Kubernetes Study Jam

Ramón Medrano Llamas - Staff Site Reliability Engineer

Google Cloud

Agenda

1. Kubernetes **Core Concepts**
2. Google **Kubernetes Engine**
3. Istio **Service Mesh**

Google Cloud

Kubernetes
Core Concepts

Google Cloud

01

At Google, everything runs in a container

On average, we launch

4 billion

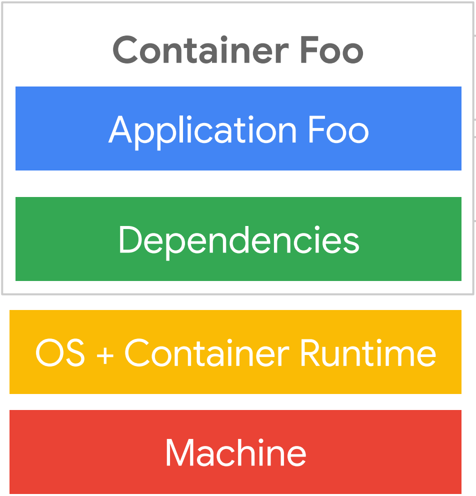
new containers per week

(That’s 571M/day, 24M/hour, or ~6600/sec)

Google Cloud

Container?

Processes isolated by OS primitives



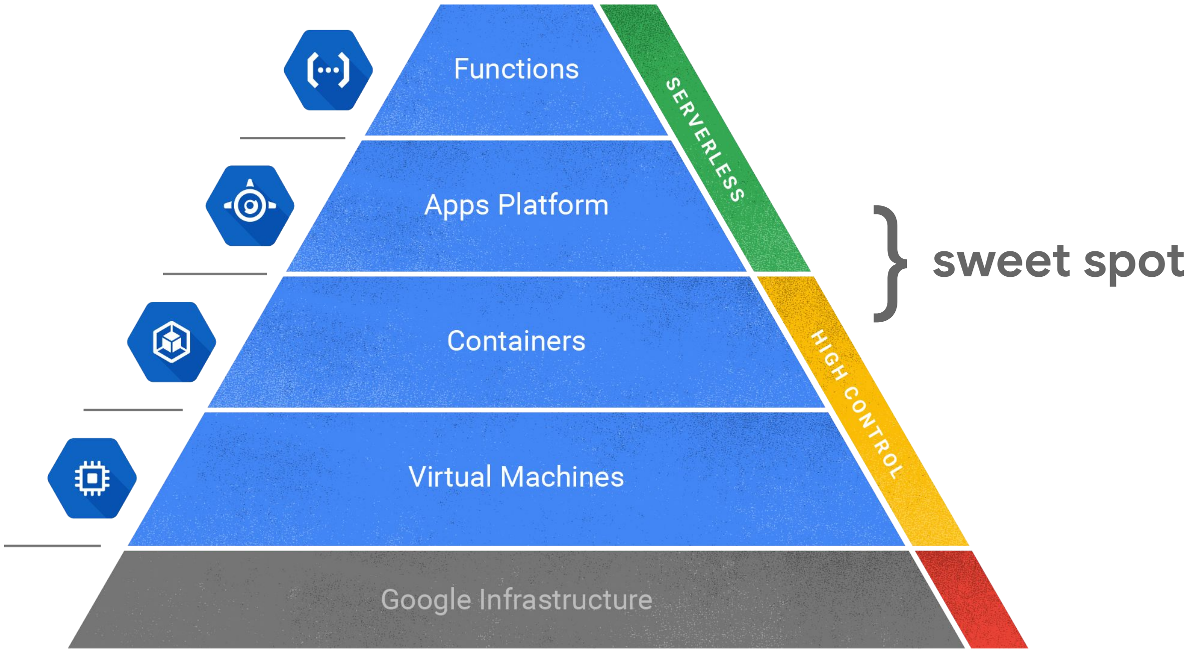
Namespaces isolate **what processes can see**

cgroups isolate **what resources can be used***

Portable across any system with
Container Runtime (local, dev, prod)

*developed by Google in 2006

The compute spectrum



Google Cloud

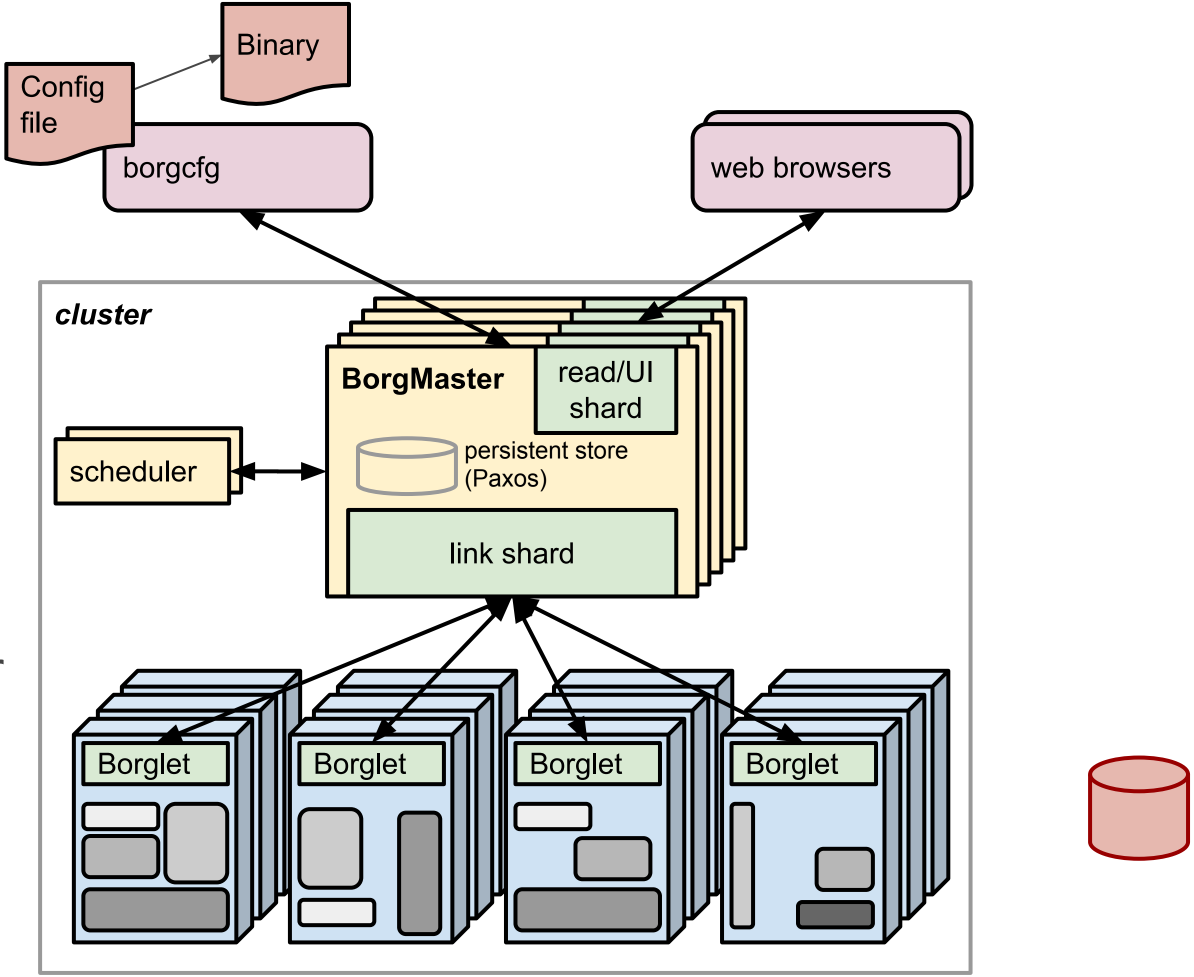
Borg

No VMs, pure containers

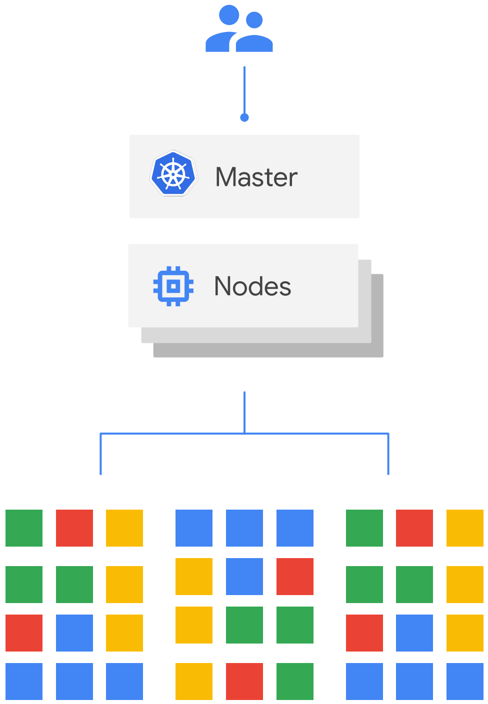
10K - 20K nodes per cluster

DC-scale job scheduling

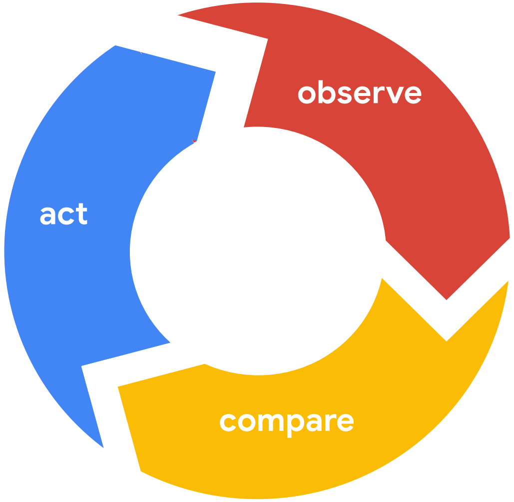
CPU, mem, disk and IO



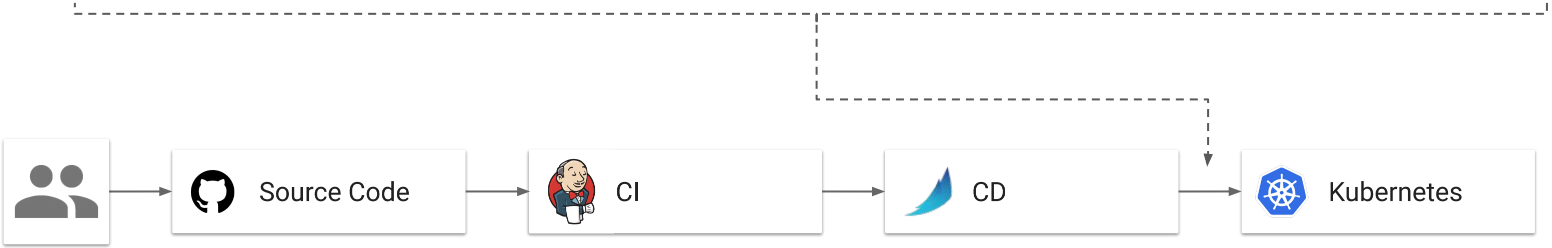
Kubernetes abstracts away infrastructure



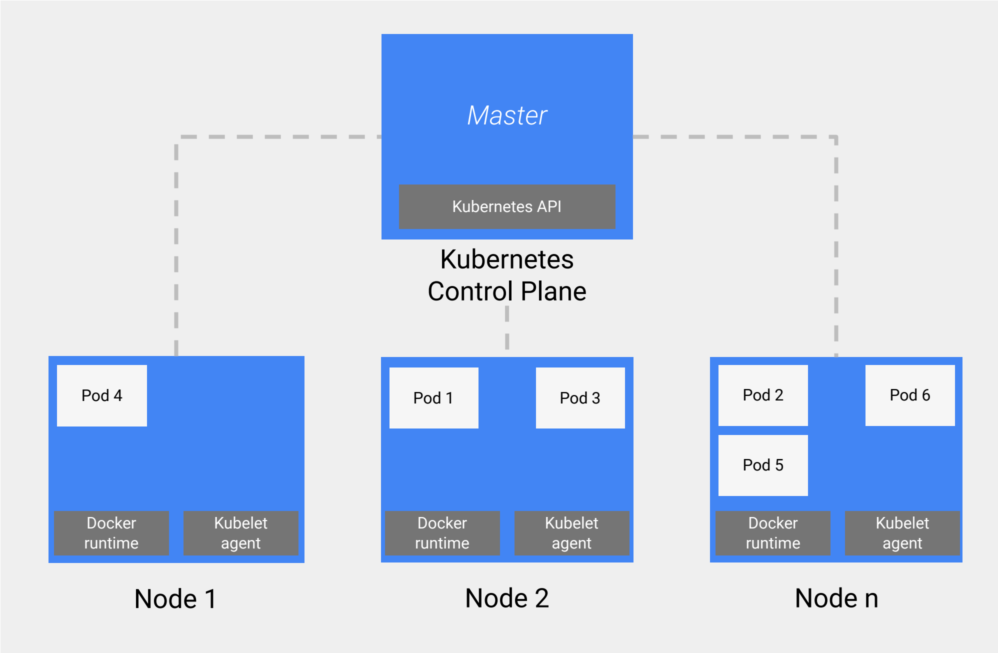
Kubernetes provides a declarative API



```
$ kubectl apply -f k8s-manifest.yaml
```



Kubernetes Architecture

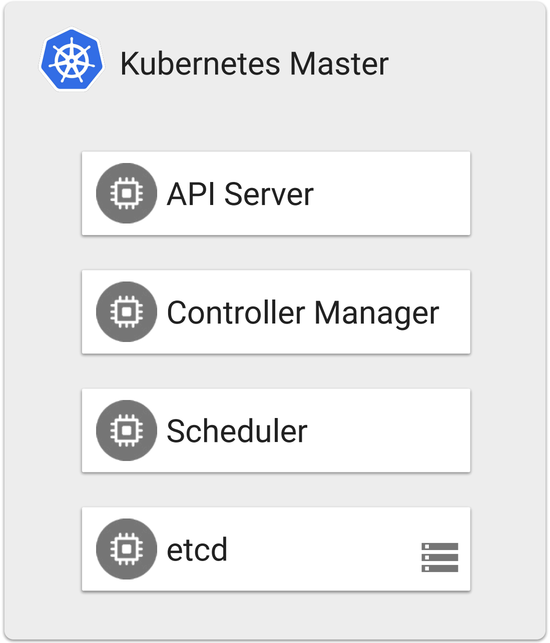


Control Plane

The Kubernetes Master also known as the Control Plane

Its job is to **know the current state of the cluster** and make decisions to **move the cluster to its desired state**.

This can be a single node but is horizontally scalable for High Availability.



Control Plane: kube-apiserver

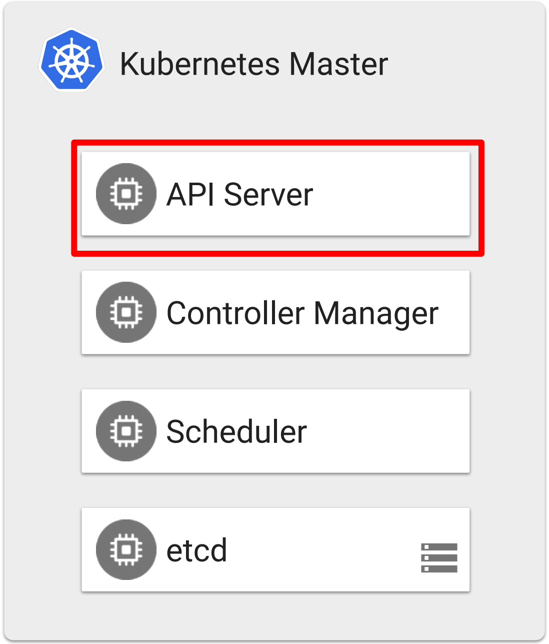
AKA The API Server

Stateless REST server that **exposes Kubernetes API,** backed by a datastore

All communication about cluster state flows through the API Server.

Validates Kubernetes objects and interacts with end users, scheduler, controller managers, and kubelets

Supports CRUD and Watch operations



Google Cloud

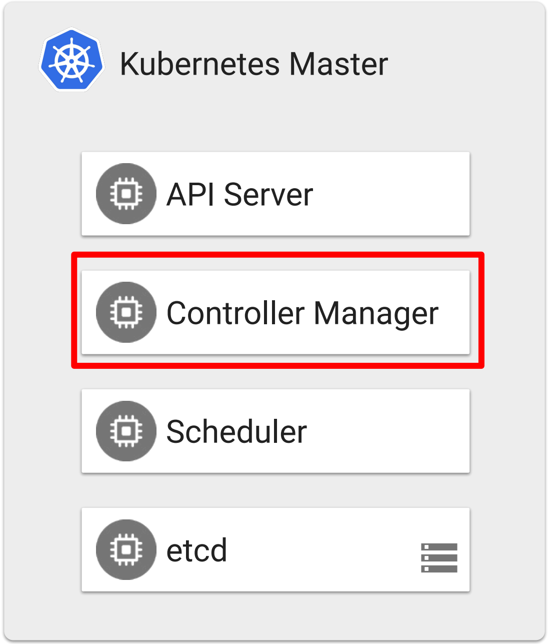
Control Plane: controller-managers

AKA **managing controllers powering Kubernetes abstractions**

20+ control loops that help abstractions like deployments work

+ cloud-controller-manager that helps Kubernetes integrate with cloud providers for persistent disk, load balancers, else

Clean separation of each controller's functionality



Google Cloud

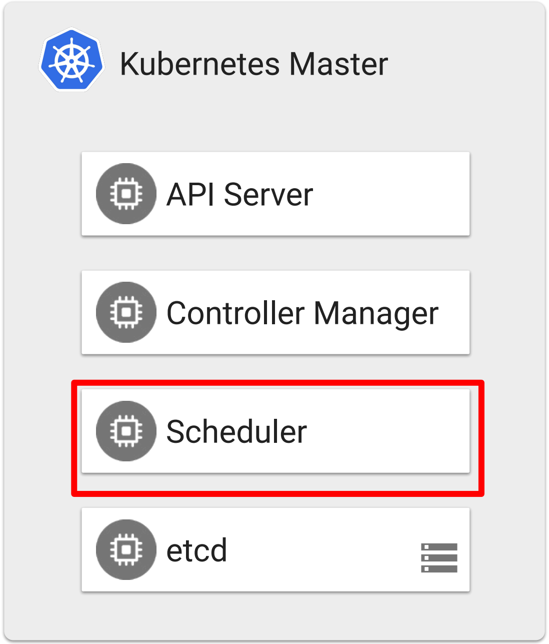
Control Plane: kube-scheduler

AKA The Scheduler

A control loop that is crucial to cluster operation by **ensuring that nodes run pods**

If the API Server stores current and desired state of the cluster, the **scheduler uses that data to make decisions about where and when pods should run**

Makes scheduling decisions based on multiple data points



Google Cloud

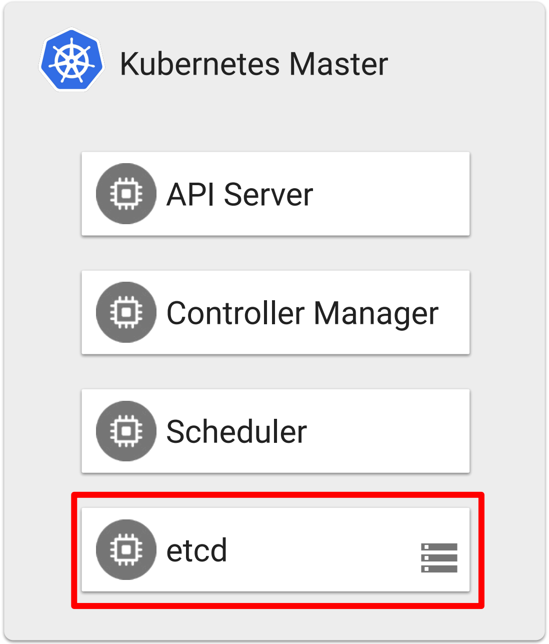
Control Plane: etcd

AKA The API Server's datastore

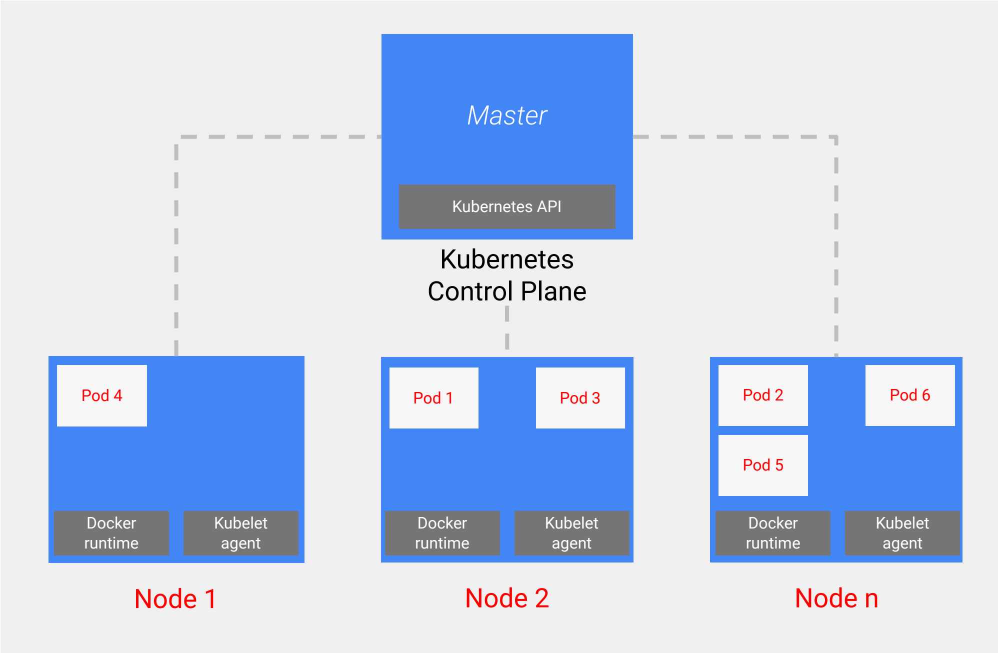
The backing service to the API Server; it's an implementation detail

Distributed, strongly consistent, and highly available kv store, **powered by Raft consensus** - this means in High Availability (HA) we must run > 2 master nodes

Persists all cluster data



Kubernetes Architecture (Revisit)



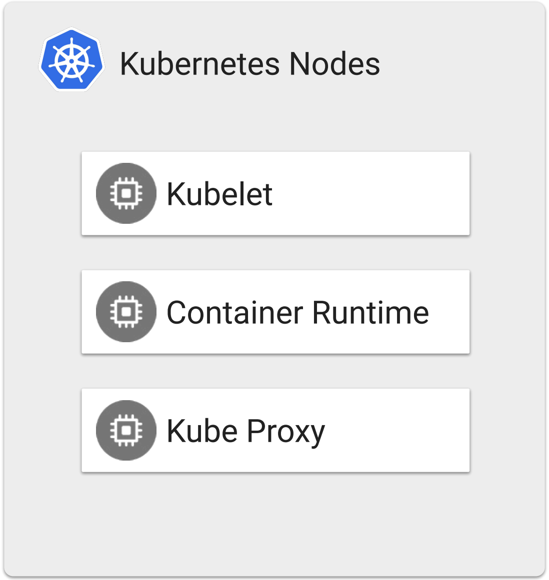
Google Cloud

Cluster Nodes

The underlying machines (physical or virtual) are known as the nodes

Nodes communicate with the API server, execute container processes, and route container traffic

These can be scaled out to many instances and sized to various configurations. Node Pools share the same VM configurations



The Node: kubelet

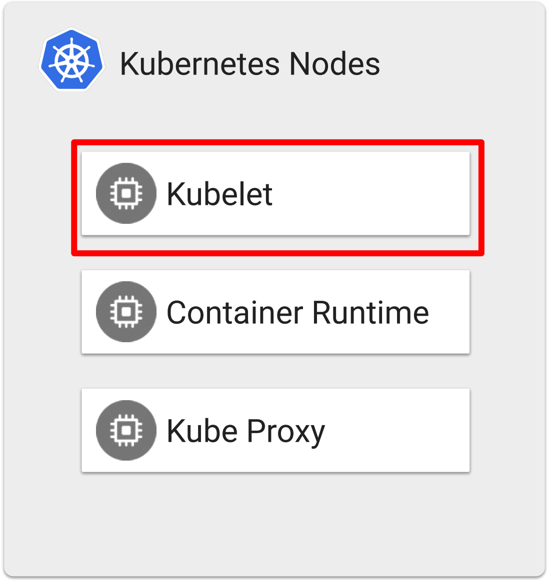
AKA the node agent

Communicates with API Server to know what pods it should run

Will kick execution of a set of containers to the Container Runtime

Will fetch secrets, environment variables from the API Server for Containers

Broadcasts status of pods, nodes



Google Cloud

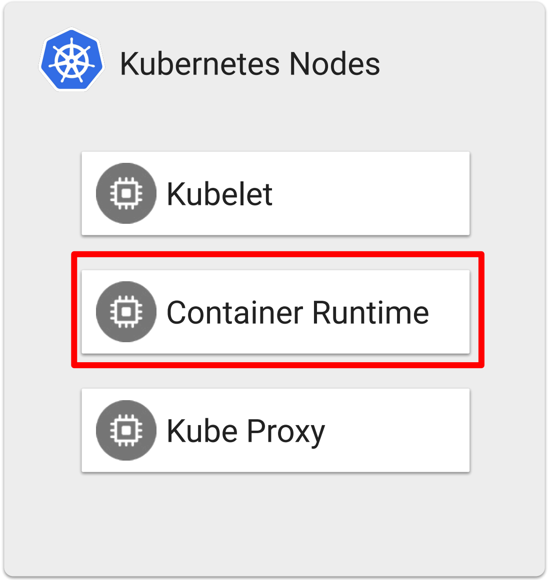
The Node: Container Runtime Interface

Default is Docker

Kubernetes also supports rkt

The Container Runtime is actually responsible for executing your processes

Looking to support all open container initiative compliant runtimes via CRI-O

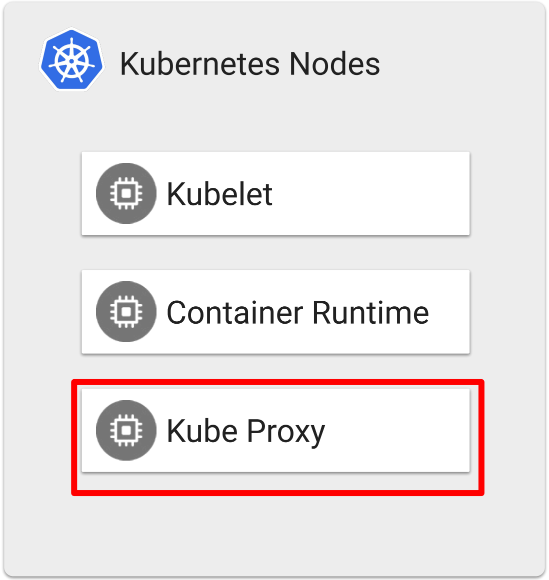


Google Cloud

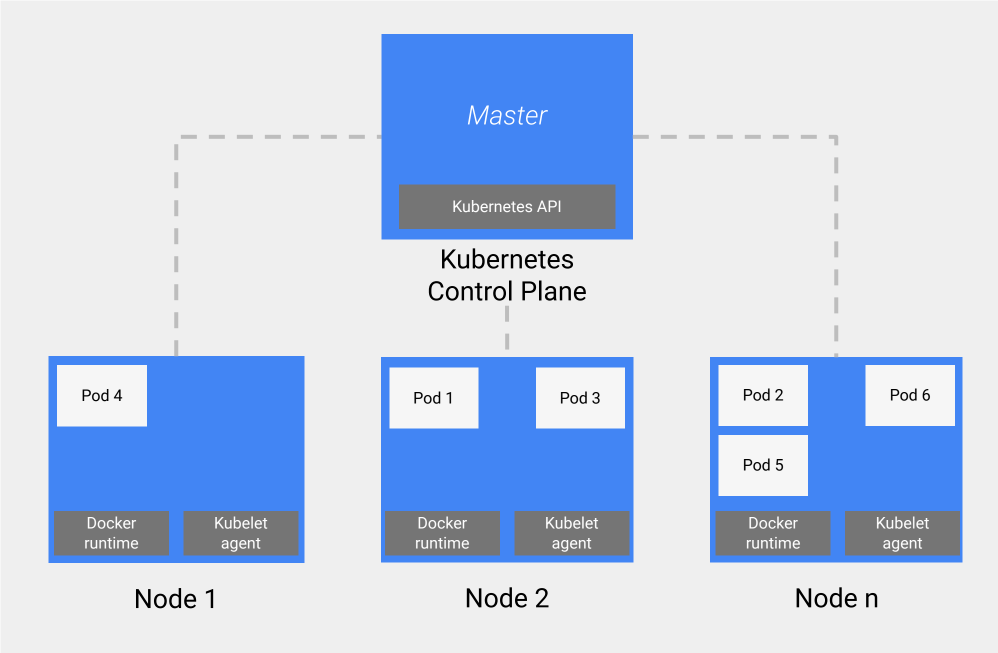
The Node: kube-proxy

Watches Pods and Services in the cluster and makes the Service IP forward traffic to the set of Pod IPs

Runs on every node and generates/updates iptables rules



Kubernetes Architecture (Recap)



Google Cloud

Core Concepts

- Namespaces
- Pods
- Deployments
- Services

Google Cloud

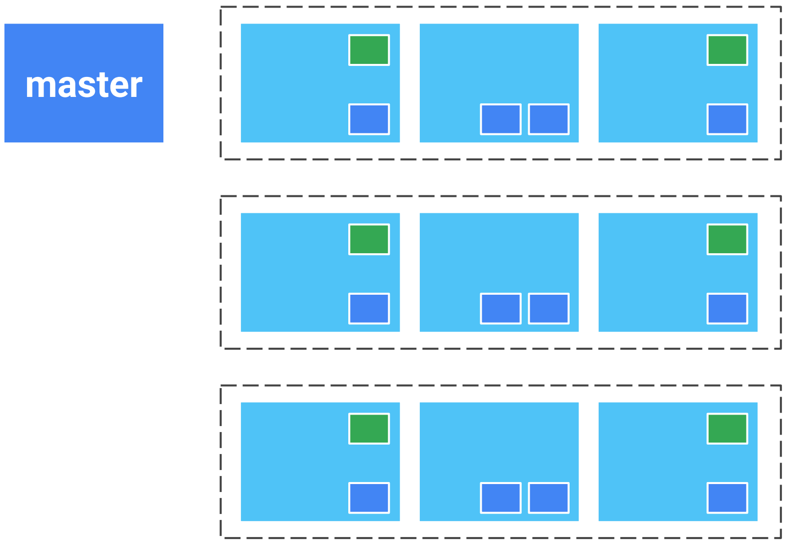
Core Concepts: Namespaces

Namespaces: Logical isolation between kubernetes objects

Most resources are scoped to a namespace, but there are parts of kubernetes outside of namespaces scope (ie nodes)

Can be used for Role Based Access Control (RBAC)

Useful for **isolating environments** within a single cluster to multiple team members



Core Concepts

- Namespaces
- Pods
- Deployments
- Services

Core Concepts: The Pod

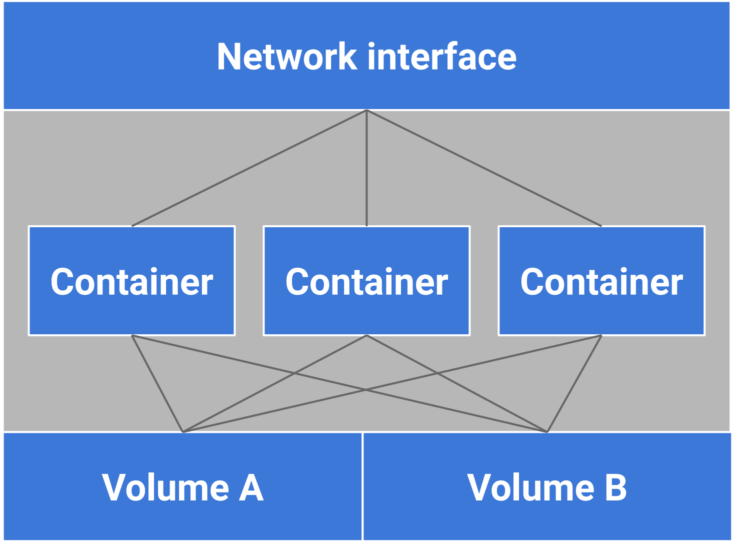
Pod: The atomic unit of Kubernetes

Comprised of one or few containers with shared networking & storage

Containers in a pod share most linux namespaces, but not control groups

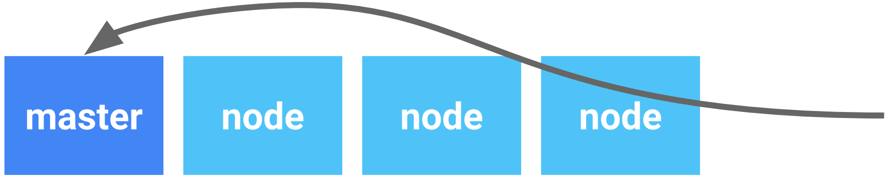
Kubernetes will nicely automate setting up namespace, cgroup

Great for packaging containers together



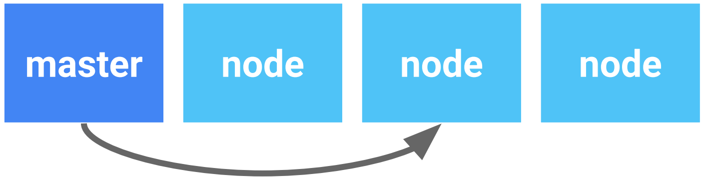
Google Cloud

Core Concepts: The Pod (and manifest)



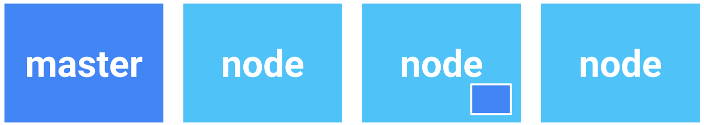

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-app
    image: my-app
  - name: nginx-ssl
    image: nginx
    ports:
    - containerPort: 80
    - containerPort: 443
```

Core Concepts: The Pod (and manifest)




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name:  my-app
    image: my-app
  - name:  nginx-ssl
    image: nginx
    ports:
    - containerPort: 80
    - containerPort: 443
```

Core Concepts: The Pod (and manifest)




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name:  my-app
    image: my-app
  - name:  nginx-ssl
    image: nginx
    ports:
    - containerPort: 80
    - containerPort: 443
```

Core Concepts

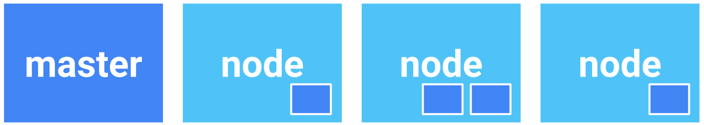
- Namespaces
- Pods
- Deployments
- Services

Core Concepts: Deployments

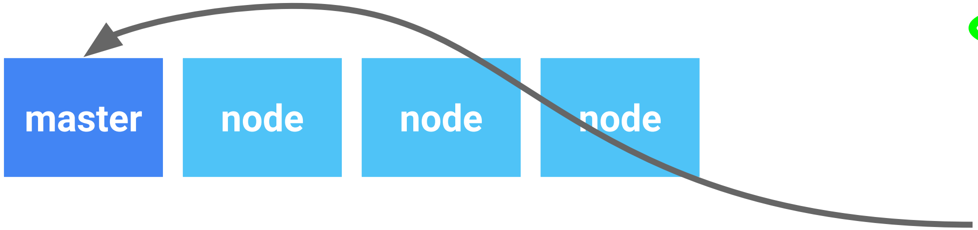
Deployment: An abstraction that allows you to define and update desired pod template and replicas

If pods are mortal, abstractions like deployments give us resiliency

One of many abstractions to control how pods are scheduled and deployed

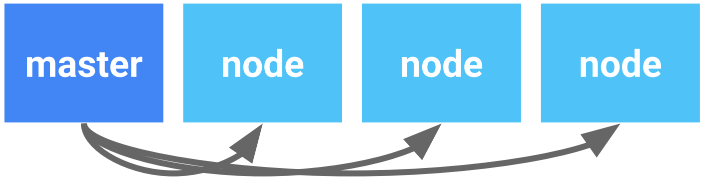


Core Concepts: Deployments




```yaml
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
        - name:  my-app
          image: my-app
        - name:  nginx-ssl
          image: nginx
          ports:
          - containerPort: 80
          - containerPort: 443
```

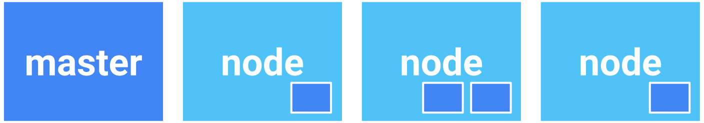
Core Concepts: Deployments




```
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
       - name:  my-app
         image: my-app
       - name:  nginx-ssl
         image: nginx
         ports:
          - containerPort: 80
          - containerPort: 443
```

Google Cloud

Core Concepts: Deployments




```yaml
kind: Deployment
apiVersion: v1beta1
metadata:
  name: frontend
spec:
  replicas: 4
  selector:
    role: web
  template:
    metadata:
      name: web
      labels:
        role: web
    spec:
      containers:
        - name:  my-app
          image: my-app
        - name:  nginx-ssl
          image: nginx
          ports:
          - containerPort: 80
          - containerPort: 443
```

Core Concepts

- Namespaces
- Pods
- Deployments
- Services

Google Cloud

Core Concepts: Services

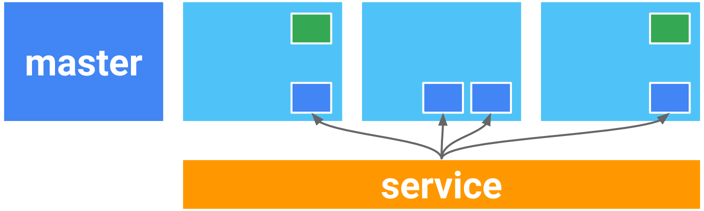
Services: Stable endpoint for pods

If pod IPs are mortal, services give us a stable way to access our pods

Provides load balancing across multiple pods

With services you can speak to pods via external IP, cluster internal IP or DNS

Service will target multiple pods with the same key/value pair metadata, known as a label selector

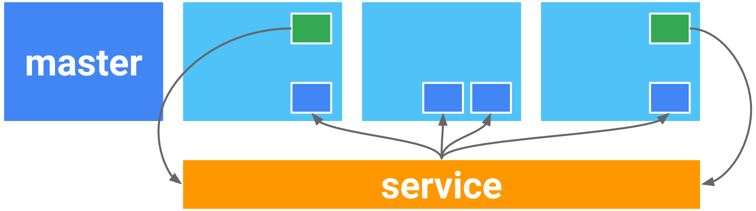


Google Cloud

Core Concepts: Services

Internal Calls

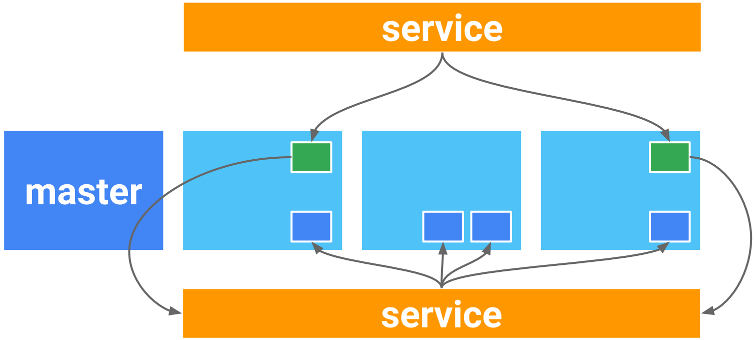
- Service Type: **ClusterIP**
    - Internal IP, available only within the cluster



Core Concepts: Services

External Calls

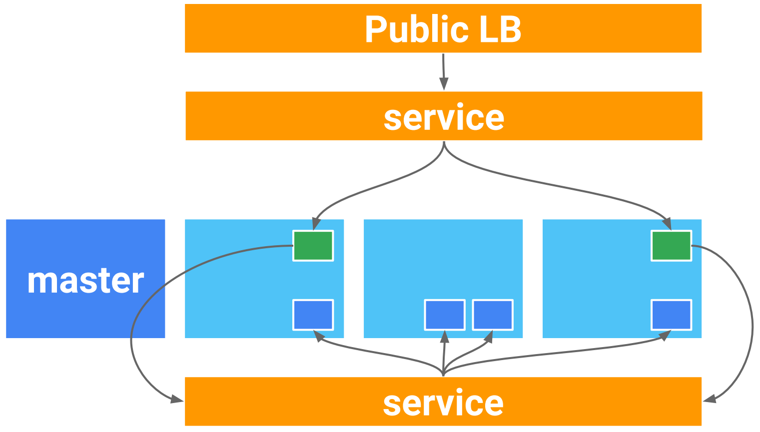
- Service Type: **NodePort**
  - externalizes service by making it available at each node's IP & specified port, routing that to ClusterIP



Core Concepts: Services

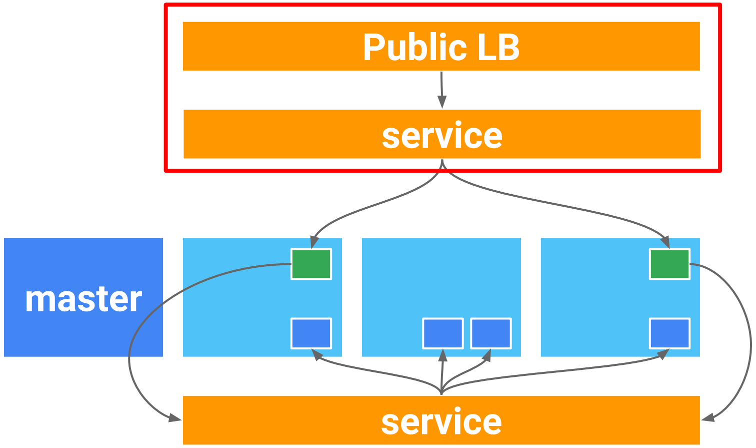
Public Load Balancers

- Service Type: **LoadBalancer**
    - Create a load balancer with the cloud provider in front of NodePort/ClusterIP



Google Cloud

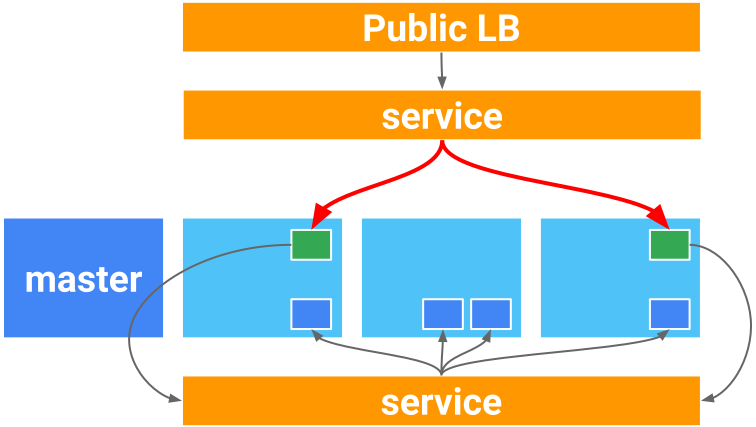
Core Concepts: Services




```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    role: web
  type: LoadBalancer
```

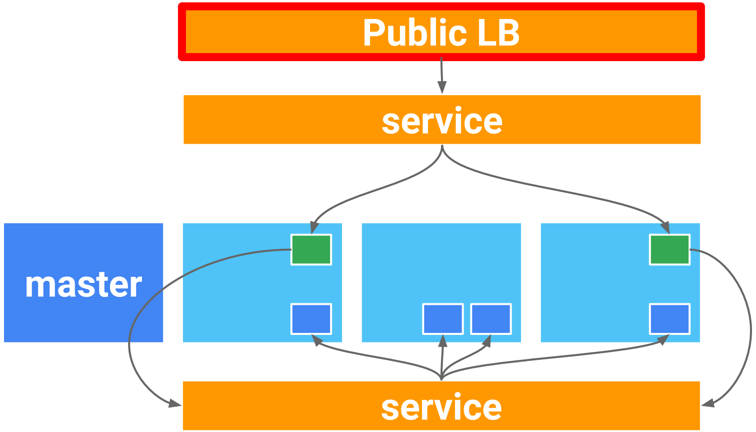
Google Cloud

Core Concepts: Services




```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    role: web
  type: LoadBalancer
```

Core Concepts: Services




```yaml
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    role: web
  type: LoadBalancer
```

Google Cloud

Kubernetes Handles...

Scheduling:
Decide where my containers should run

Lifecycle and health:
Keep my containers running despite failures

Scaling:
Make sets of containers bigger or smaller

Naming and discovery:
Find where my containers are now

Load balancing:
Distribute traffic across a set of containers

Storage volumes:
Provide data to containers

Logging and monitoring:
Track what's happening with my containers

Debugging and introspection:
Enter or attach to containers

Identity and authorization:
Control who can do things to my containers

Custom Resource Definitions

Example CRD

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: securedeployments.ctl.gcp.solutions
spec:
  group: ctl.gcp.solutions
  version: v1
  scope: Namespaced
  names:
    plural: securedeployments
    singular: securedeployment
    kind: SecureDeployment
    shortNames: ["sd", "securedeploy"]
```

```
$ kubectl get sd
$ kubectl describe securedeploy
```

CRDs

When?

- You want to create a new kind of object
- You want to package multiple objects as one

What?

- Extension of the Kubernetes API
- You write the spec and build a controller

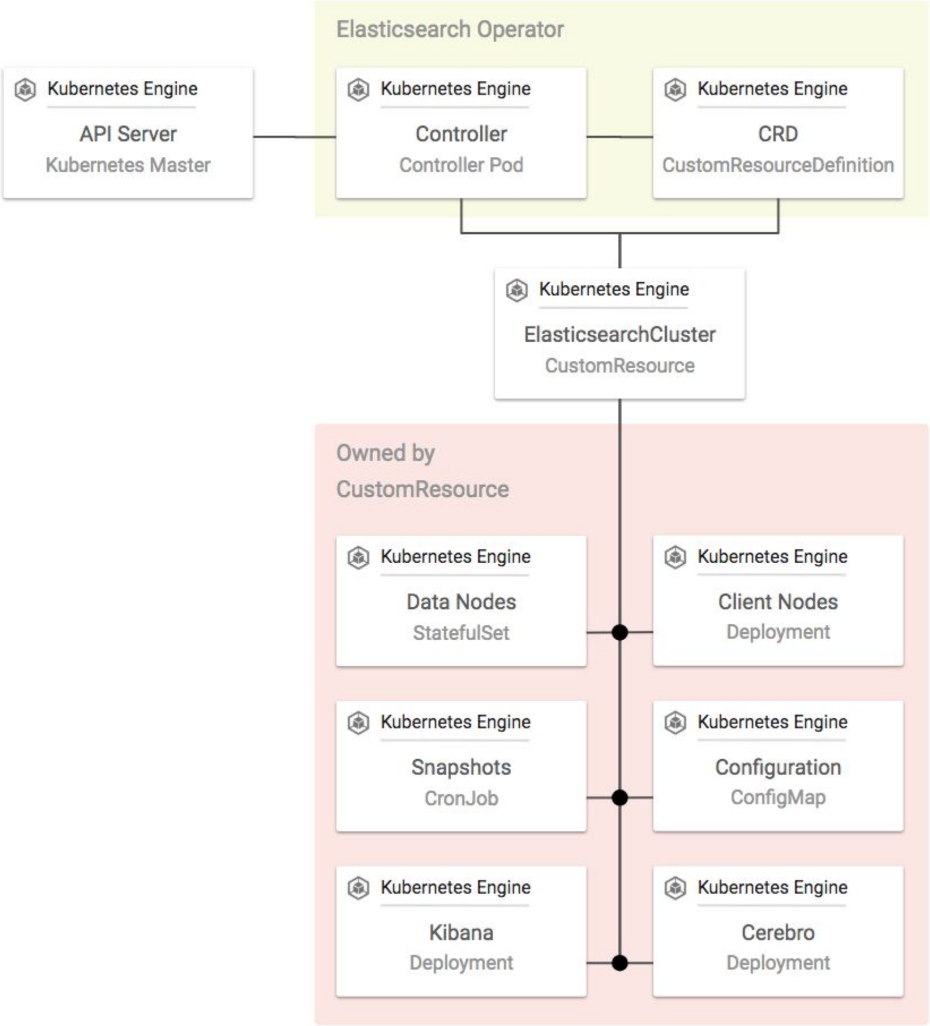
Where?

- Docs: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

Google Cloud

The Operator Pattern

Google Cloud



Google Cloud

Elasticsearch Operator

```
apiVersion: enterprises.upmc.com/v1
kind: ElasticsearchCluster
metadata:
  namespace: elasticsearch
  name: example-es-cluster
spec:
  kibana:
    image: kibana/kibana-oss:6.1.3
  cerebro:
    image: cerebro:0.6.8
  elastic-search-image: elasticsearch-kubernetes:6.1.3_1
  client-node-replicas: 3
  master-node-replicas: 2
  data-node-replicas: 3
  data-volume-size: 100Gi
  snapshot:
    scheduler-enabled: true
    type: gcs
    bucket-name: my-project-snapshots
    cron-schedule: "@every 2m"
    image:
cloud-solutions-group/elasticsearch-cron:0.0.4
```

The kind defined by the CustomResourceDefinition

Operator gives me:

1. Elasticsearch Cluster with configurable topology
2. Kibana
3. Cerebro (dashboard)
4. Snapshot jobs with cron schedule

Google Cloud

Google Kubernetes Engine

02

Google Cloud

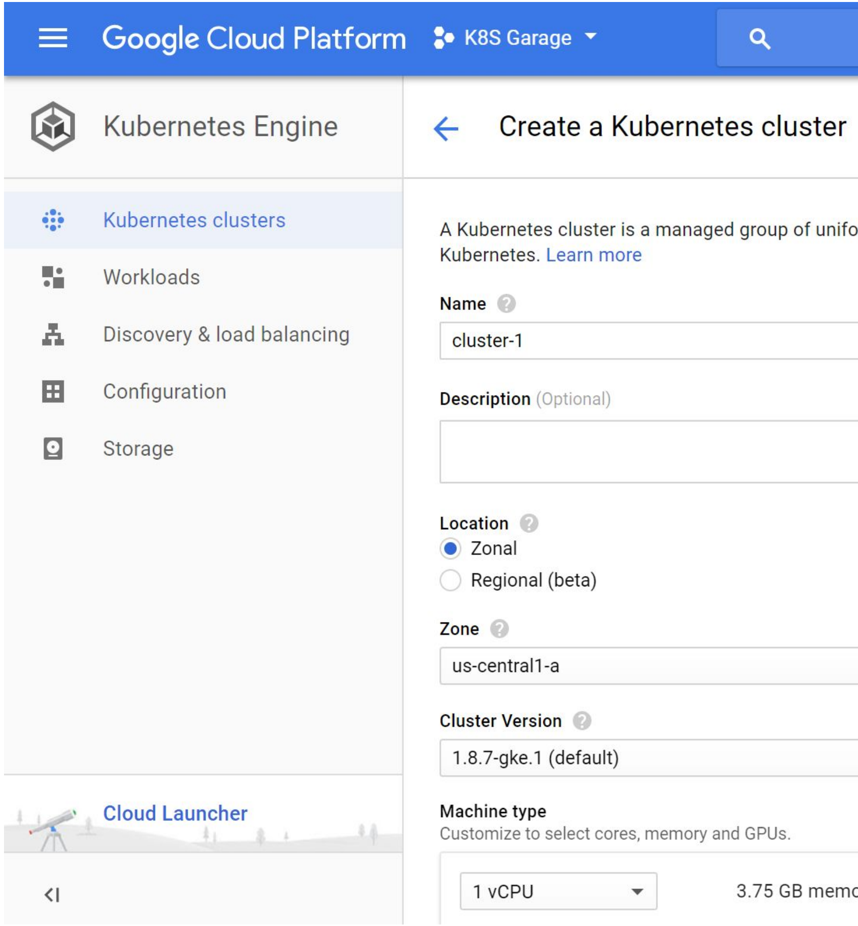
Kubernetes the **Easy Way**

Start a cluster with one-click

View your clusters and workloads in a single pane of glass

Let Google keep your cluster up and running

Google Cloud

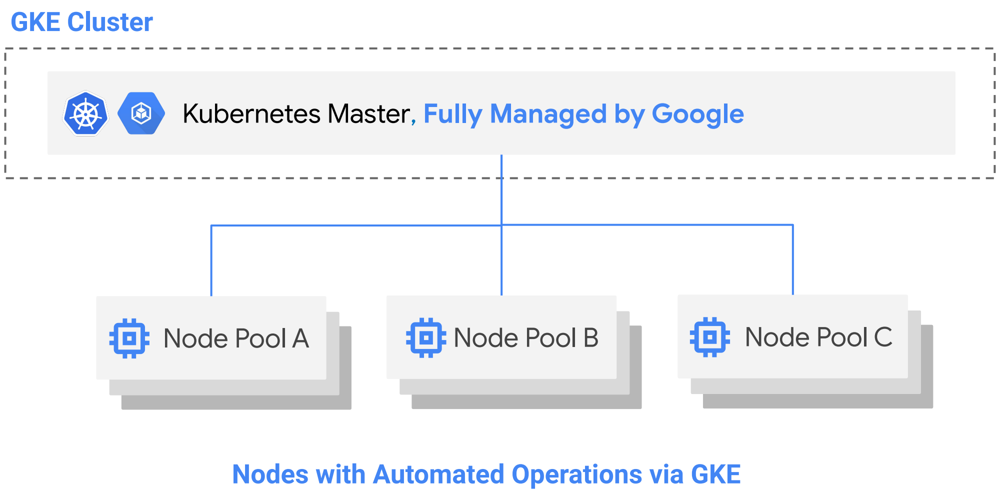


Enter Google Kubernetes Engine

GKE is Google Cloud's Kubernetes Platform

Generally Available since August 2015

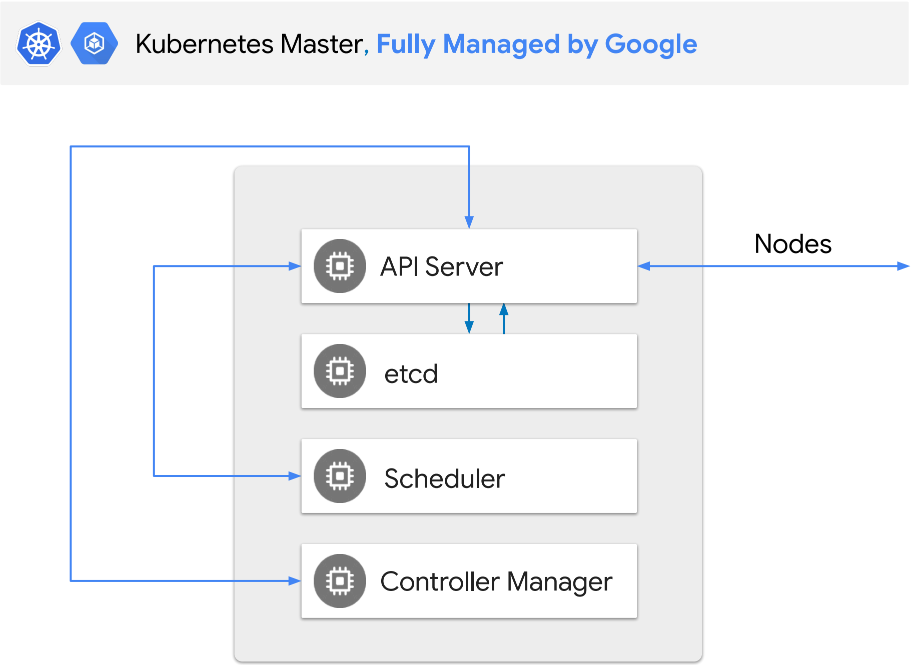
Take advantage of the **deep integration with Google Cloud Platform** features and services



Fully Managed K8s Control Plane

Site Reliability Engineers **manage, scale, and upgrade** the control plane in a **Google-owned project**

Upstream Kubernetes, tracks open source releases closely



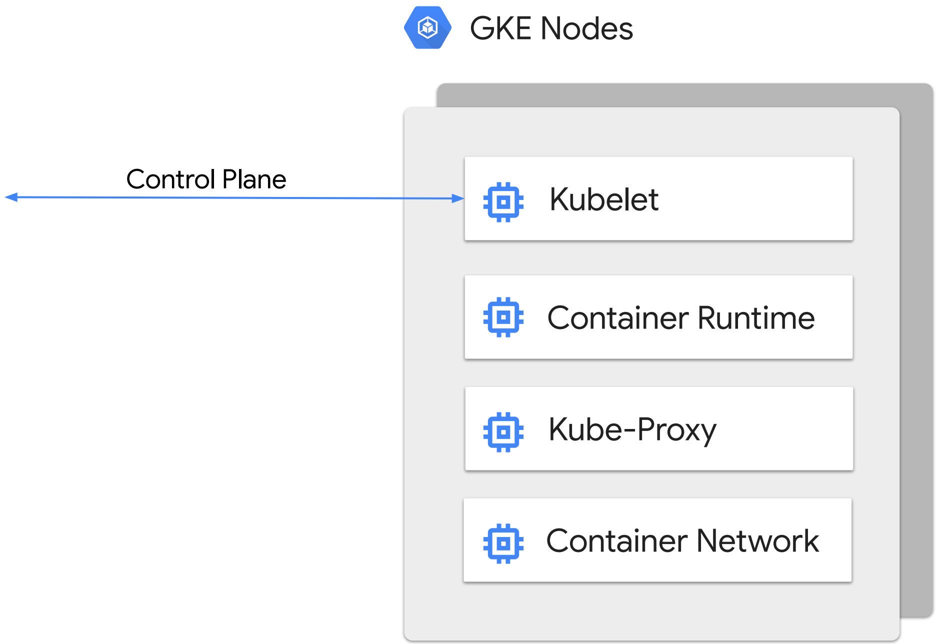
Google Cloud

Managed K8s Nodes

Nodes in GKE run in **customer projects**, and...

GKE provides **automation** to help **keep nodes healthy and up-to-date**

GKE Nodes can run either Container-Optimized OS or Ubuntu



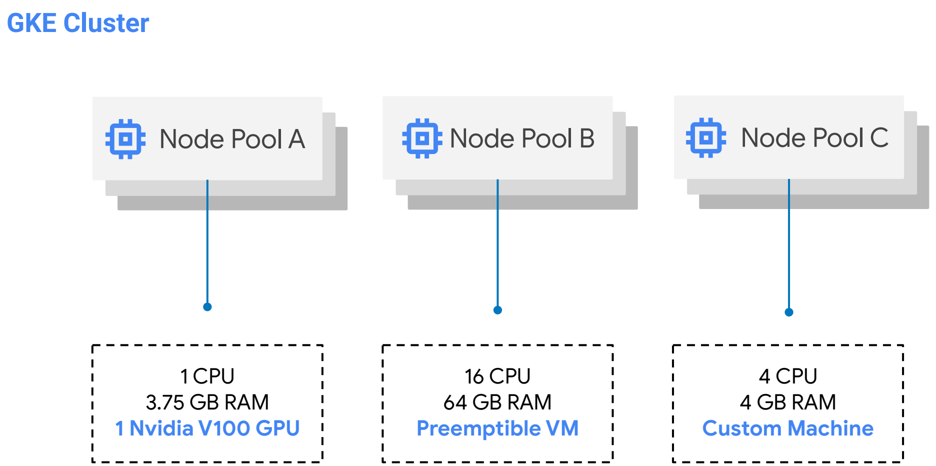
Google Cloud

Node Pools for Diverse Workloads

GKE Clusters support **multiple Node Pools** with heterogeneous resources.

Users can create Node Pools with:

- **Preemptible VMs**
- **GPUs or Local SSDs**
- **Custom Machine Types**



Google Cloud

Auto **Kubernetes**

Auto-repair

Automatically initiate repair process for nodes that fail a health check.

Auto-upgrade

Keep the control plane and nodes in the cluster up-to-date with the latest stable version

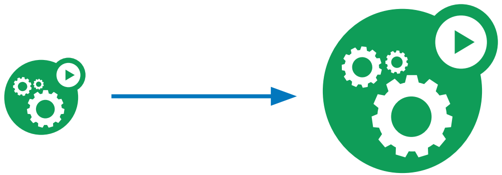
Auto-scale

Cluster autoscaling handles increased demand and scales back as needed

GKE Autoscaling Paradigms

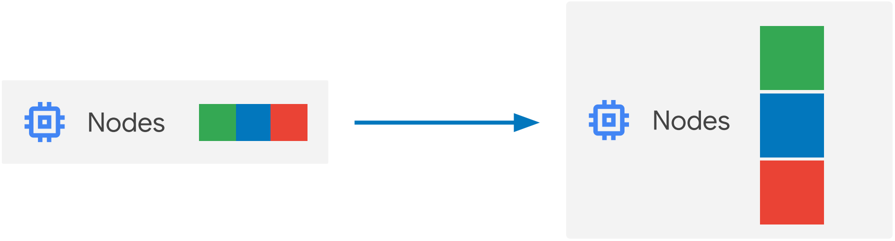
Scale Workloads Vertically
Vertical Pod Autoscaling

Triggers: VPA Recommendations



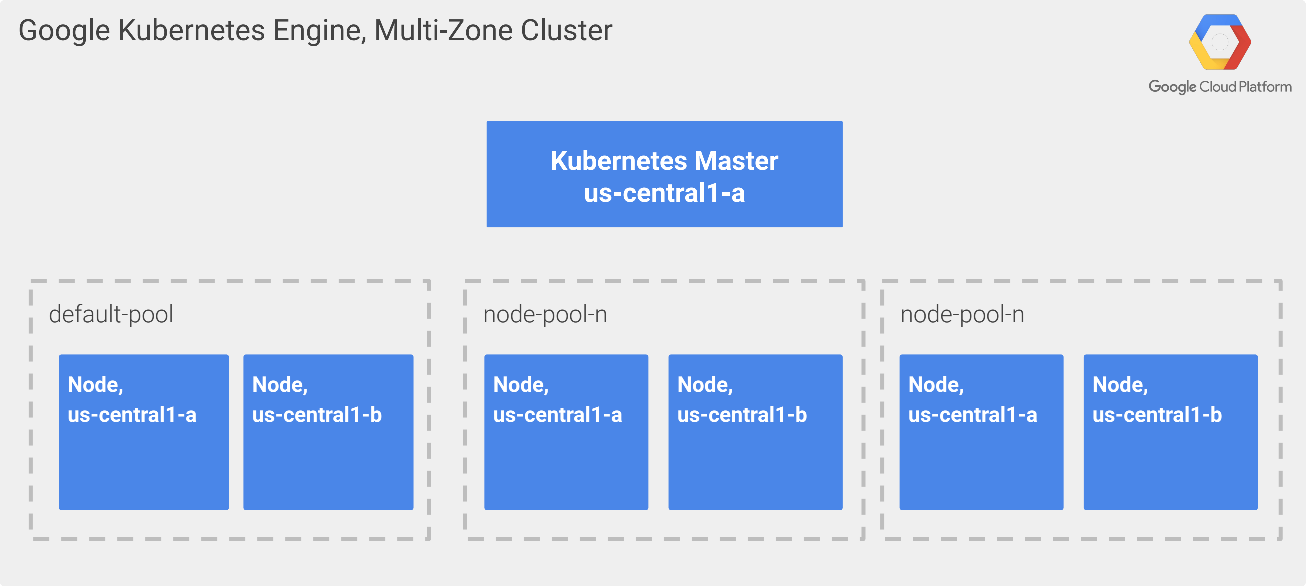
Scale Infrastructure Dynamically
Node Auto Provisioning

Trigger: Resources Required by Pods Larger than Existing Node Pools



Multi-zone and Regional Clusters

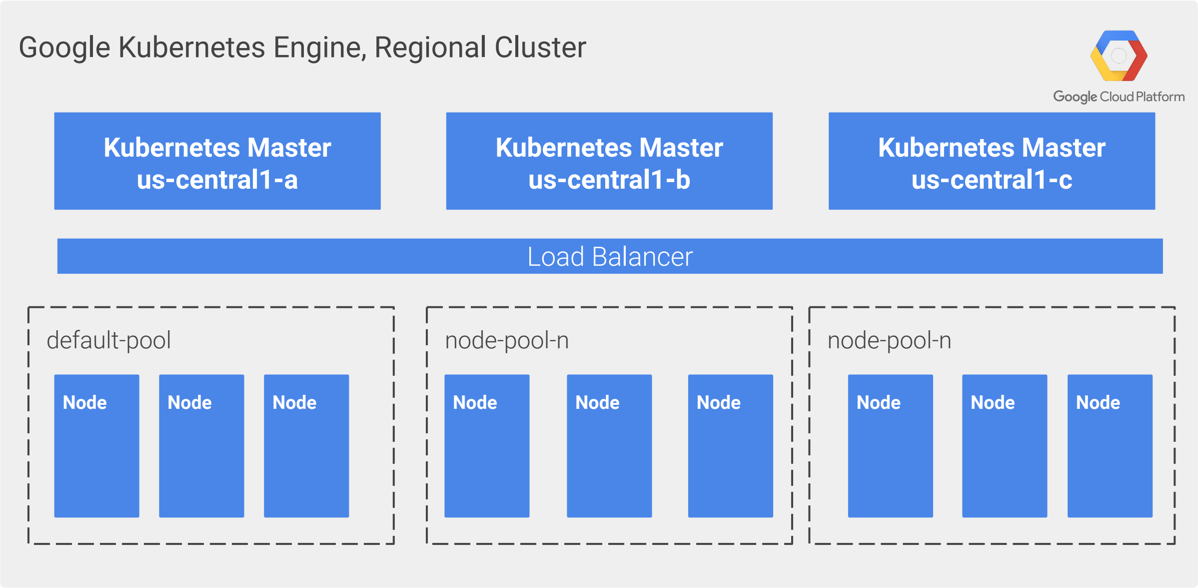
Multi-Zone Clusters: Enables higher service level by deploying nodes across multiple zones



Google Cloud

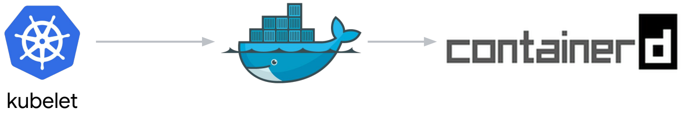
Multi-zone and Regional Clusters

Regional Clusters: Enables zero-downtime upgrades and 99.95% uptime by deploying multiple masters

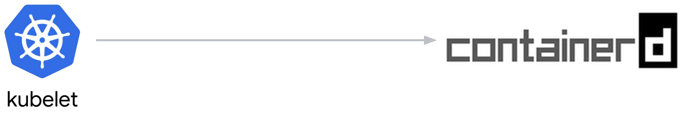


containerd runtime

- The full Docker runtime is largely unused by Kubernetes, and represents a large code surface-area
- containerd is the CRI-compliant minimal Docker component
- Available for node pools running COS and GKE 1.11+
- Use the new runtime-agnostic `crictl` utility to troubleshoot individual containers

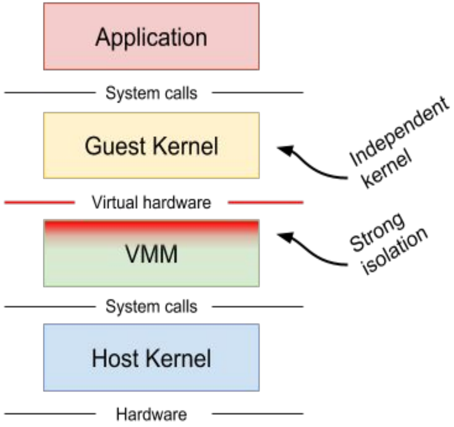


Previously, ***dockerd*** *was the proxy to* ***containerd***

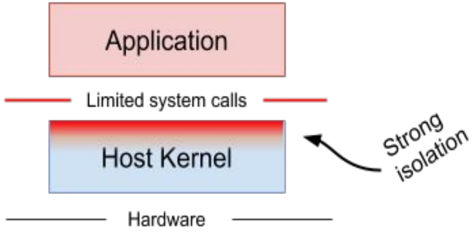


Now, the kubelet can speak directly to ***containerd***

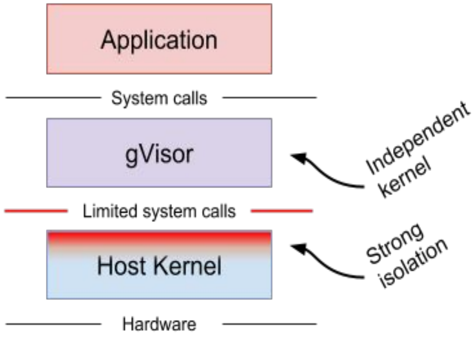
Sandbox Pods (gvisor runtime)



Machine-level virtualization



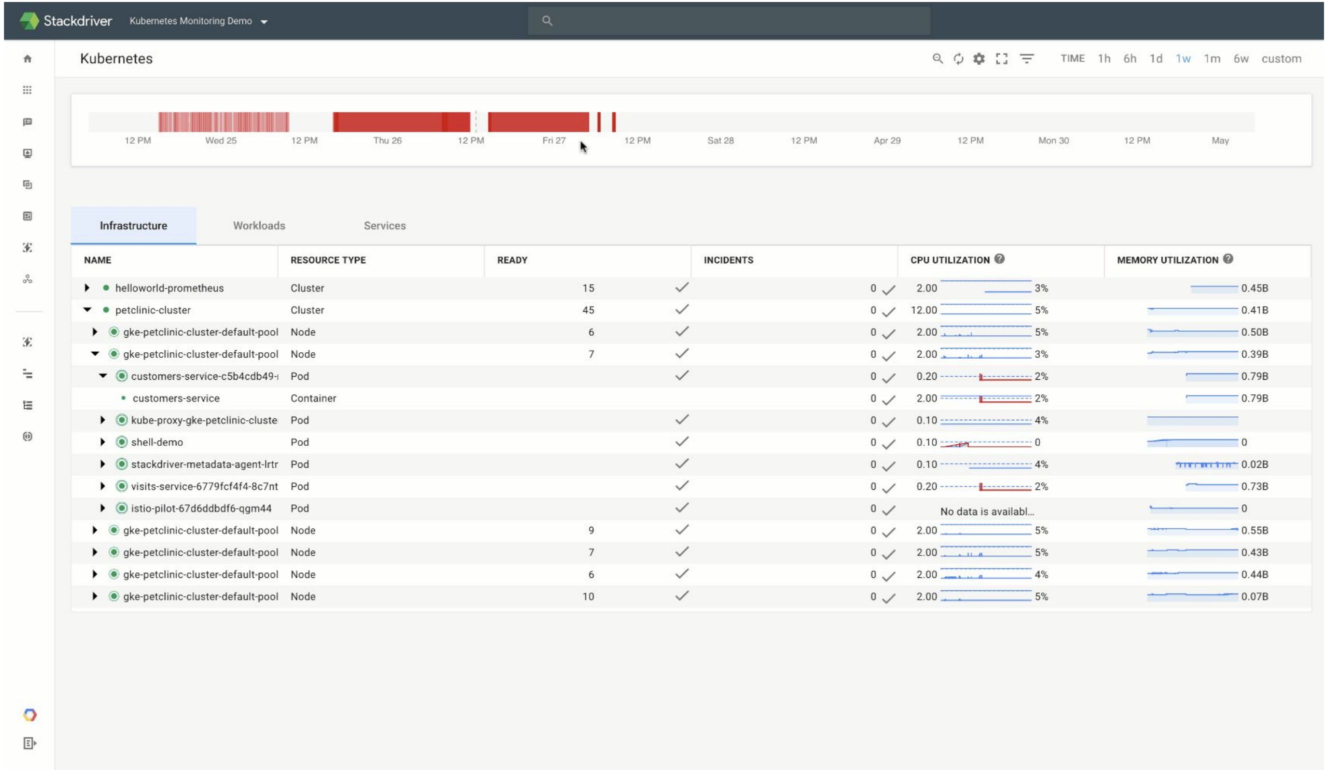
Rule-based execution



gvisor

Stackdriver Kubernetes Monitoring

- Kubernetes-aware monitoring
- Drill down through clusters, nodes and pods right through to the container



Google Cloud

Knative

- Building-blocks for serverless workloads – on Kubernetes
    - Serverless without the lock-in of serverless!
- Three main components
    - **Build** – turns your code into runnable containers
    - **Serving** – revisions, traffic splitting, autoscaling
    - **Eventing** – enables late-binding to event sources and consumers, consistent with the emerging CloudEvents specification
- Backed by Google, Pivotal, IBM, RedHat, and SAP.

Knative

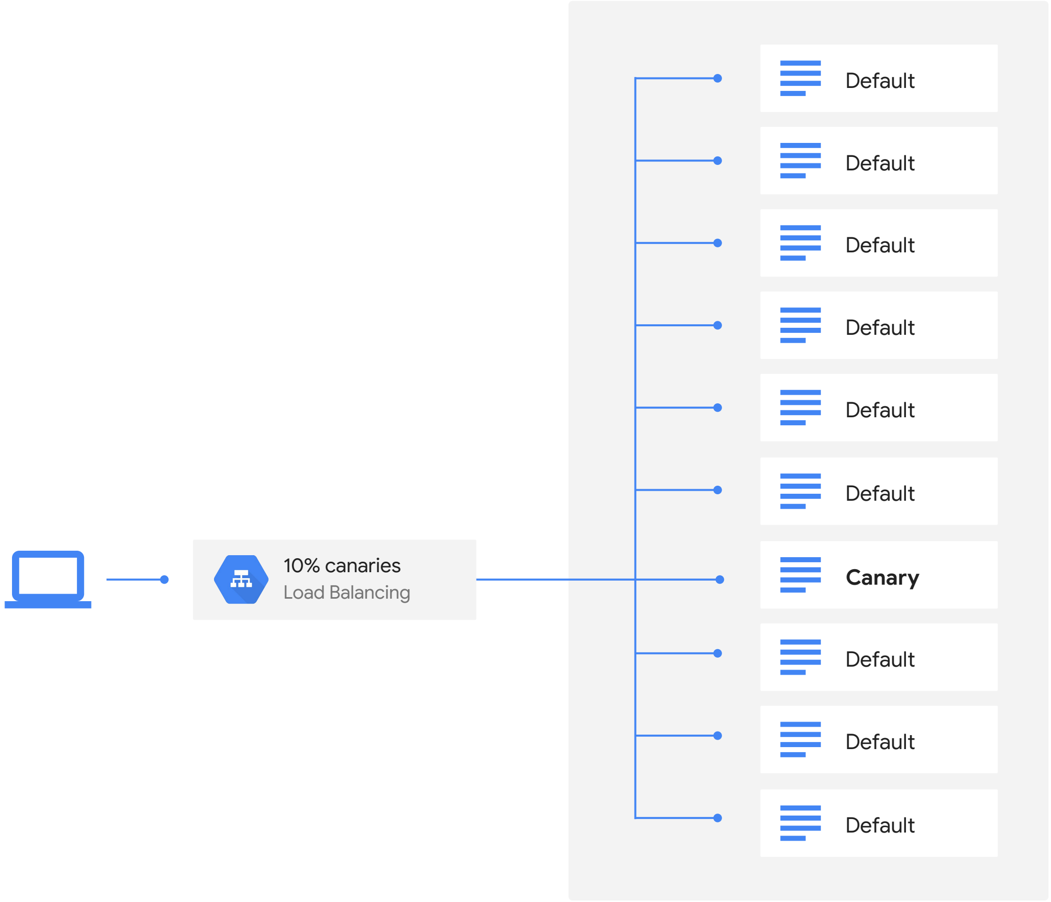
Istio
Service Mesh

03

Google Cloud

In the past

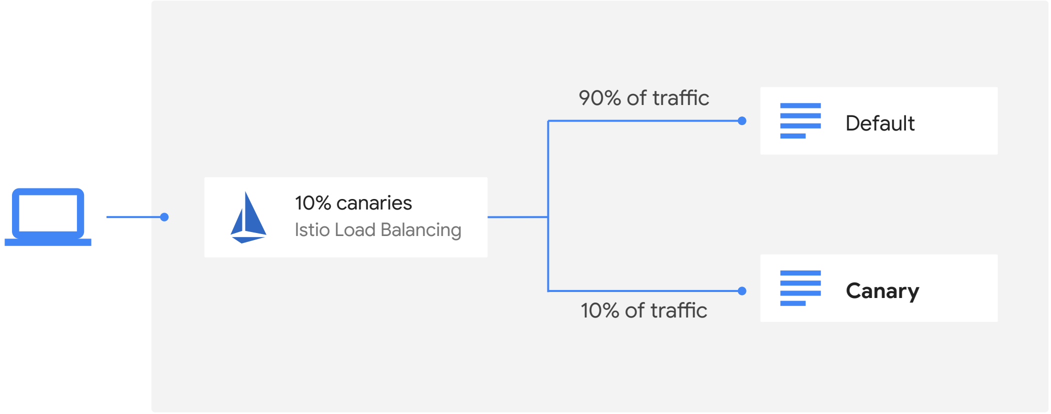
Traffic control tied to infrastructure



Google Cloud

With Istio

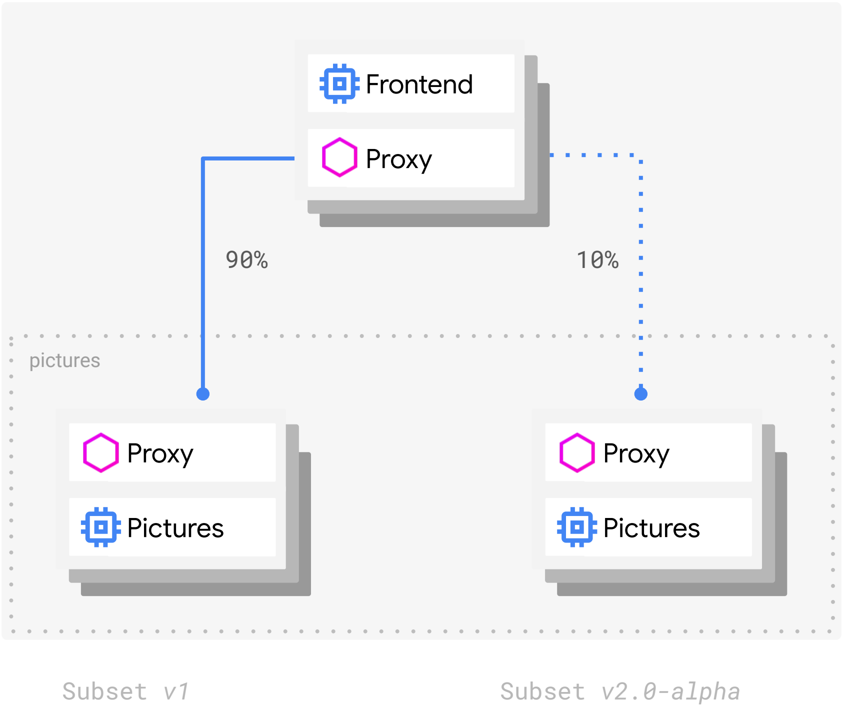
Traffic flow *separated* from infrastructure



Google Cloud

App Rollout

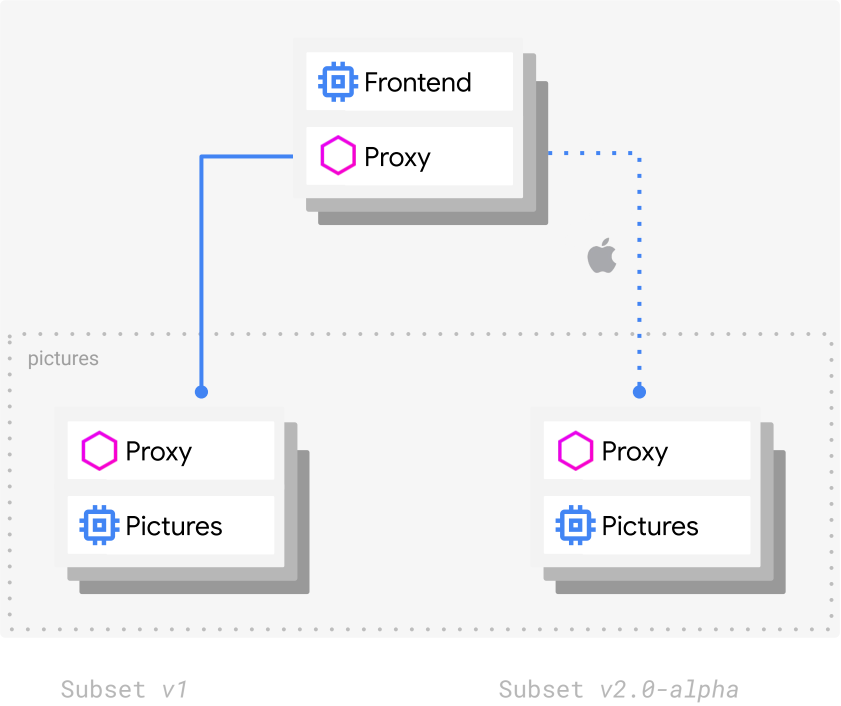
```
hosts:
  - pictures
http:
  - route:
    - destination:
        host: pictures
        subset: v1
      weight: 90
    - destination:
        host: pictures
        subset: v2.0-alpha
      weight: 10
```



Google Cloud

Traffic steering

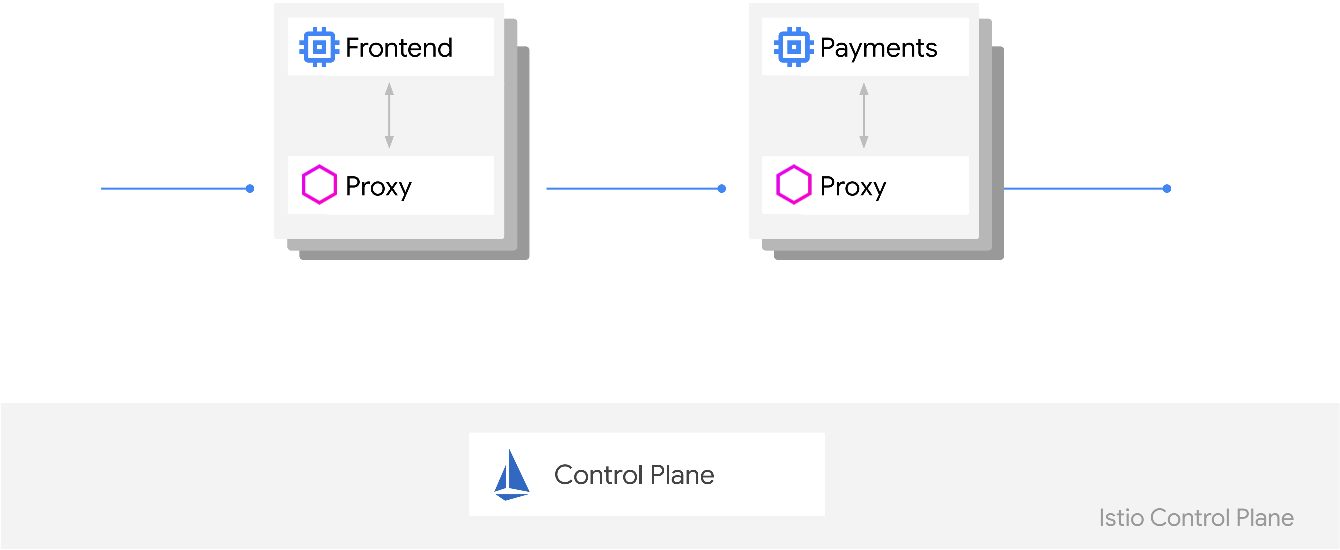
```
hosts:
  - pictures
http:
  - match:
    - headers:
        user-agent:
          regex: ^(.*?;)?(iPhone)(;.*)?$
    route:
    - destination:
        host: pictures
        subset: v2.0-alpha
  - route:
    - destination:
        host: pictures
        subset: v1
```



Envoy, the Istio service proxy

- A C++ based L4/L7 service proxy
- Extensible with the concept of L4/L7 "filters"
- Battle-tested @ Lyft
- Traffic routing and splitting, health checks, circuit breakers, timeouts, retry budgets, fault injection, …
- HTTP/2 & gRPC
- Transparent proxying, designed for observability
- Control plane config protocol xDS

With Istio



Google Cloud

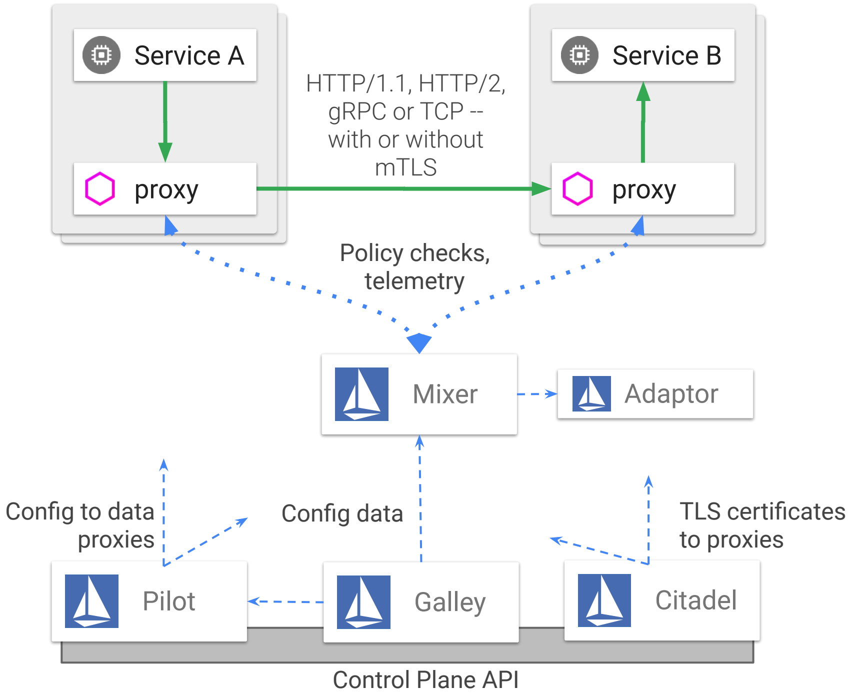
Istio Architectural Components

Pilot: Control plane to configure and push service communication policies.

Mixer: Policy enforcement with a flexible plugin model for providers for a policy.

Citadel: Service-to-service auth[n,z] using mutual TLS, with built-in identity and credential management.

Galley: Validates user config on behalf of the other control plane components



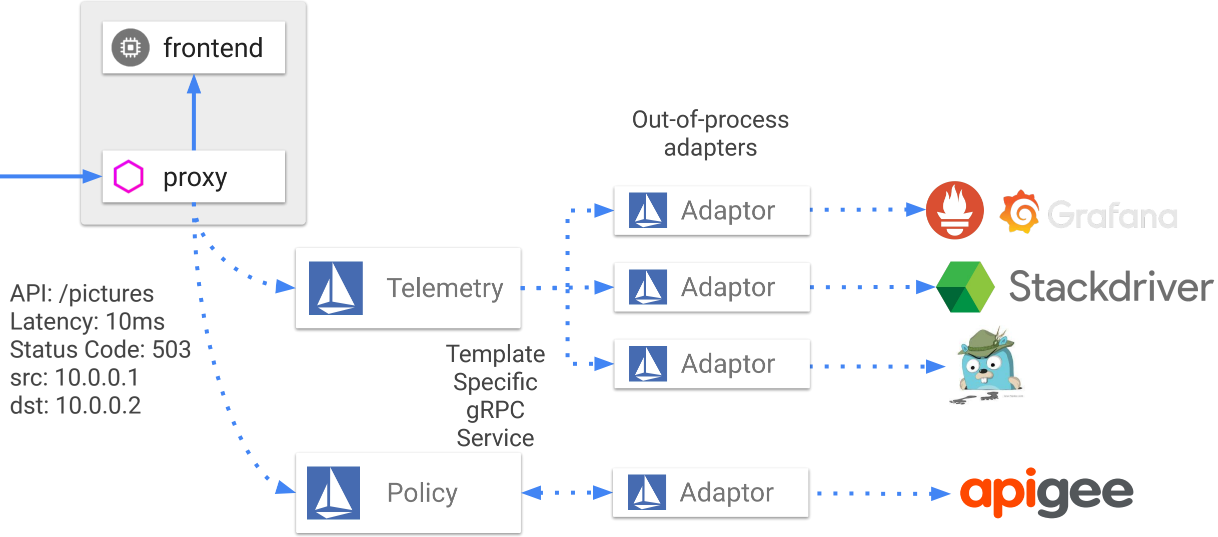
Google Cloud

Mixer: Extensibility

Mixer has an open API and a pluggable architecture: Send telemetry, logs and traces to your system of choice

Out-of-process adapters allows independent scaling of mixer and the adapter, add additional backends without having to redeploy mixer

Istio 1.1 defaults: Telemetry enabled, Policy disabled



https://github.com/istio/istio/tree/master/mixer/adapter

Google Cloud

That’s a wrap!