale independently to optimize use of infrastructure.
- Concerns about securing inter-service traffic.
- Multiple deployments.
- Need to ensure that you don’t break clients as versions change.
- Must maintain backward compatibility with clients as the microservice evolves.

The key to architecting microservice applications is recognizing service boundaries

01

Decompose applications by feature to minimize dependencies

- Reviews service
- Orders service
- Products service
- Etc.

02

Organize services by architectural layer

- Web, Android, and iOS user interfaces
- Data access services

03

Isolate services that provide shared functionality

- Authentication service
- Reporting service
- Etc.

Microservice Best Practices

The 12-factor app is a set of best practices for building web or software-as-a-service applications

- Maximize portability
- Deploy to the cloud
- Enable continuous deployment
- Scale easily

THE TWELVE-FACTOR APP

The 12 factors

01

Codebase

One codebase tracked in revision control, many deploys

- Use a version control system like Git.
- Each app has one code repo and vice versa.

02

Dependencies

Explicitly declare and isolate dependencies

- Use a package manager like Maven, Pip, NPM to install dependencies.
- Declare dependencies in your code base.

03

Config

Store config in the environment

- Don't put secrets, connection strings, endpoints, etc., in source code.
- Store those as environment variables.

04

Backing Services

Treat backing services as attached resources

- Databases, caches, queues, and other services are accessed via URLs.
- Should be easy to swap one implementation for another.

The 12 factors (continued)

05

Build, release, run

Strictly separate build and run stages

- Build creates a deployment package from the source code.
- Release combines the deployment with configuration in the runtime environment.
- Run executes the application.

06

Processes

Execute the app as one or more stateless processes

- Apps run in one or more processes.
- Each instance of the app gets its data from a separate database service.

07

Port binding

Export services via port binding

- Apps are self-contained and expose a port and protocol internally.
- Apps are not injected into a separate server like Apache.

08

Concurrency

Scale out via the process model

- Because apps are self-contained and run in separate process, they scale easily by adding instances.

The 12 factors (continued)

09

Disposability

Maximize robustness with fast startup and graceful shutdown

- App instances should scale quickly when needed.
- If an instance is not needed, you should be able to turn it off with no side effects.

10

Dev/prod parity

Keep development, staging, and production as similar as possible

- Container systems like Docker makes this easier.
- Leverage infrastructure as code to make environments easy to create.

11

Logs

Treat logs as event streams

- Write log messages to standard output and aggregate all logs to a single source.

12

Admin processes

Run admin/management tasks as one-off processes

- Admin tasks should be repeatable processes, not one-off manual tasks.
- Admin tasks shouldn't be a part of the application.

03

REST

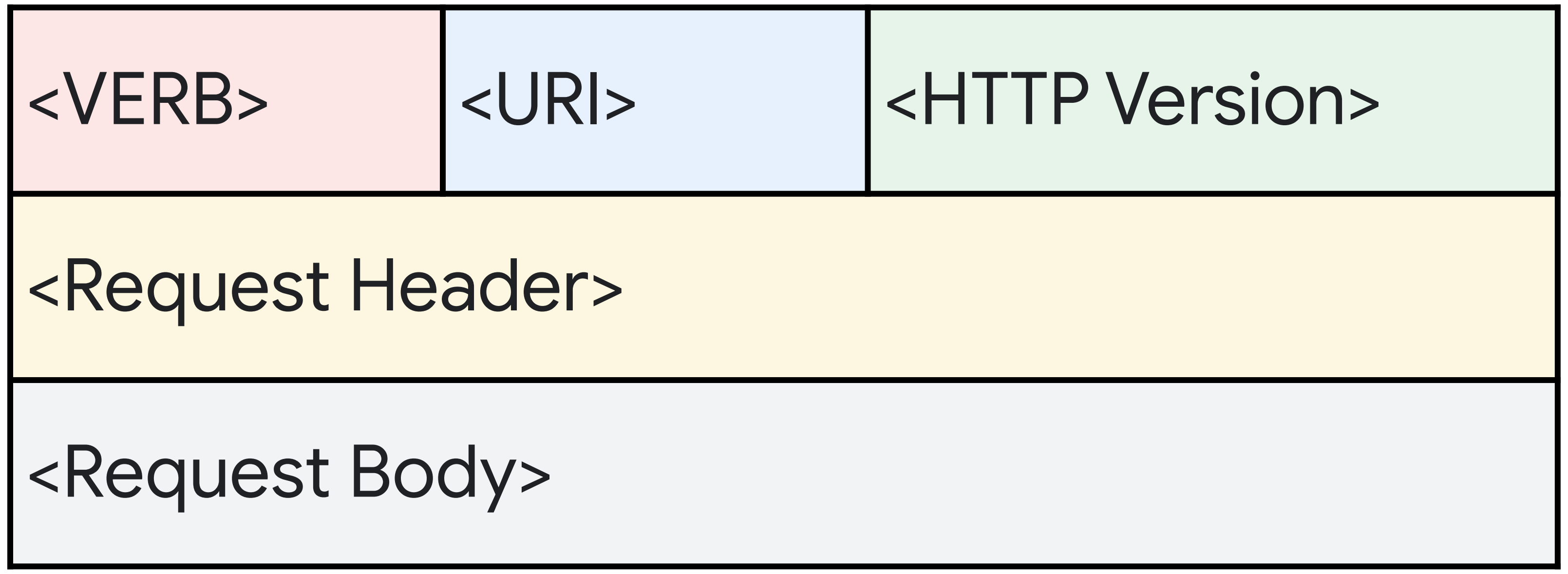
A good microservice design is loosely coupled

- Clients shouldn't need to know too many details of services they use
- Services communicate via HTTPS using text-based payloads
  - Client makes GET, POST, PUT, or DELETE request
  - Body of the request is formatted as JSON or XML
  - Results returned as JSON, XML, or HTML
- Services should add functionality without breaking existing clients
  - Add, but don't remove, items from responses

If microservices aren't loosely coupled, you'll end up with a really complicated monolith.

Clients access services using HTTP requests

- VERB: GET, PUT, POST, DELETE
- URI: Uniform Resource Identifier (endpoint)
- Request Header: metadata about the message
  - Preferred representation formats (e.g., JSON, XML)
- Request Body: (Optional) Request state
  - Representation (JSON, XML) of resource



<VERB> <URI> <HTTP Version>

<Request Header>

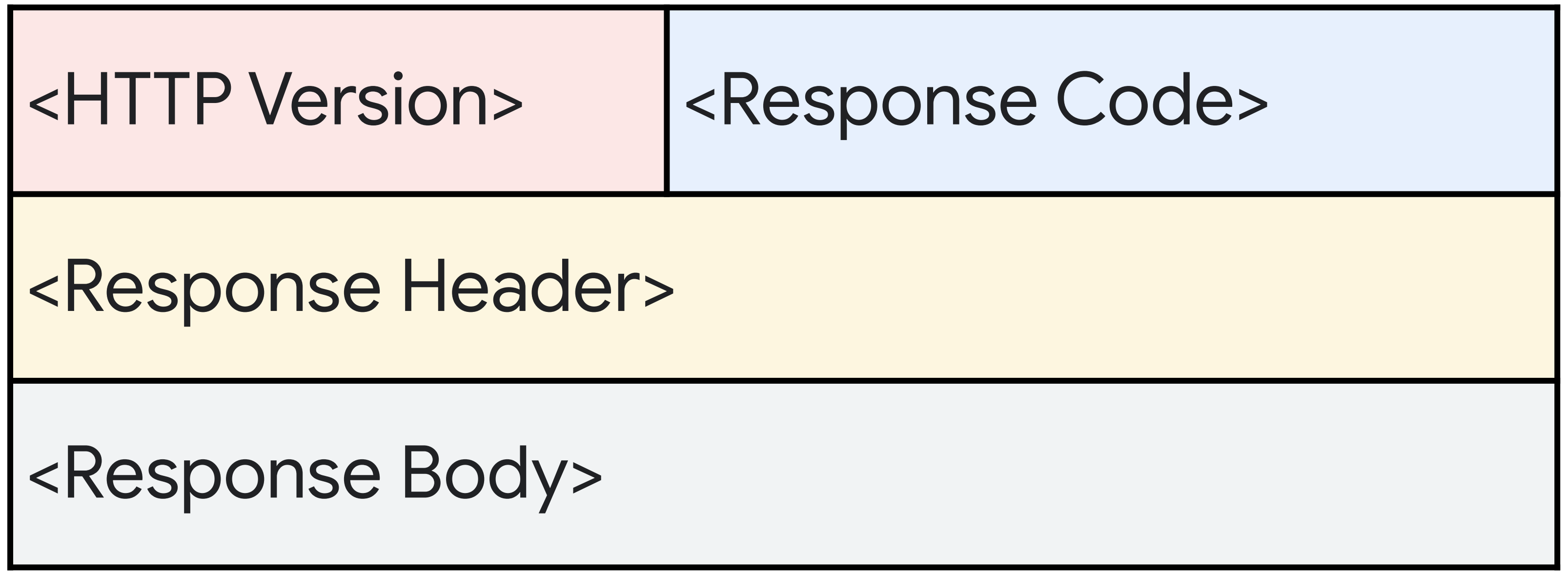
<Request Body>

The HTTP verb tells the server what to do

- **GET** is used to retrieve data
- **POST** is used to create data
  - Generates entity ID and returns it to the client
- **PUT** is used to create data or alter existing data
  - Entity ID must be known
  - *PUT should be* idempotent, *which means that whether the request is made once or multiple times, the effects on the data are exactly the same*
- **DELETE** is used to remove data

Services return HTTP responses

- Response Code: 3-digit HTTP status code
  - 200 codes for success
  - 400 codes for client errors
  - 500 codes for server errors
- Response Body: contains resource representation
  - JSON, XML, HTML, etc.



<HTTP Version> <Response Code>

<Response Header>

<Response Body>

04

APIs

It’s important to design consistent APIs for services

- Each Google Cloud service exposes a REST API
  - Functions are in the form: service.collection.verb
  - Parameters are passed either in the URL or in the request body in JSON format
- For example, the Compute Engine API has...
  - A service endpoint at: https://compute.googleapis.com
  - Collections include instances, instanceGroups, instanceTemplates, etc.
  - Verbs include insert, list, get, etc.
- So, to see all your instances, make a GET request to:

  https://compute.googleapis.com/compute/v1/projects/{project}/zones/{zone}/instances

Google Cloud provides two tools, Cloud Endpoints and Apigee, for managing APIs

Both provide tools for:

- User authentication
- Monitoring
- Securing APIs
- Etc.

Both support OpenAPI and gRPC

Cloud Endpoints

Apigee API Platform

More resources

API Design Guide

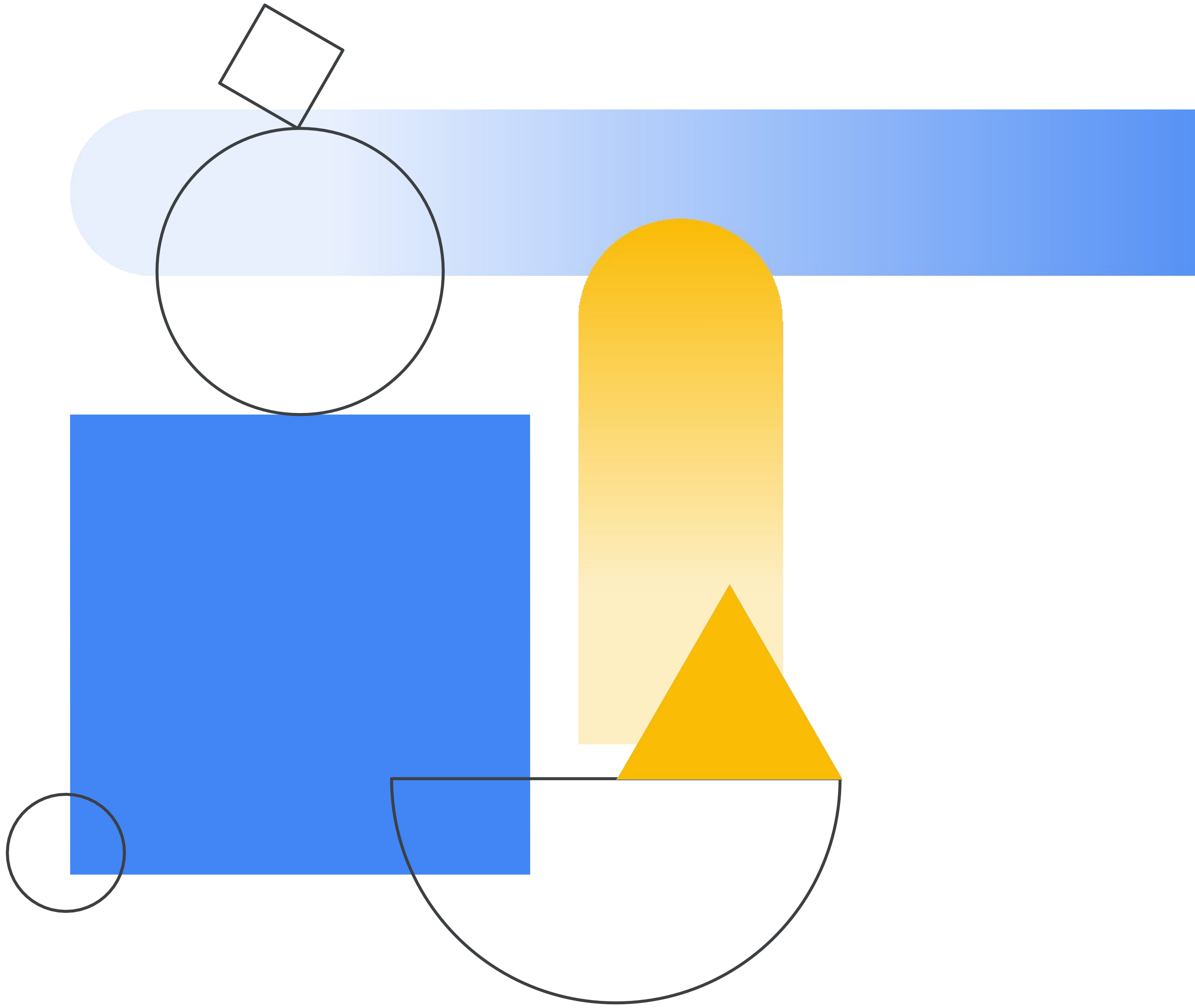
https://cloud.google.com/apis/design/

Authenticating service-to-service calls with Google Cloud Endpoints

https://youtu.be/4PgX3yBJEyw

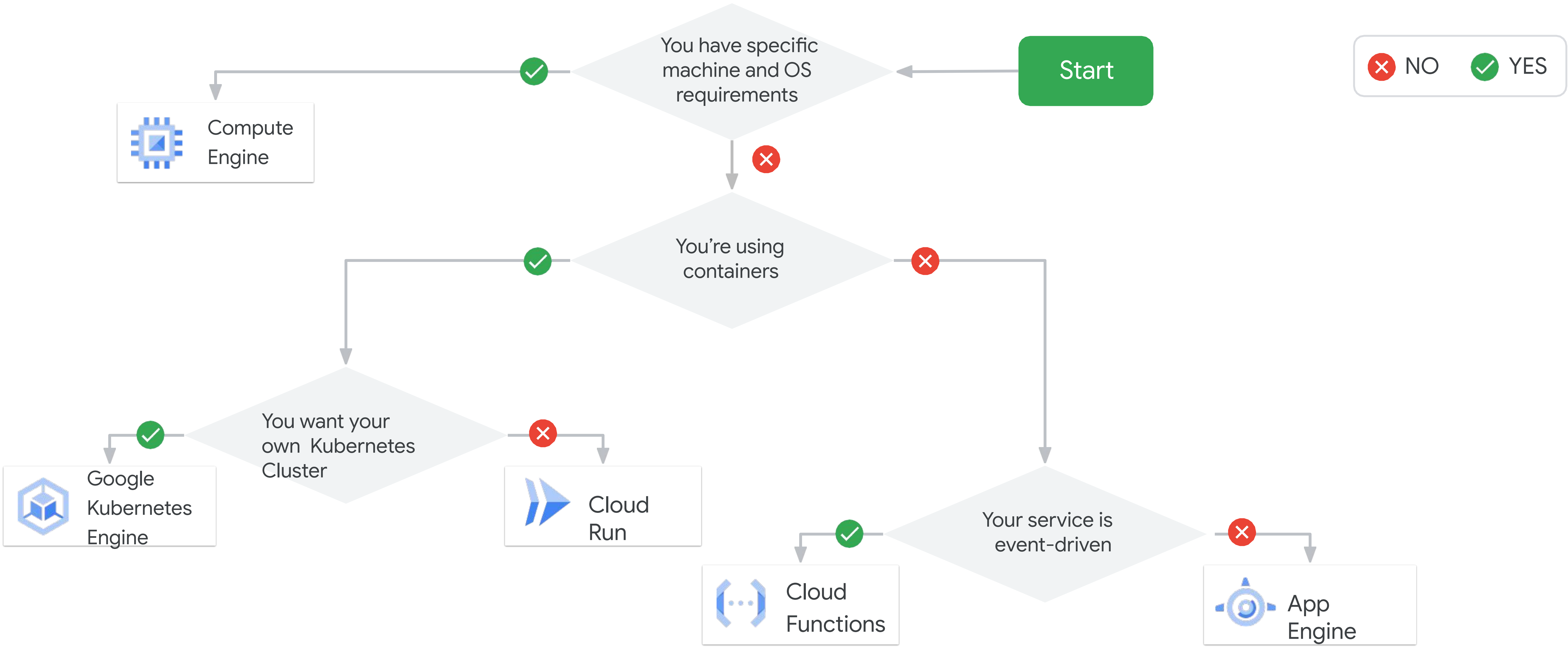
Google Cloud

Deploying Applications to Google Cloud

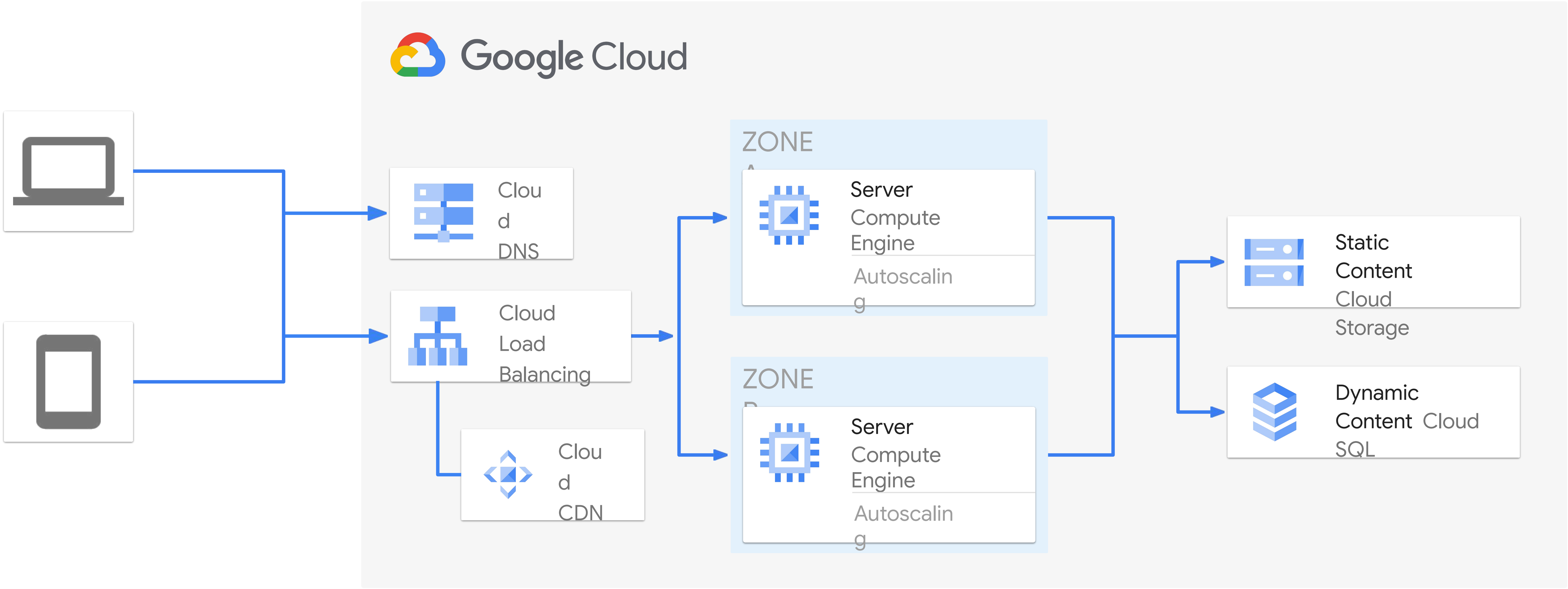


Google Cloud

Choosing a Google Cloud deployment platform


Start
You have specific machine and OS requirements
Compute Engine
You're using containers
You want your own Kubernetes Cluster
Google Kubernetes Engine
Cloud Run
Your service is event-driven
Cloud Functions
App Engine
✗ NO
✓ YES

Use Compute Engine when you need complete control over operating systems, for apps that are not containerized or self-hosted databases



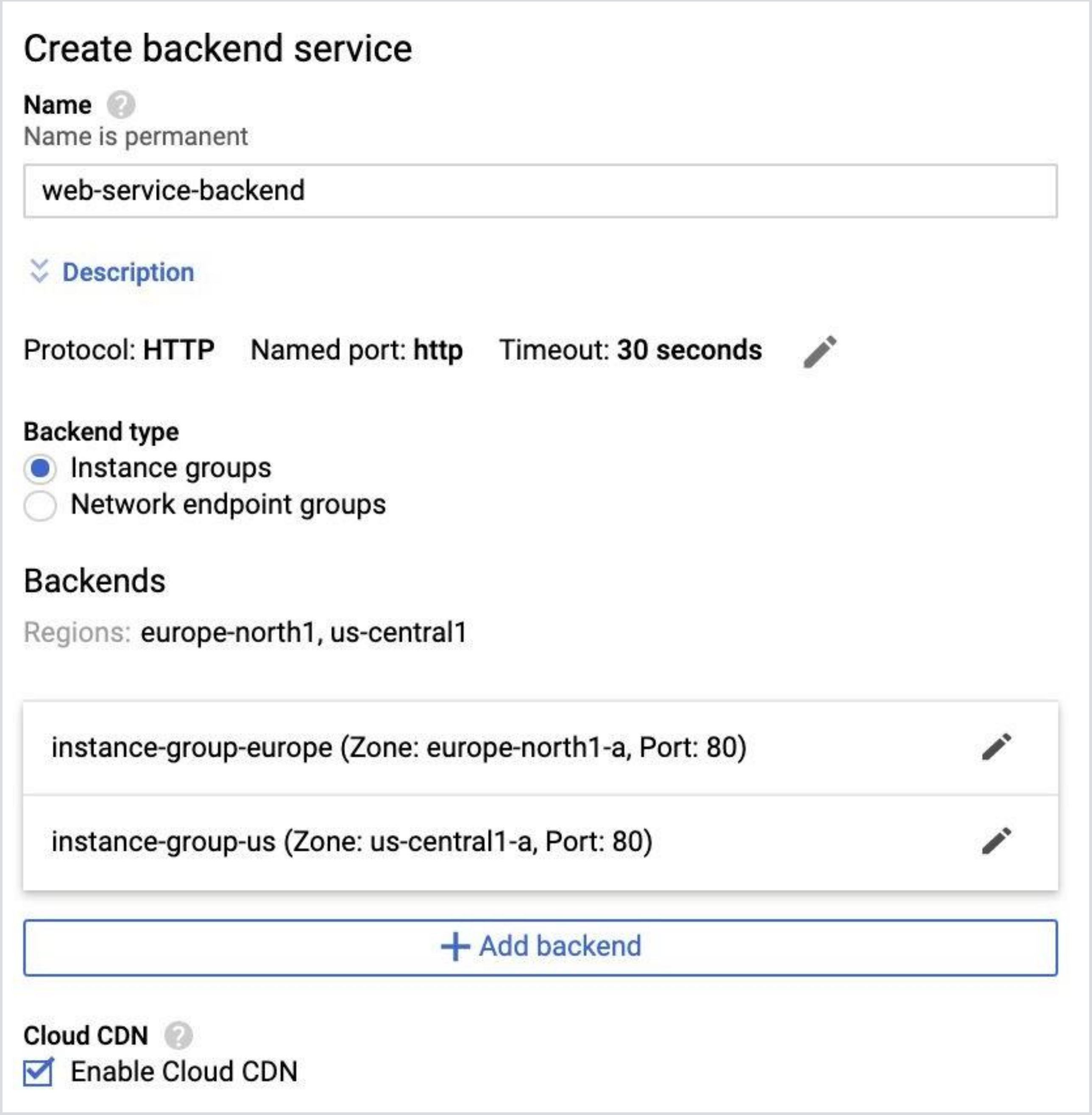
Managed instance groups create VMs based on instance templates

- Instance templates define the VMs: image, machine type, etc.
  - Test to find the smallest machine type that will run your program.
- Use a Startup Script to install your program from a Git repo.

- Instance group manager creates the machines.
- Set up auto scaling to optimize cost and meet varying user workloads.
- Add a health check to enable auto healing.
- Use multiple zones for high availability.

Use one or more instance groups as the backend for load balancers

- Use a global load balancer if you have instance groups in multiple regions.
- Enable the CDN to cache static content.
- For external services, set up SSL.
- For internal services, don't provide a public IP address.



Create backend service

Name
Name is permanent
web-service-backend

Description

Protocol: **HTTP** Named port: **http** Timeout: **30 seconds**

Backend type
● Instance groups
○ Network endpoint groups

Backends
Regions: europe-north1, us-central1

instance-group-europe (Zone: europe-north1-a, Port: 80)

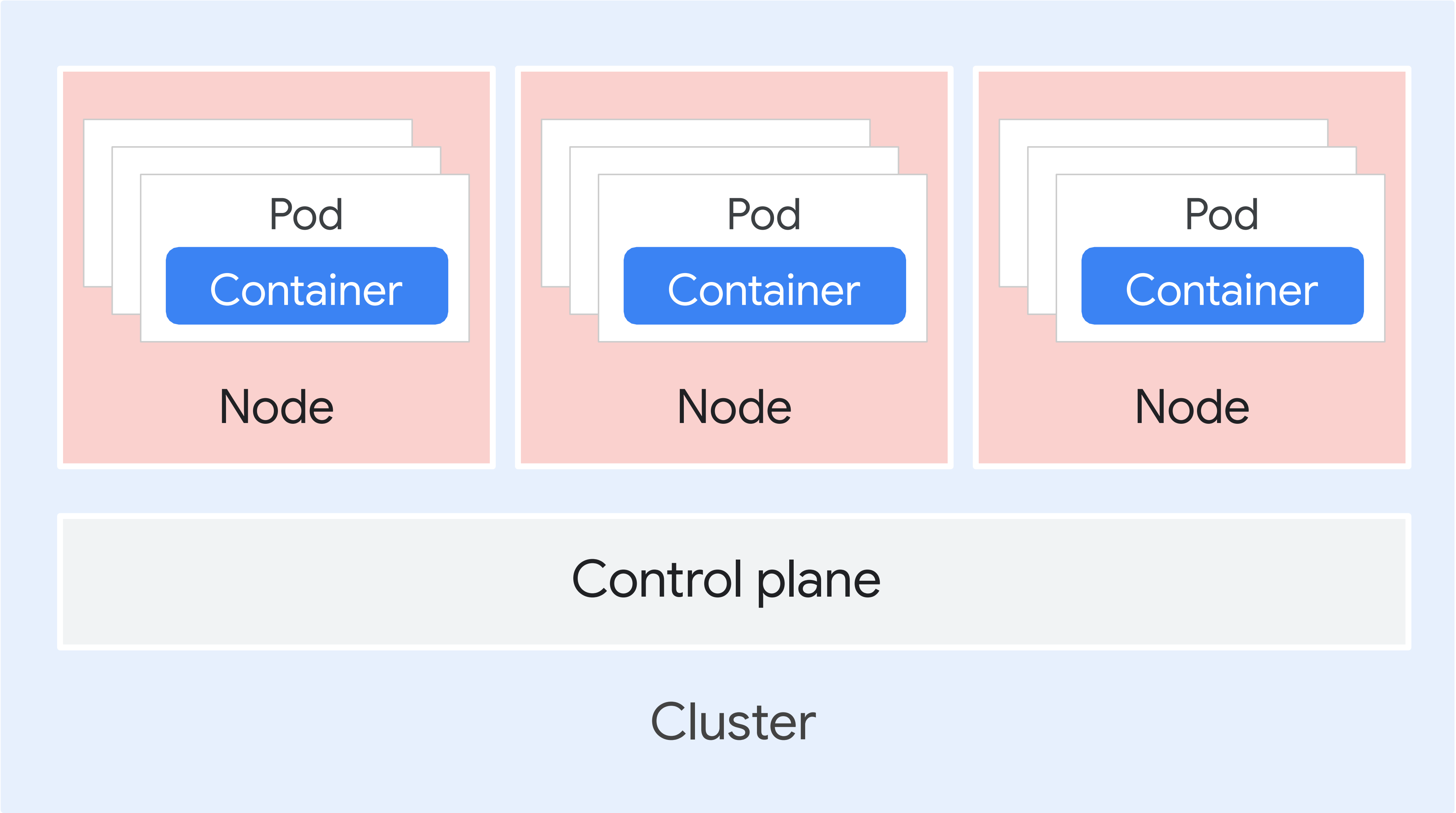
instance-group-us (Zone: us-central1-a, Port: 80)

Cloud CDN
☑ Enable Cloud CDN

Google Cloud Deployment Platforms

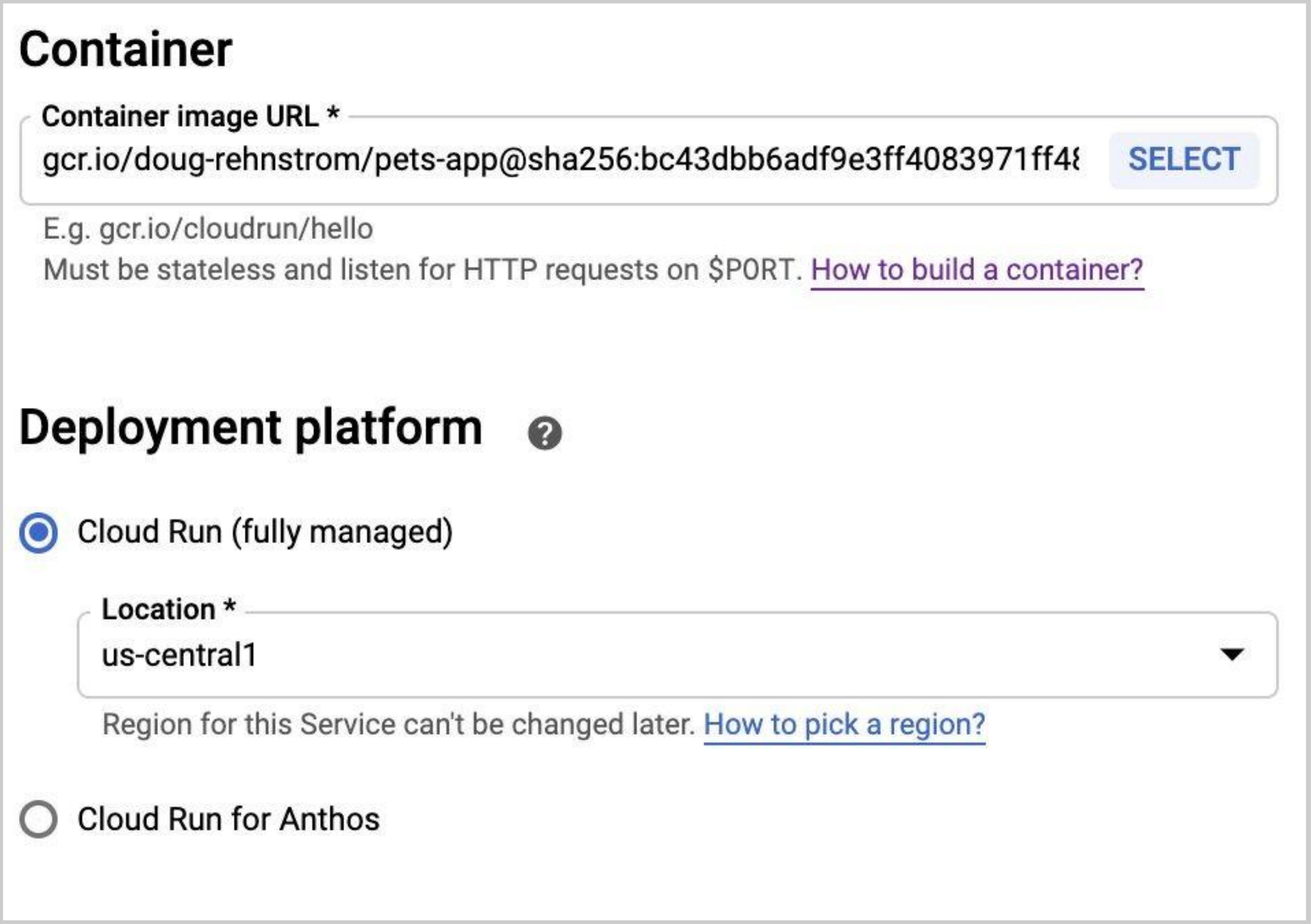
Google Kubernetes Engine (GKE) automates the creation and management of compute infrastructure

- Kubernetes clusters have a collection of nodes.
- In GKE, nodes are Compute Engine VMs.
- Services are deployed into pods.
- Optimize resource utilization by deploying multiple services to the same cluster.
- You pay for the VMs.



Cloud Run allows you to deploy containers to Google-managed Kubernetes clusters

- Cloud Run allows you to use Kubernetes without the cluster management or configuration code.
- Apps must be stateless.
- Need to deploy apps using Docker images in Container Registry.
- Can also use Cloud Run to automate deployment to your own GKE cluster.



Container

Container image URL *
gcr.io/doug-rehnstrom/pets-app@sha256:bc43dbb6adf9e3ff4083971ff4ɛ

E.g. gcr.io/cloudrun/hello
Must be stateless and listen for HTTP requests on $PORT. How to build a container?

Deployment platform

◉ Cloud Run (fully managed)

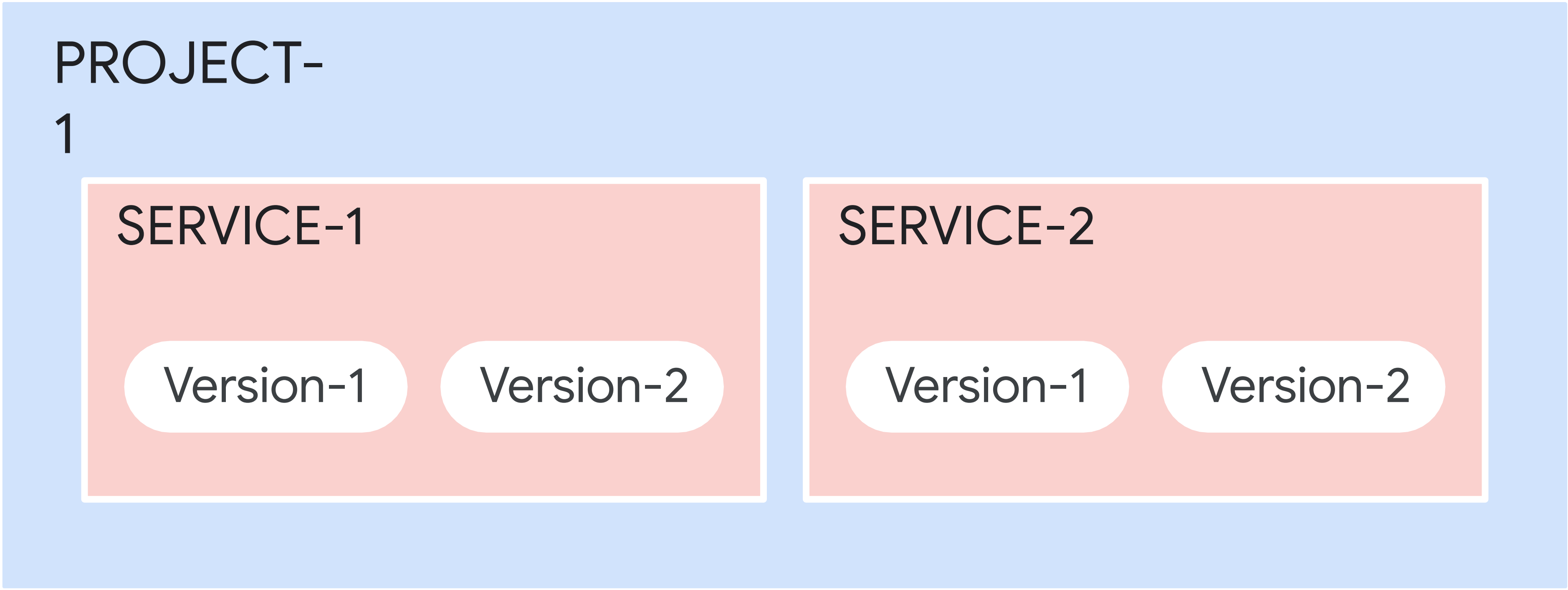
Location *
us-central1

Region for this Service can't be changed later. How to pick a region?

○ Cloud Run for Anthos

App Engine was designed for microservices

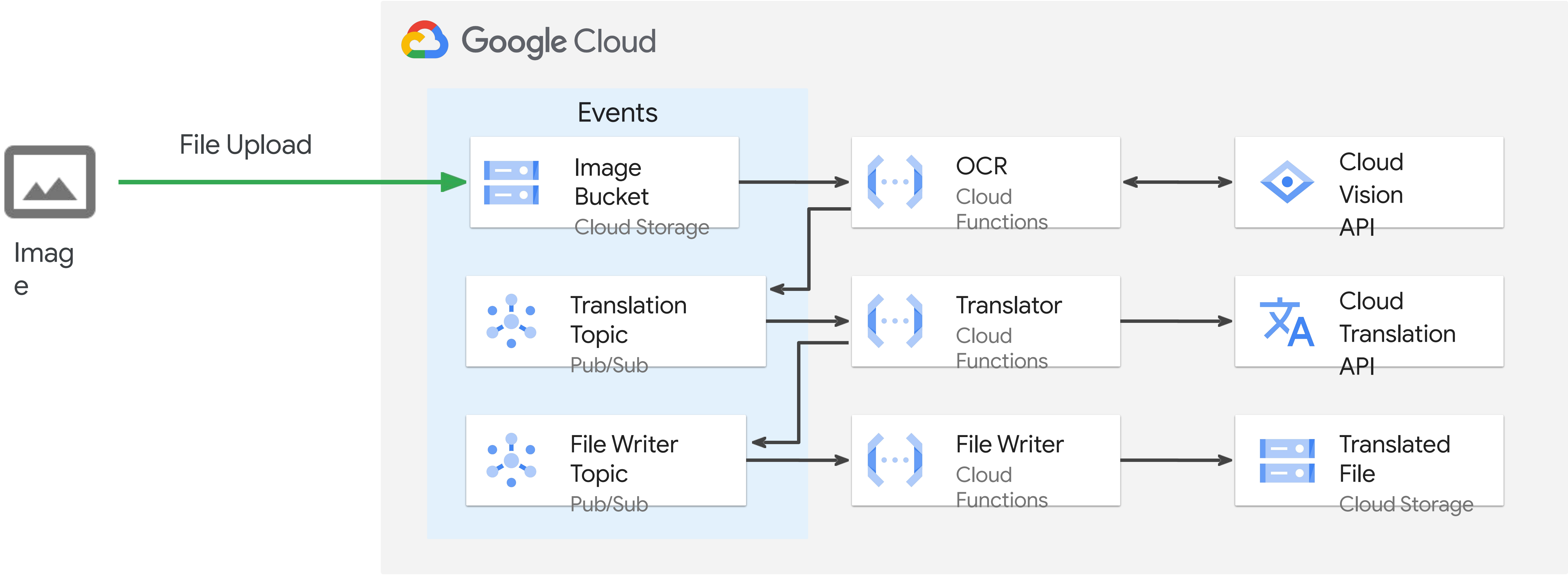
- Each Google Cloud project can contain 1 App Engine application.
- An application has 1 or more services.
- Each service has 1 or more versions.
- Versions have 1 or more instances.
- Automatic traffic splitting for switching versions.



Proprietary + Confidential

Cloud Functions is great way to create loosely coupled, event-driven microservices

- Can be triggered by changes in a storage bucket, Pub/Sub messages, or web requests
- Completely managed, scalable, and inexpensive



Google Cloud

More resources

Migration to Google Cloud:
Deploying your workloads

https://cloud.google.com/solutions/migration-to-gcp-deploying-your-workloads

Compute Engine

https://cloud.google.com/compute/

GKE

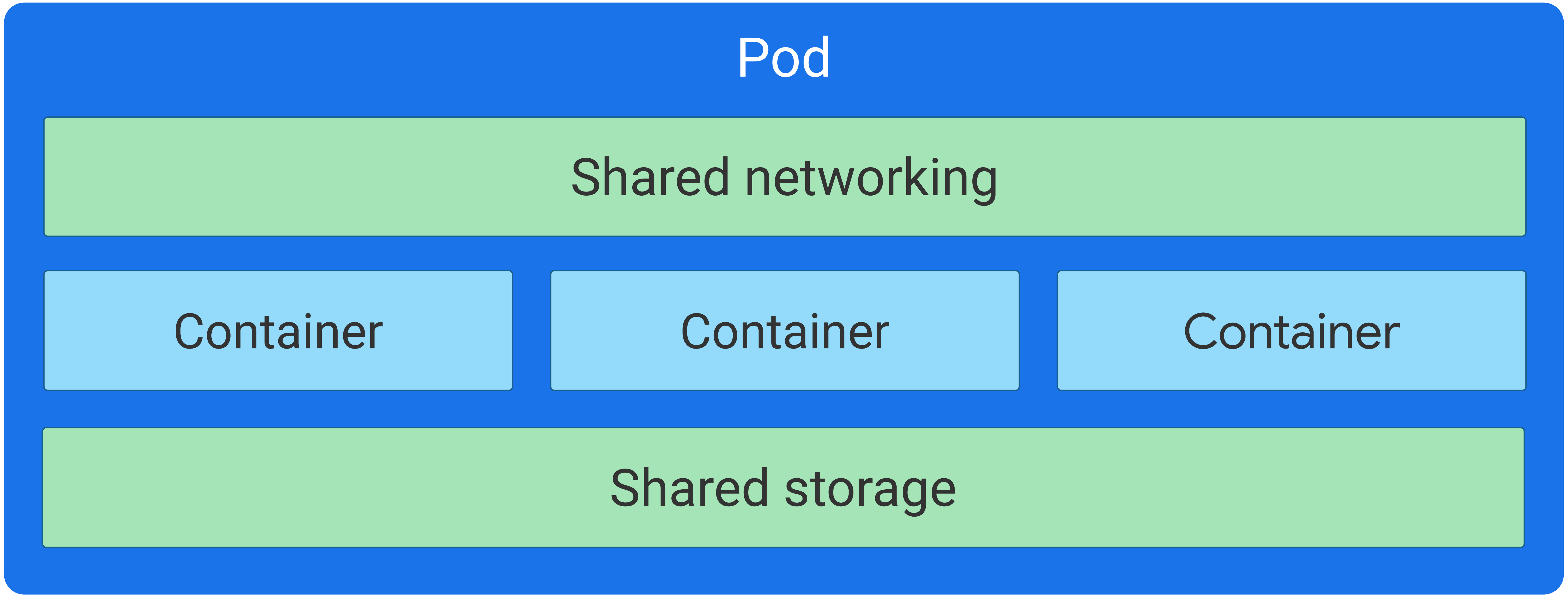
https://cloud.google.com/kubernetes-engine/

App Engine

https://cloud.google.com/appengine/

Kubernetes Architecture

Containers in a Pod share resources



Pod

Shared networking

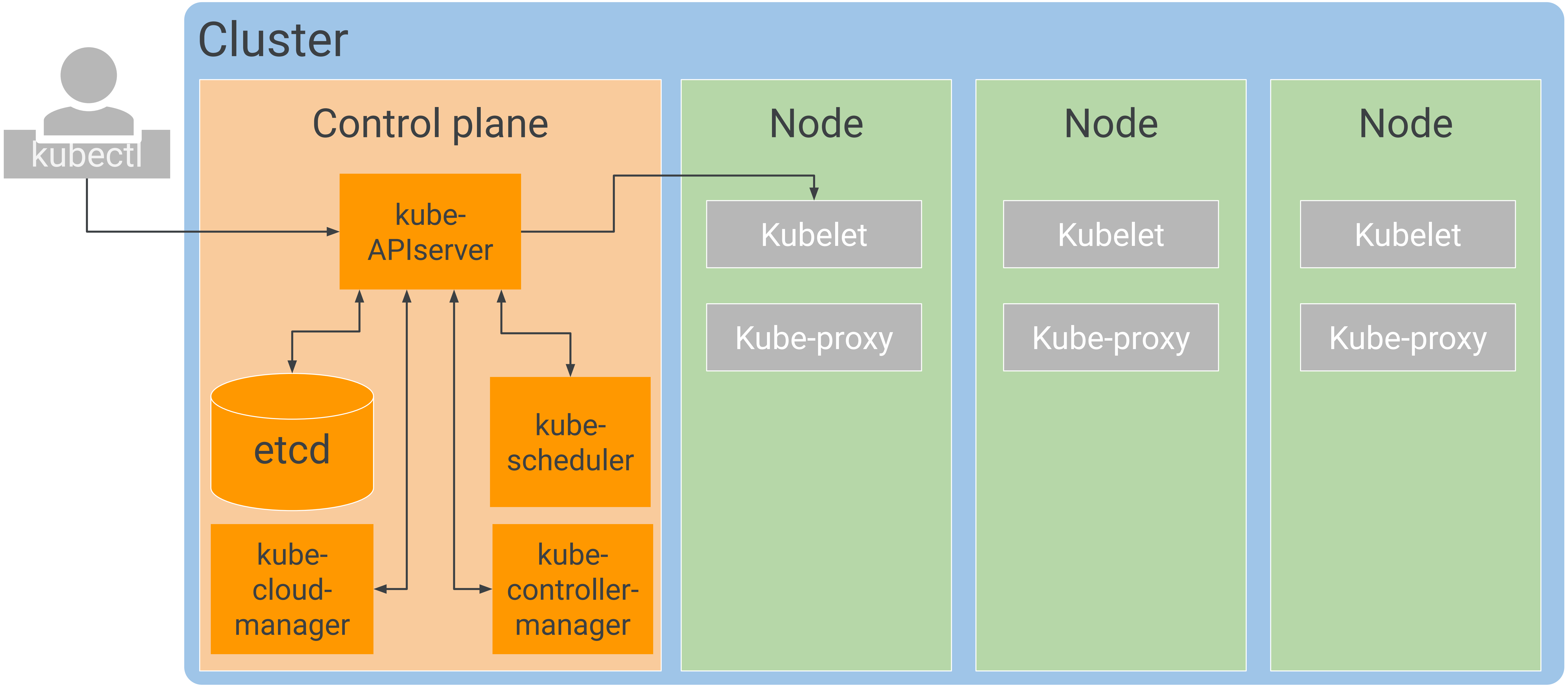
Container

Container

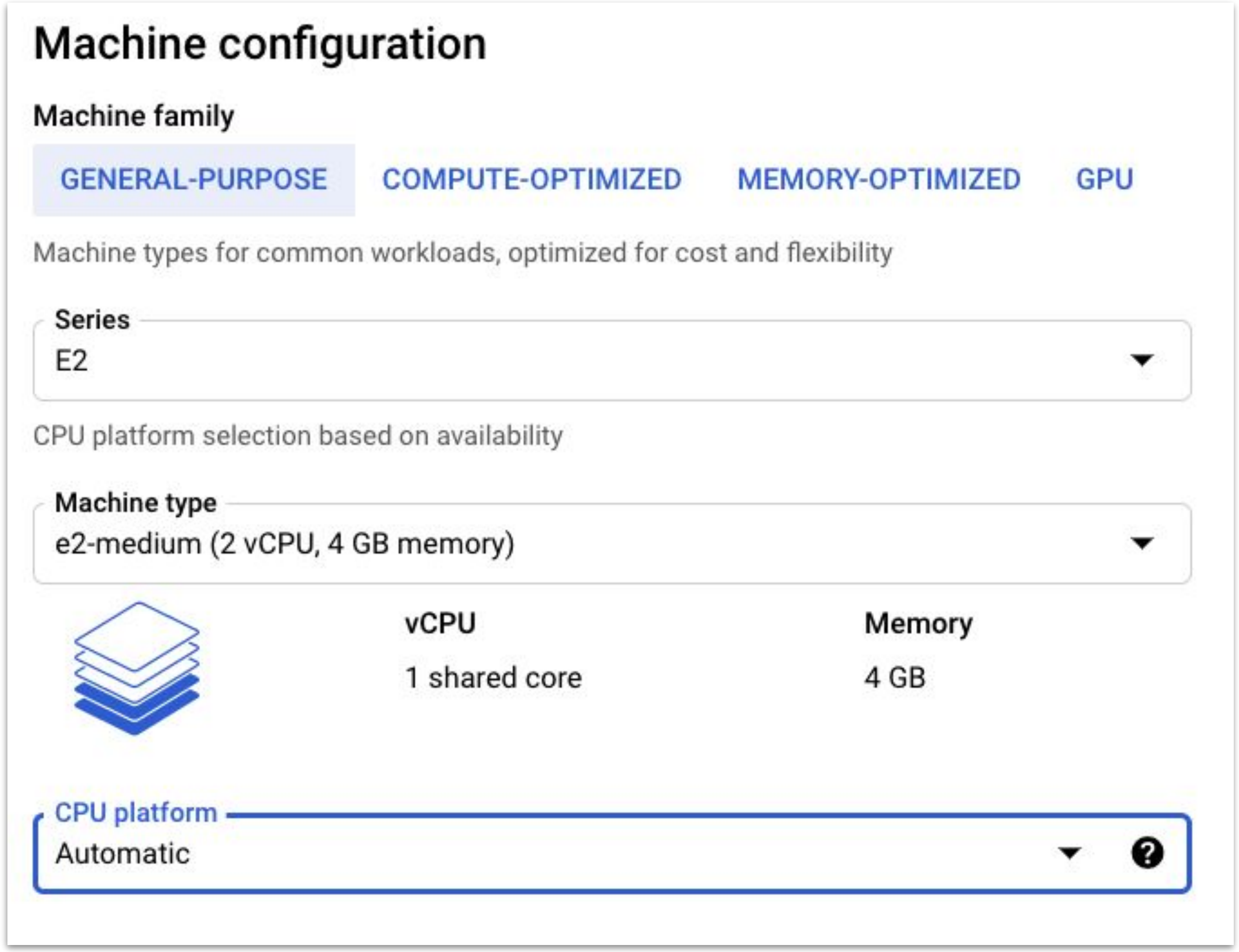
Container

Shared storage

Cooperating processes make a Kubernetes cluster work



GKE: More about nodes



Machine configuration

Machine family

Machine types for common workloads, optimized for cost and flexibility

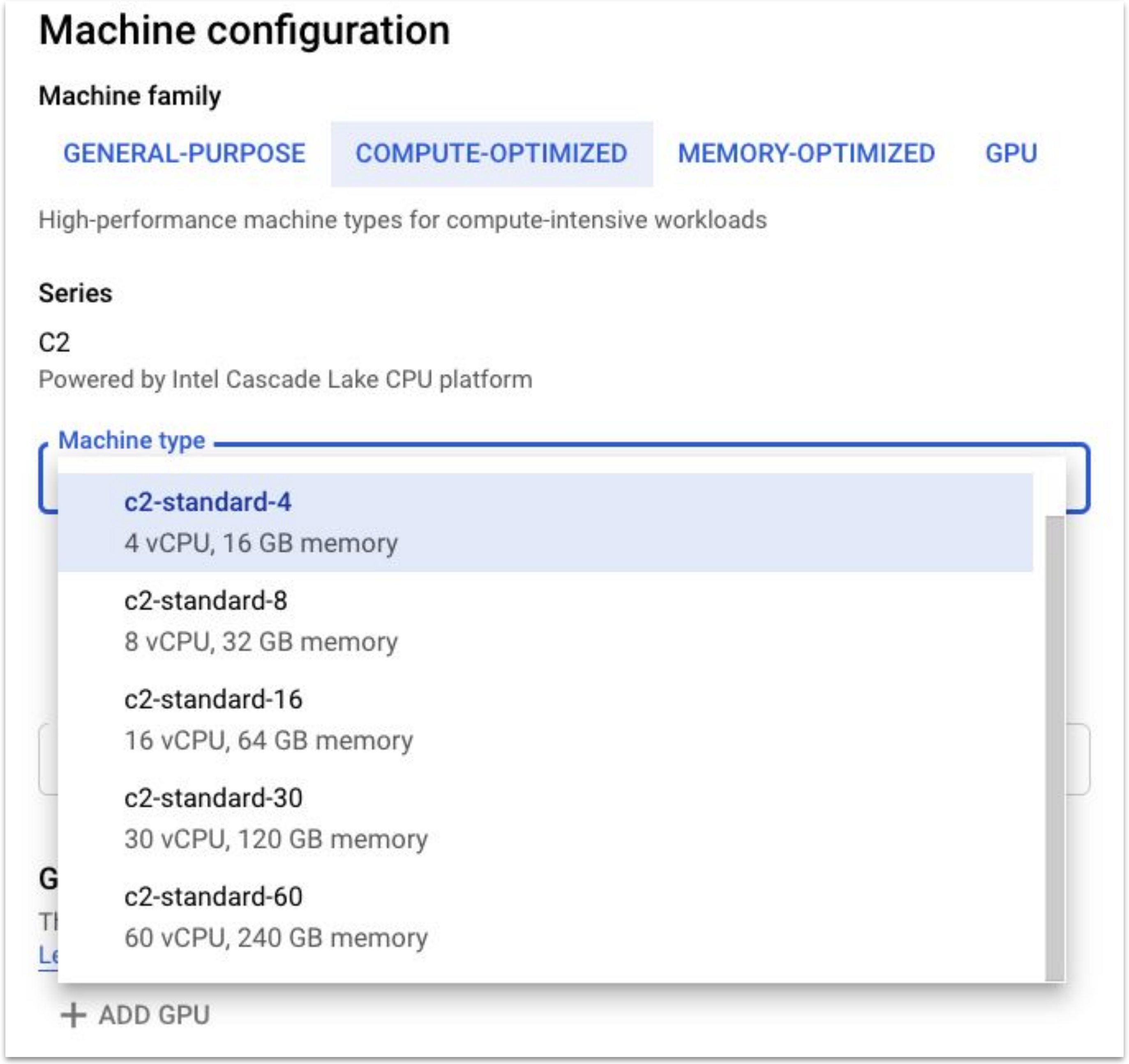
Series: E2

CPU platform selection based on availability

Machine type: e2-medium (2 vCPU, 4 GB memory)

vCPU	Memory
1 shared core	4 GB

CPU platform: Automatic



Machine configuration

Machine family

High-performance machine types for compute-intensive workloads

Series

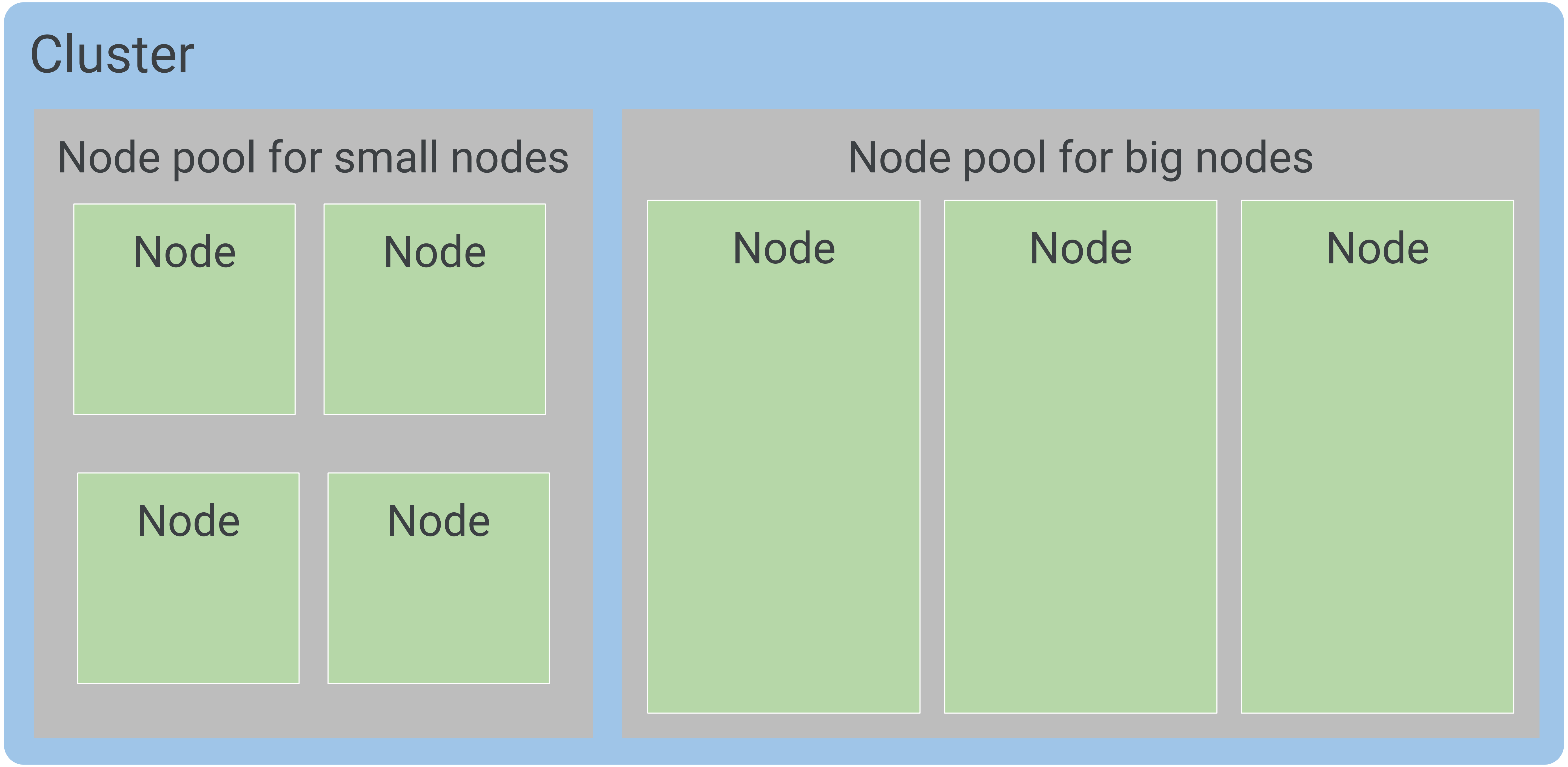
C2

Powered by Intel Cascade Lake CPU platform

Machine type

- c2-standard-4
  4 vCPU, 16 GB memory
- c2-standard-8
  8 vCPU, 32 GB memory
- c2-standard-16
  16 vCPU, 64 GB memory
- c2-standard-30
  30 vCPU, 120 GB memory
- c2-standard-60
  60 vCPU, 240 GB memory

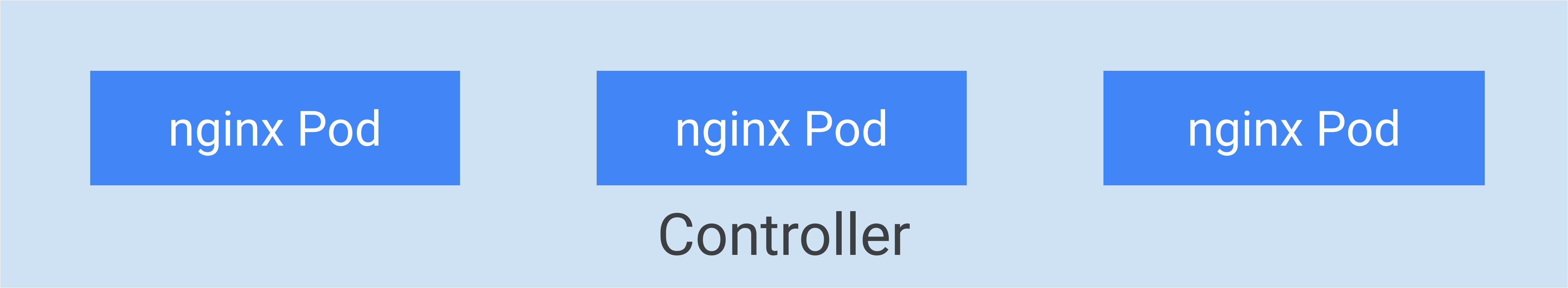
Use node pools to manage different kinds of nodes



Objects are defined in a YAML file

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name:  nginx
    image: nginx:latest
```

Pods and Controller Objects



nginx Pod

nginx Pod

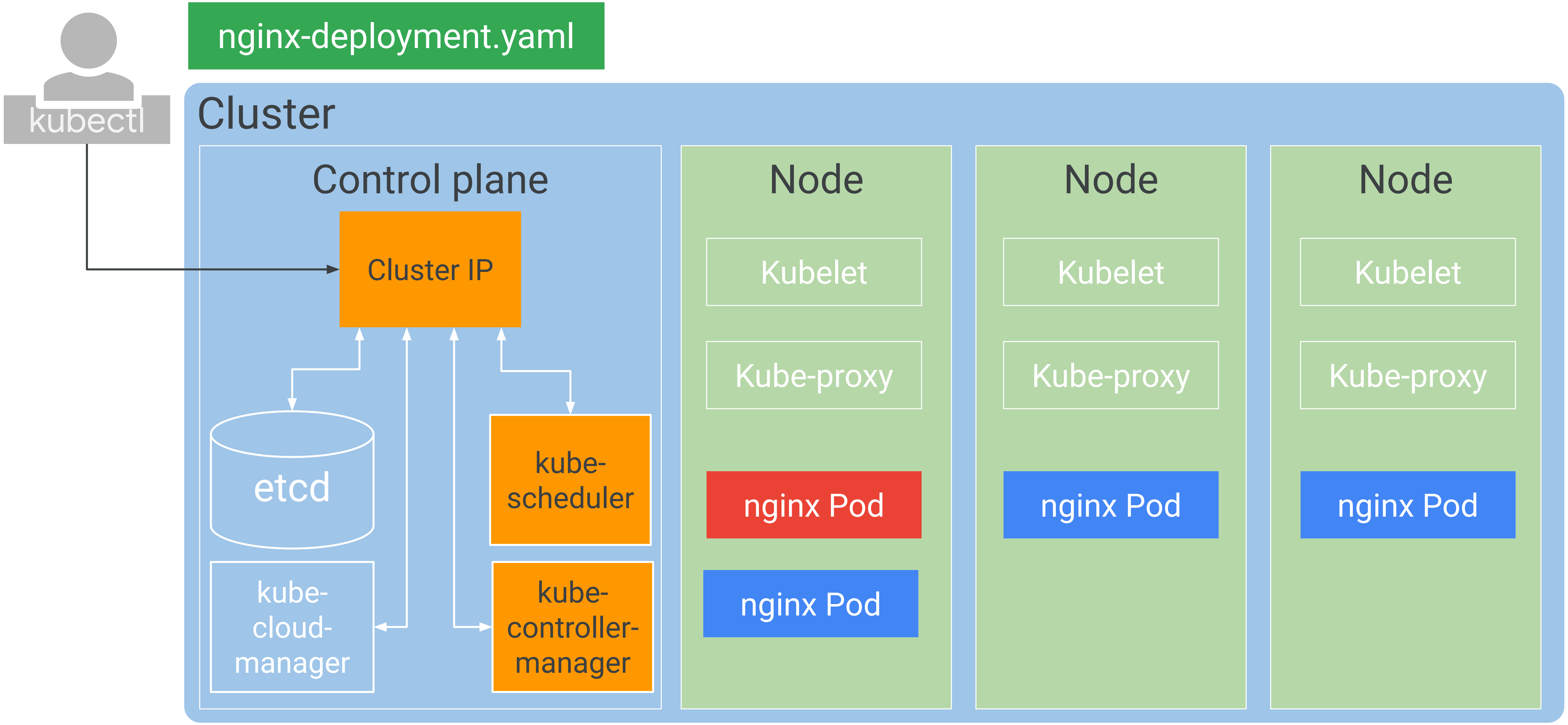
nginx Pod

Controller

Controller object types

- Deployment
- StatefulSet
- DaemonSet
- Job

A Deployment maintains the desired state



Deployments ensure that sets of Pods are running

```
apiVersion: apps/v1
kind: Deployment
metadata:
   name: nginx-deployment
   labels:
     app: nginx
spec:
   replicas: 3
   selector:
     matchLabels:
        app: nginx
   template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```

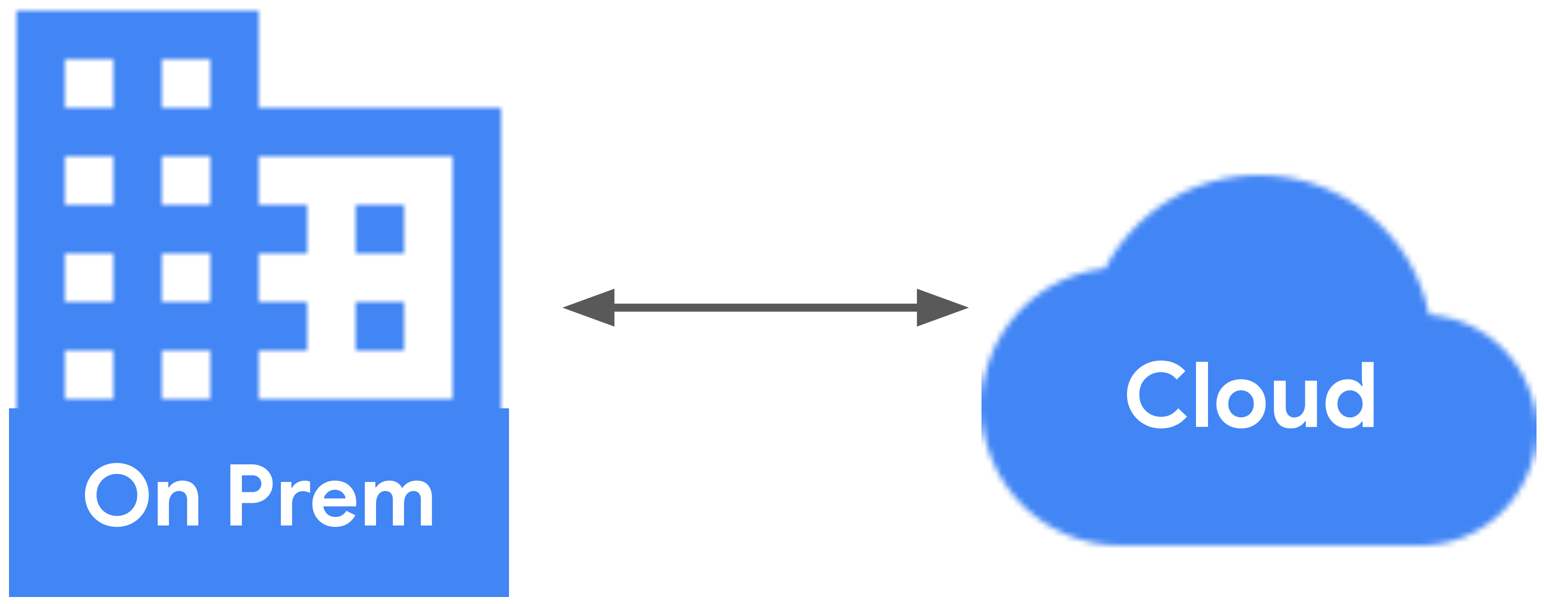
Google Cloud

Introducing Anthos

Hybrid Cloud Overview

- Companies would like to adopt the cloud in their own pace, alongside their on premise.

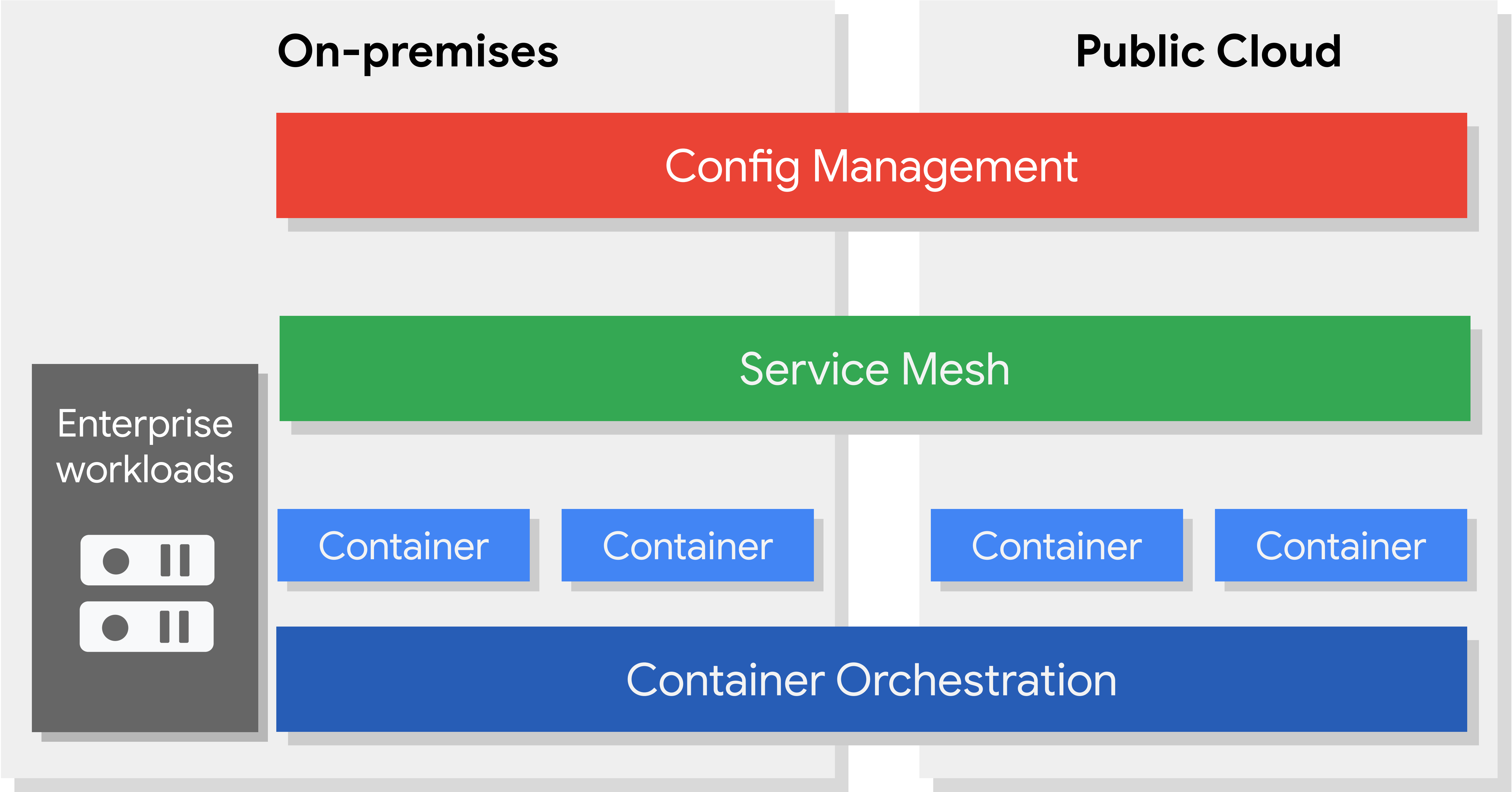
- The hybrid environment empowers companies to pick and choose the best of both worlds, and create bespoke infrastructure



Hybrid environment wishlist

- Write once, deploy in any cloud
- Accelerate developer velocity
- Consistency across environments
- Interoperability with legacy workloads
- Increased observability and SLO
- Decoupling across critical components
- Increased workload mobility
- Avoid vendor lock in

Bringing it together



Google Cloud

Container Orchestration

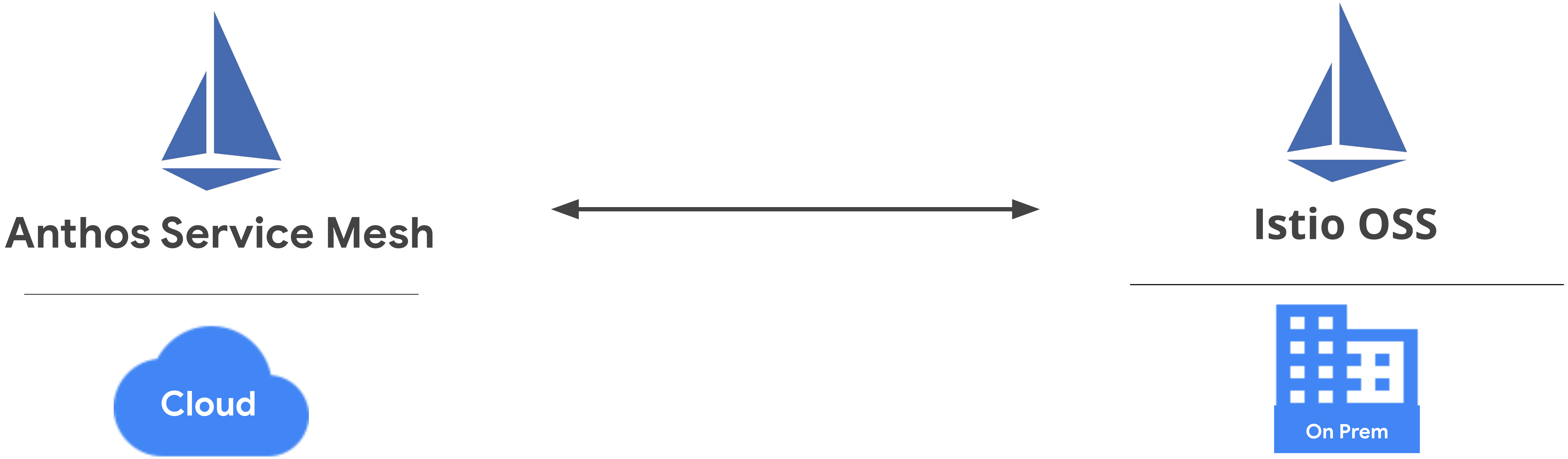
Google Kubernetes Engine

- Operates within GCP
- Managed, production-ready environment for deploying containerized applications
- Operate Seamlessly with High Availability and SLA
- Runs Certified Kubernetes ensuring portability across clouds and on-premises.
- Auto node repair, auto upgrade, auto scaling
- Regional clusters for high availability with multiple masters, node storage replication across 3 zones

Anthos clusters

- Operates On-Premises
- Turn-key, production-grade, conformant Kubernetes with best-practice configuration
- Easy upgrade path to the latest Kubernetes releases that have been validated and tested by Google
- Access to Container services on GCP such as Cloud Build, Container Registry, Audit Logging, and more.
- Integration with Istio, Knative, Marketplace Solutions
- Consistent Kubernetes versions and experience across environments

Service Mesh



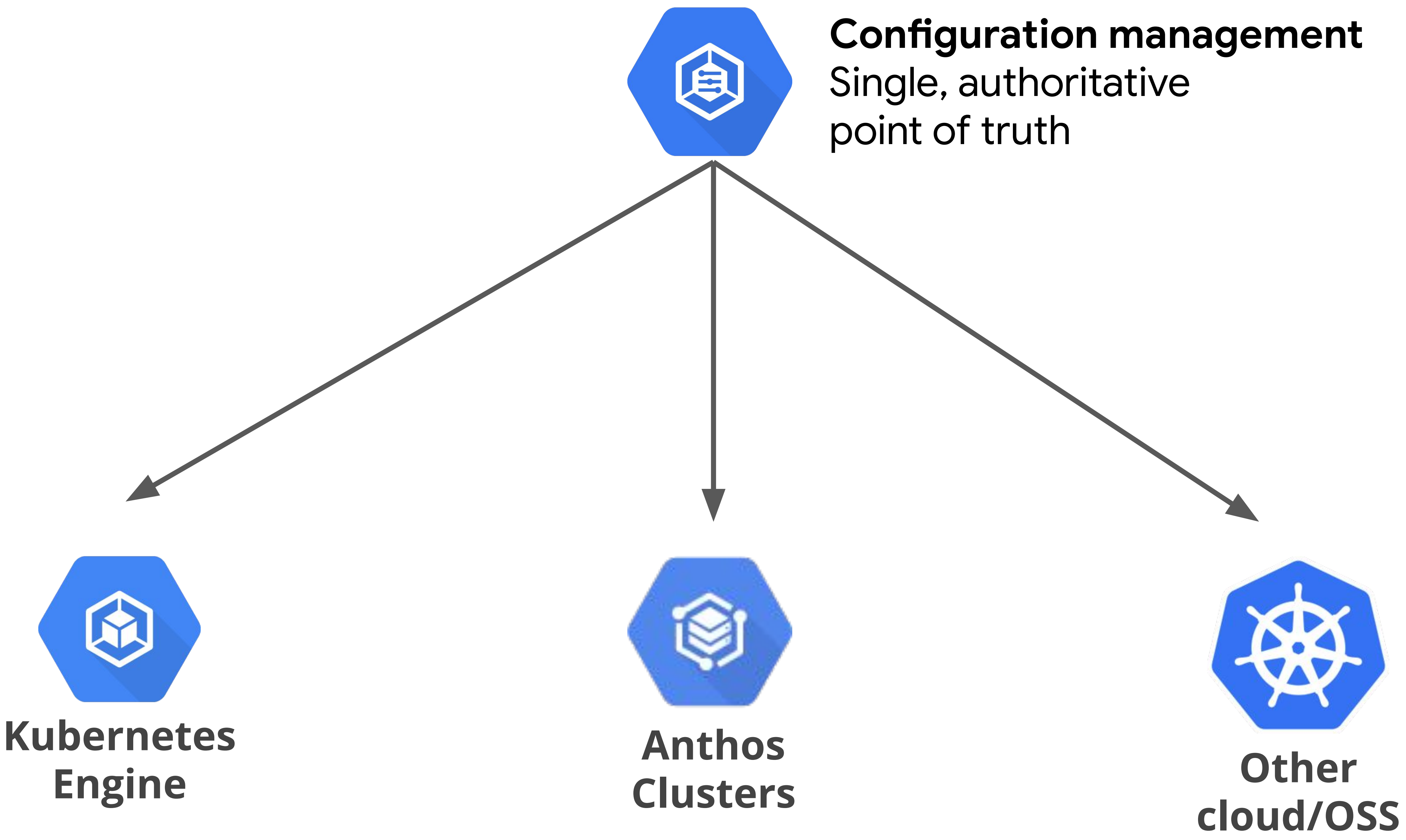
Anthos Service Mesh

- Based on Istio
- Automated/managed installation
- Automatic upgrades
- Managed control plane

Istio OSS

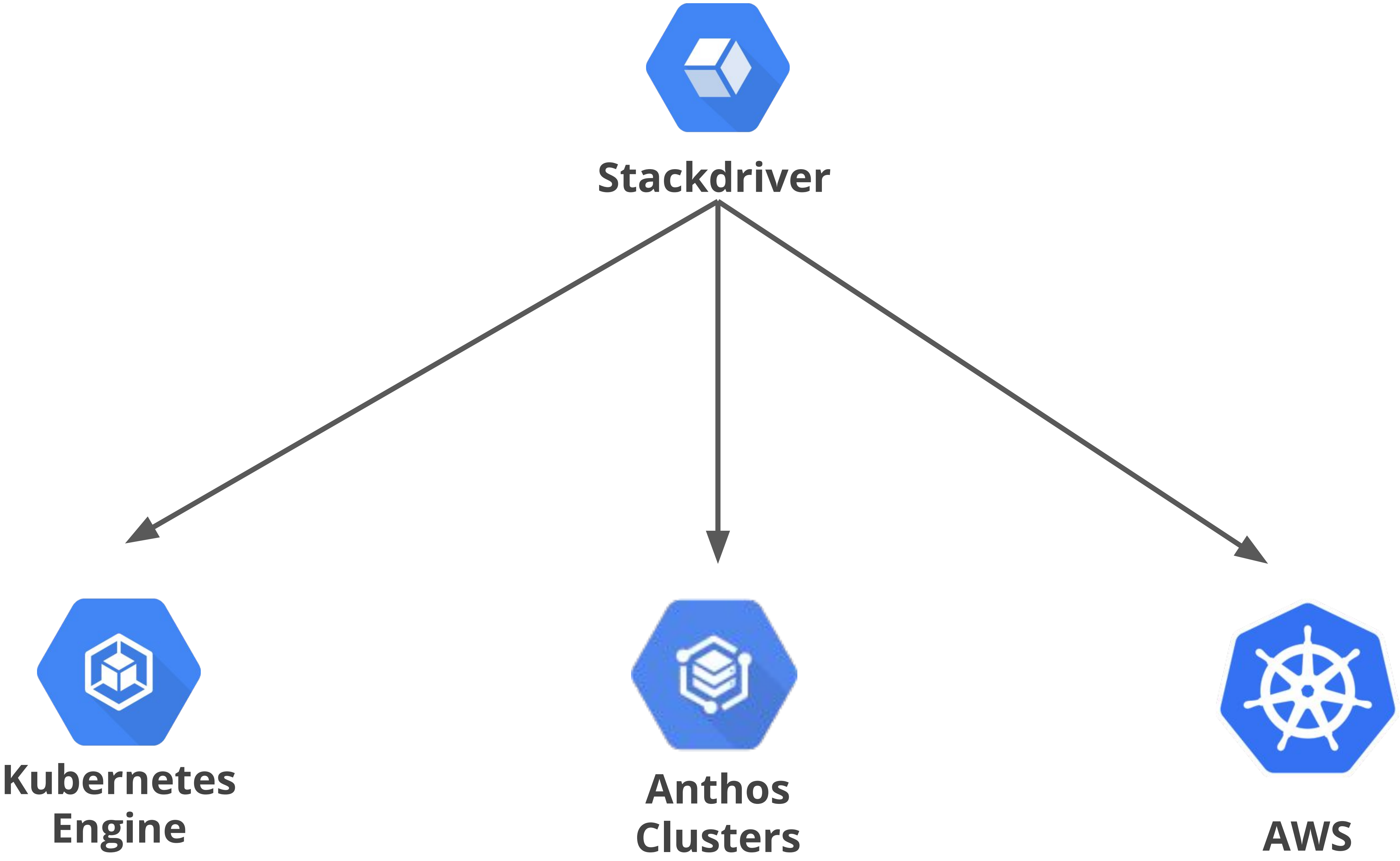
- Self managed open source version
- Fully integrates with Anthos Service Mesh
- Creates a seamless interoperability between the services hosted in different environments

Multicluster Management



Configuration management
Single, authoritative
point of truth

Observability



Stackdriver

Kubernetes Engine

Anthos Clusters

AWS

Anthos

Anthos is a modern application management platform that provides a unified model for computing, networking, and even service management across clouds and data centers.

Kubernetes Engine

Anthos GKE

Anthos Config Management

Anthos Service Mesh

Cloud Logging & Monitoring

Ingress for Anthos

Migrate for Anthos

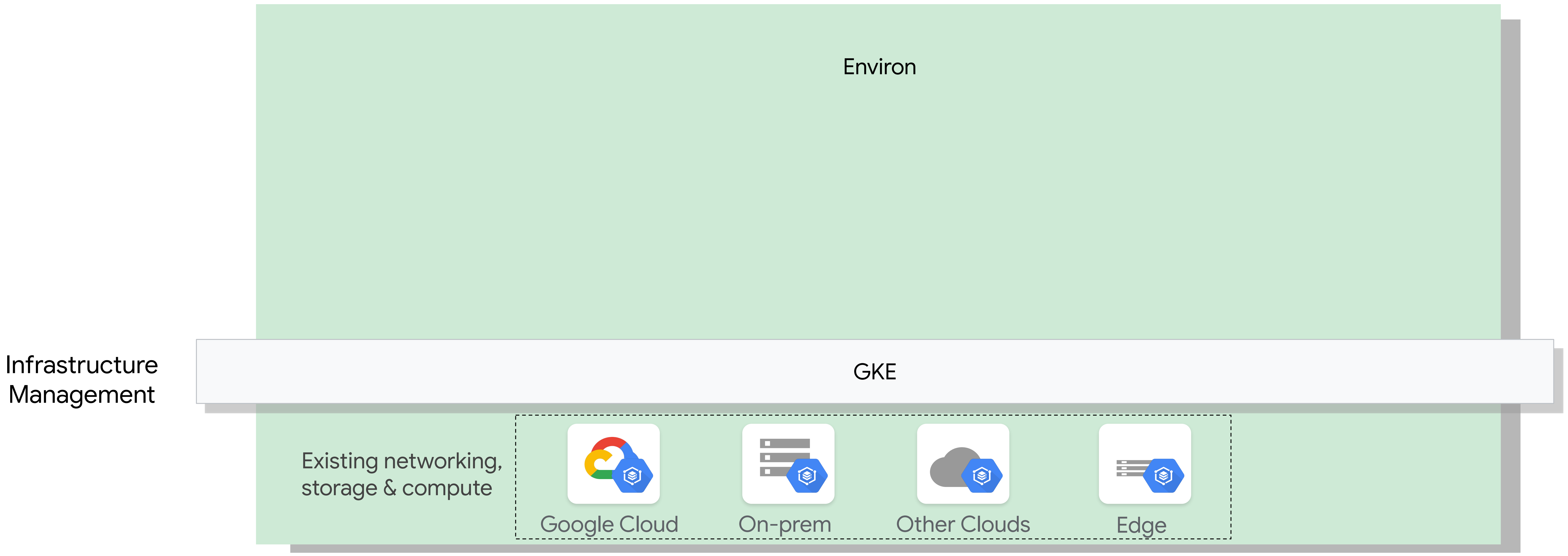
Cloud Run

Cloud Build

GCP Marketplace for Anthos

The technology stack is built on consistent set of APIs based on open-source technologies which empowers developers and operators with a single methodology that applies to on premise, GCP and other cloud providers

Anthos technical overview



Environ

Infrastructure Management

GKE

Existing networking, storage & compute

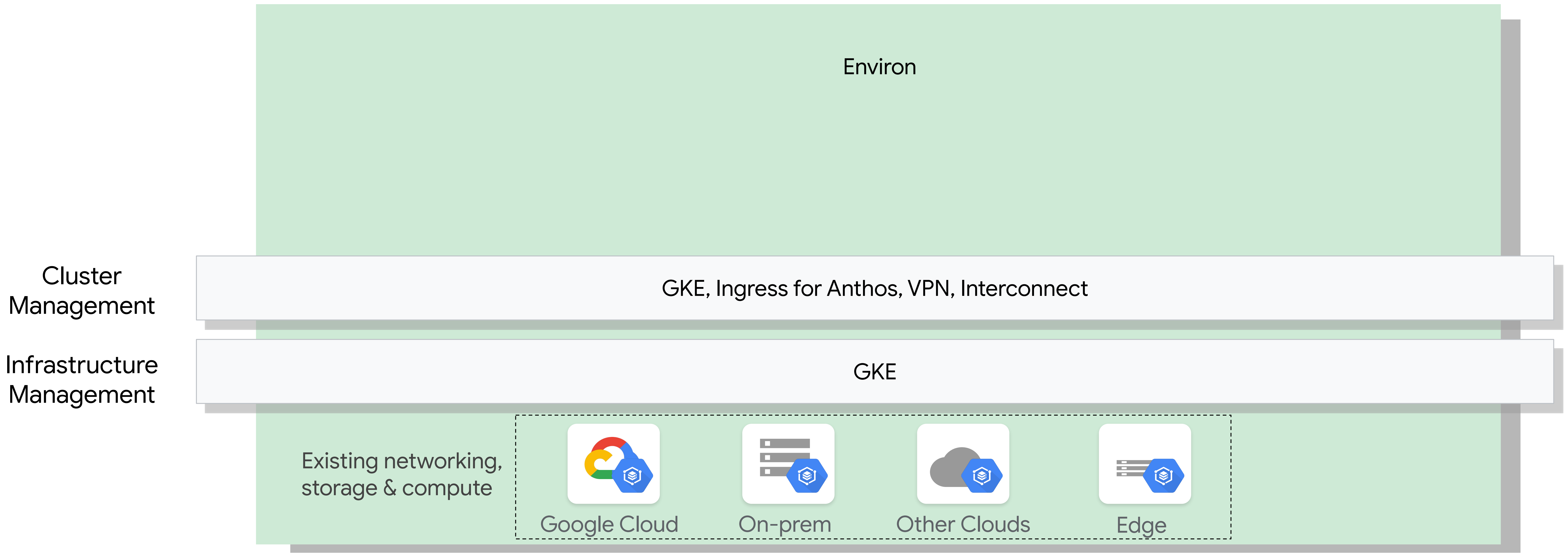
Google Cloud

On-prem

Other Clouds

Edge

Anthos technical overview



Environ

Cluster Management

GKE, Ingress for Anthos, VPN, Interconnect

Infrastructure Management

GKE

Existing networking, storage & compute

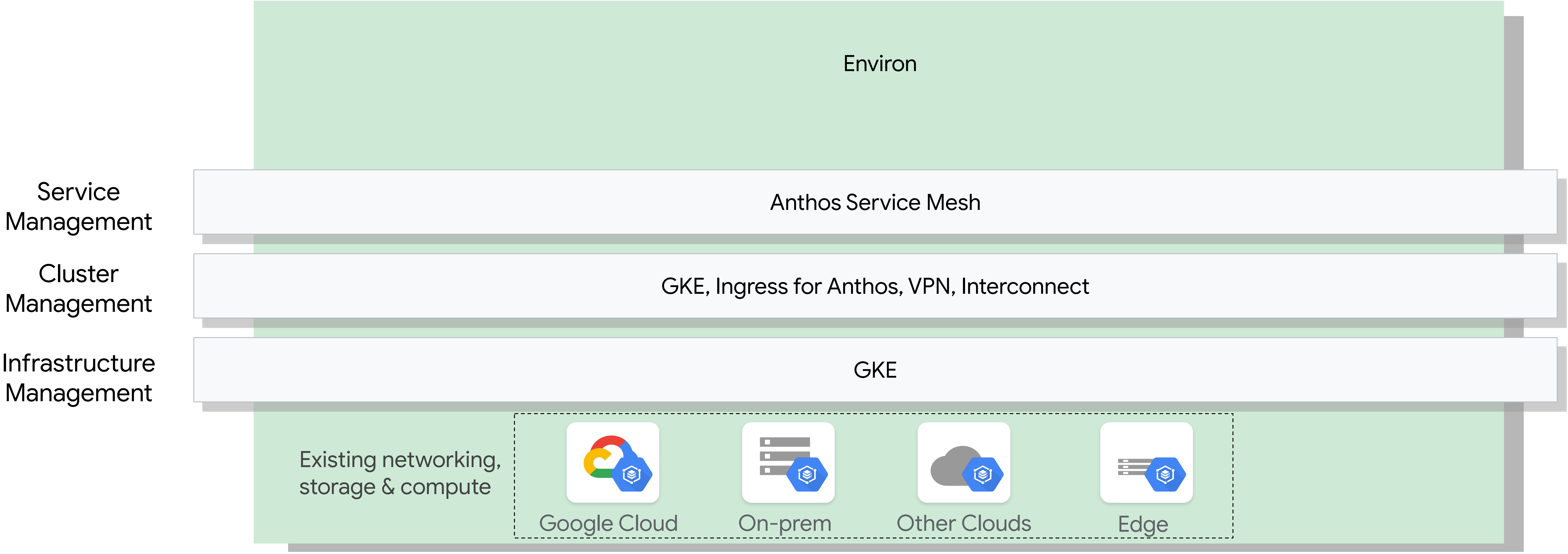
Google Cloud

On-prem

Other Clouds

Edge

Anthos technical overview



Environ

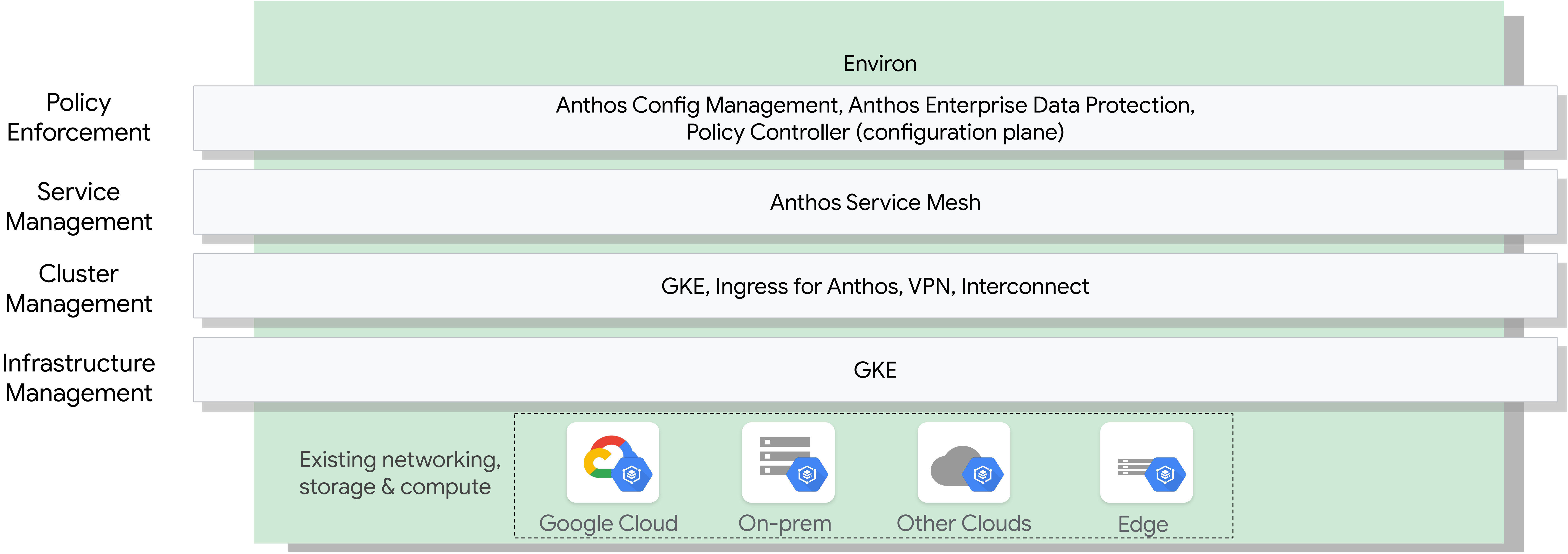
Service Management — Anthos Service Mesh

Cluster Management — GKE, Ingress for Anthos, VPN, Interconnect

Infrastructure Management — GKE

Existing networking, storage & compute

Anthos technical overview



Environ

Policy Enforcement: Anthos Config Management, Anthos Enterprise Data Protection, Policy Controller (configuration plane)

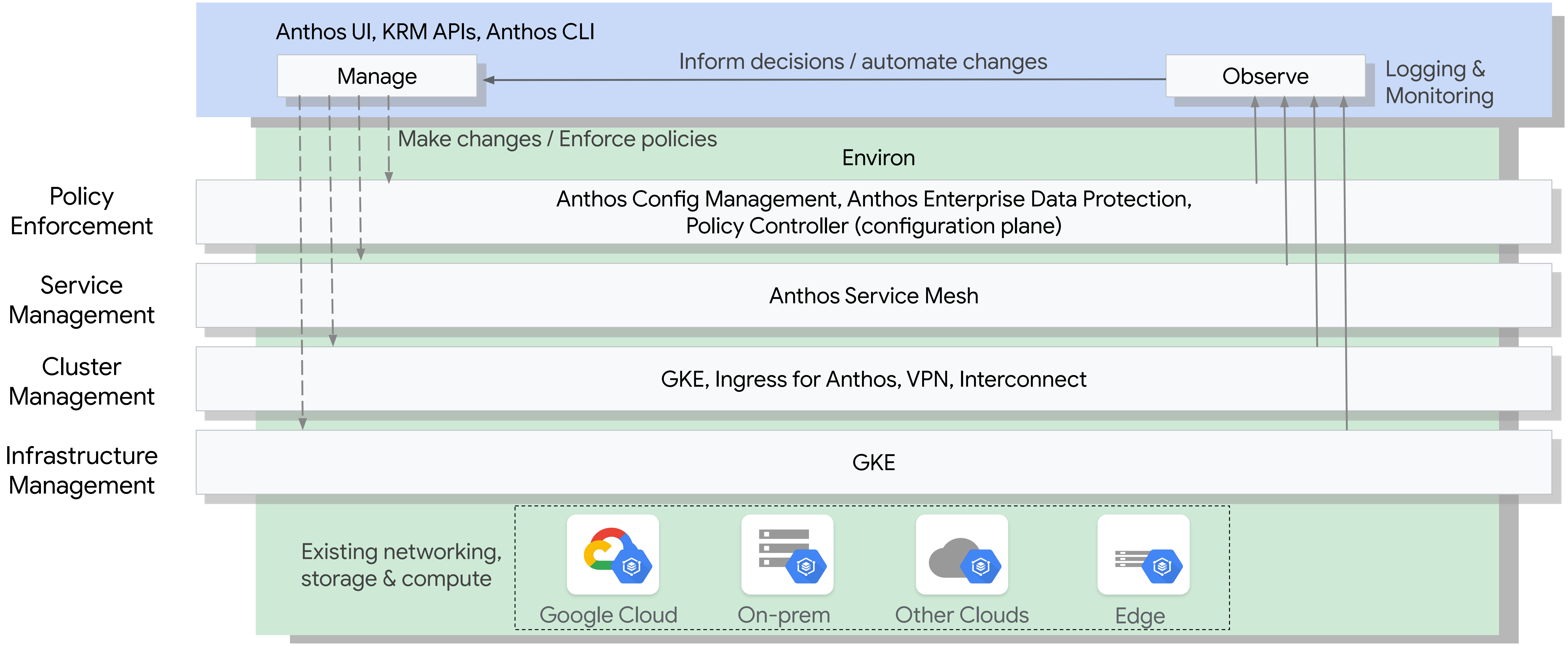
Service Management: Anthos Service Mesh

Cluster Management: GKE, Ingress for Anthos, VPN, Interconnect

Infrastructure Management: GKE

Existing networking, storage & compute

Anthos technical overview



Anthos UI, KRM APIs, Anthos CLI

Manage

Inform decisions / automate changes

Observe

Logging & Monitoring

Make changes / Enforce policies

Environ

Policy Enforcement

Anthos Config Management, Anthos Enterprise Data Protection, Policy Controller (configuration plane)

Service Management

Anthos Service Mesh

Cluster Management

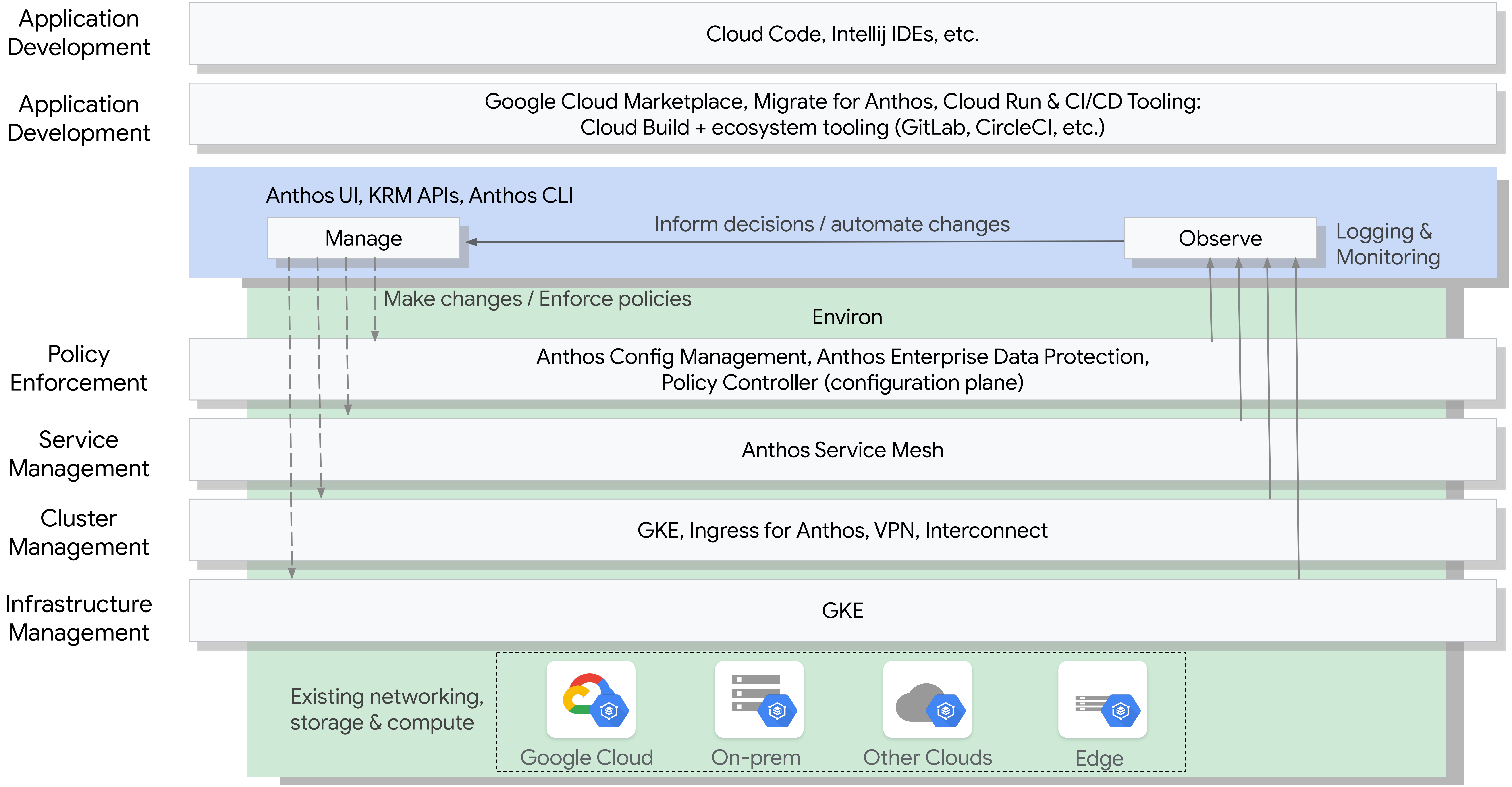
GKE, Ingress for Anthos, VPN, Interconnect

Infrastructure Management

GKE

Existing networking, storage & compute

Anthos technical overview



Application Development — Cloud Code, Intellij IDEs, etc.

Application Development — Google Cloud Marketplace, Migrate for Anthos, Cloud Run & CI/CD Tooling: Cloud Build + ecosystem tooling (GitLab, CircleCI, etc.)

Anthos UI, KRM APIs, Anthos CLI

Manage

Inform decisions / automate changes

Observe

Logging & Monitoring

Make changes / Enforce policies

Environ

Policy Enforcement — Anthos Config Management, Anthos Enterprise Data Protection, Policy Controller (configuration plane)

Service Management — Anthos Service Mesh

Cluster Management — GKE, Ingress for Anthos, VPN, Interconnect

Infrastructure Management — GKE

Existing networking, storage & compute

Google Cloud

On-prem

Other Clouds

Edge

Anthos Benefits

- A consistent platform for all your application deployments, both legacy as well as cloud native, while offering a service-centric view of all your environments.

- Build enterprise-grade containerized applications faster with managed Kubernetes on cloud and on-premises environments. Create a fast, scalable software delivery pipeline with cloud-native tooling and guidance.

- Leverage a programmatic, outcome-focused approach to managing policies for apps across environments, and enable greater awareness and control with a unified view of your services' health and performance.

Google Cloud