Kubernetes & Hybrid Deployments

Sandeep Parikh
Head of Solutions, Americas East
Google Cloud
@crcsmnky

Hey, That's Me!

I run the Americas East half of the Google Cloud Solutions Architecture team. We build repeatable architectural patterns and guidance in the form of whitepapers, code, etc.

Before Google, I was at MongoDB, Apple, and a bunch of startups. I live in Austin. It's hot there. Seriously.

Find me on Twitter @crcsmnky

Glossary

Things you probably already know but it doesn't hurt to cover just in case.

Kubernetes is a system for managing clusters of containers, including orchestration, scheduling, etc.

Pods are the deployable units in a cluster. Pods have one or more tightly coupled containers.

Services define abstractions across a logical set of Pods and a policy to access them

Replica Sets ensure that a number of Pods are running at any given time.

Namespaces provide "virtual clusters" backed by the same physical cluster.

Container Engine is a service for deploying managed Kubernetes clusters in Google Cloud.

Table of Contents

Deployment Types

Example Use Cases

Things to Remember

Getting Started

Deployment Types

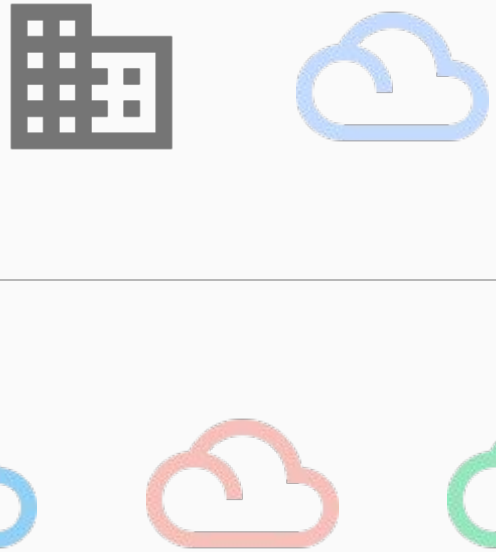
Deployments

Hybrid

Heterogeneous

Multi-Cloud

Public/Private



Why Heterogeneous?

Maxed out resources

Limited geo reach

High Availability

Avoid Vendor Lock-In

Compute Flexibility

Access to services

Heterogeneous is Hard™

Example Use Cases

Use Cases

Splitting traffic across multiple deployments

Multi-cloud deployments for high availability

Multi-cloud for geographic reach

Fronting on-premise data with cloud

Using cloud for dev/test workloads

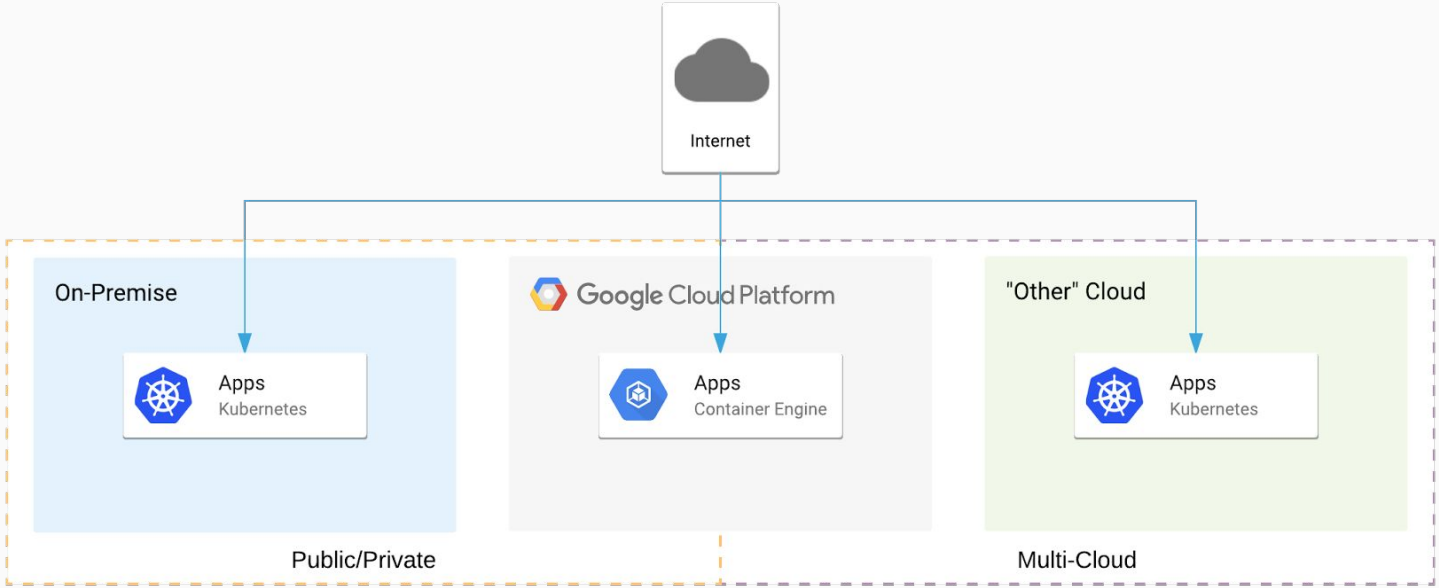
Multi-Cloud

Traffic Splitting

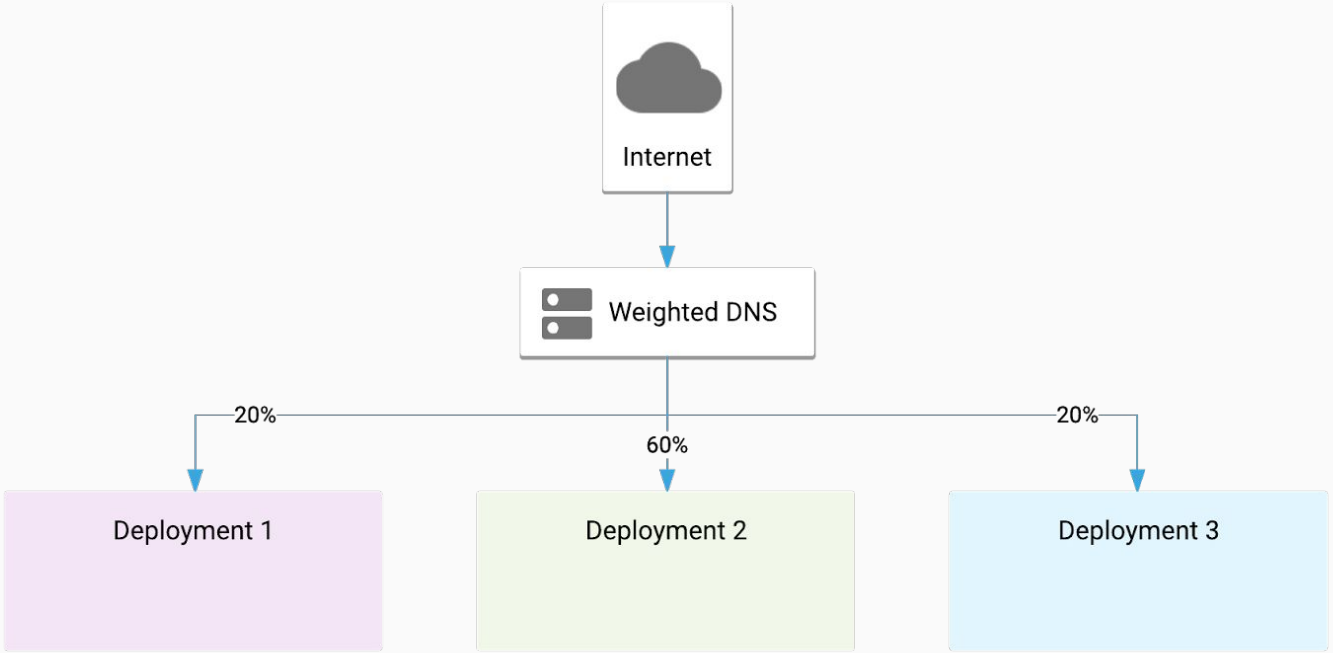
High Availability

Geographic Reach

Deployment Types



Incoming Requests

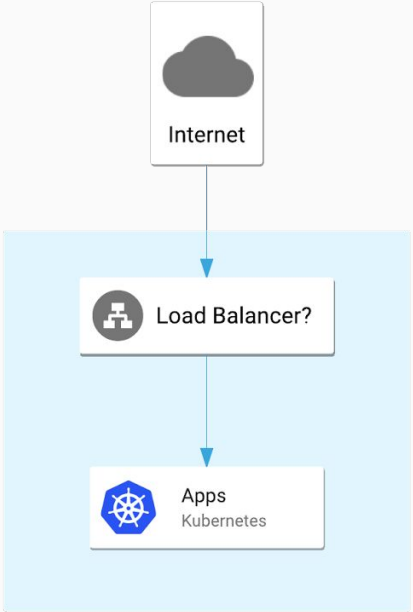


Dyn

Akamai

NS1.

Handling Requests




```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: [NodePort | LoadBalancer]
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
```

Handling Requests with Ingress

"An Ingress is a collection of rules that allow inbound connections to reach the cluster services."

https://kubernetes.io/docs/user-guide/ingress/

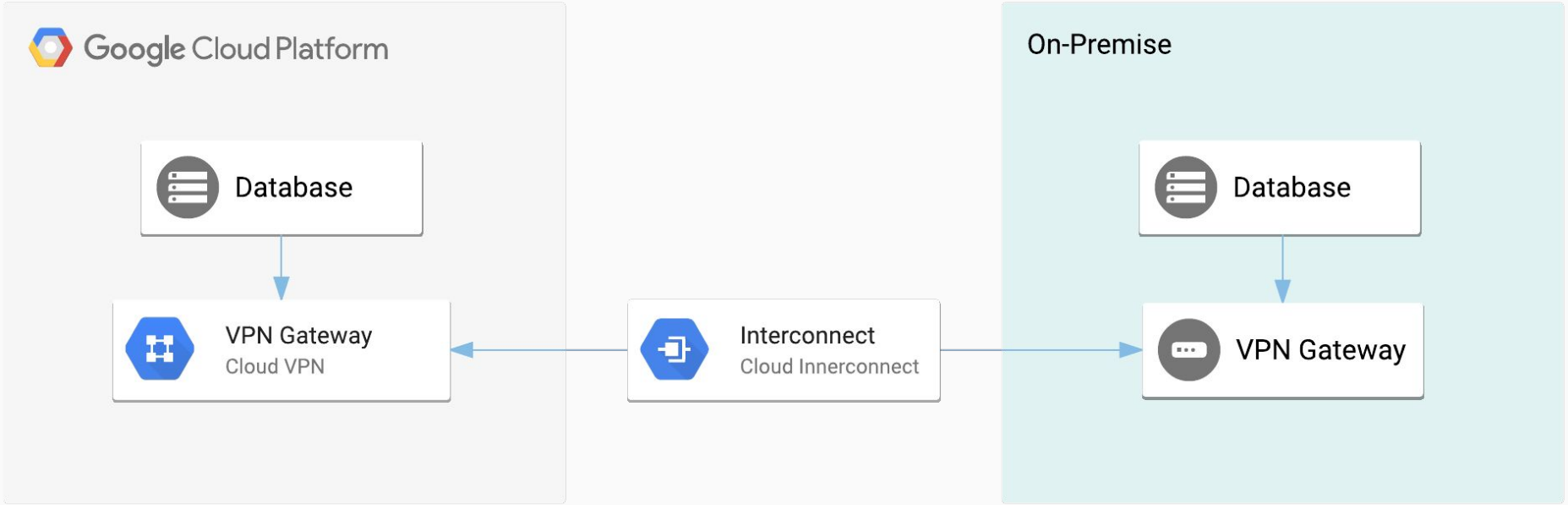
Services are Layer 4 (IP + Port)

Ingress (beta) is Layer 7

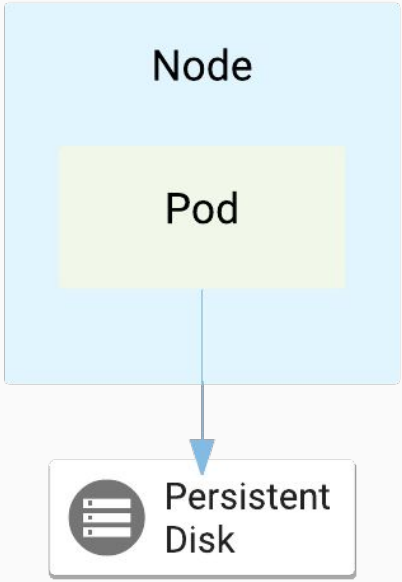
Ingress maps incoming traffic to backend services

- By HTTP host headers
- By HTTP URL paths

Shared Services



Stateful in Kubernetes



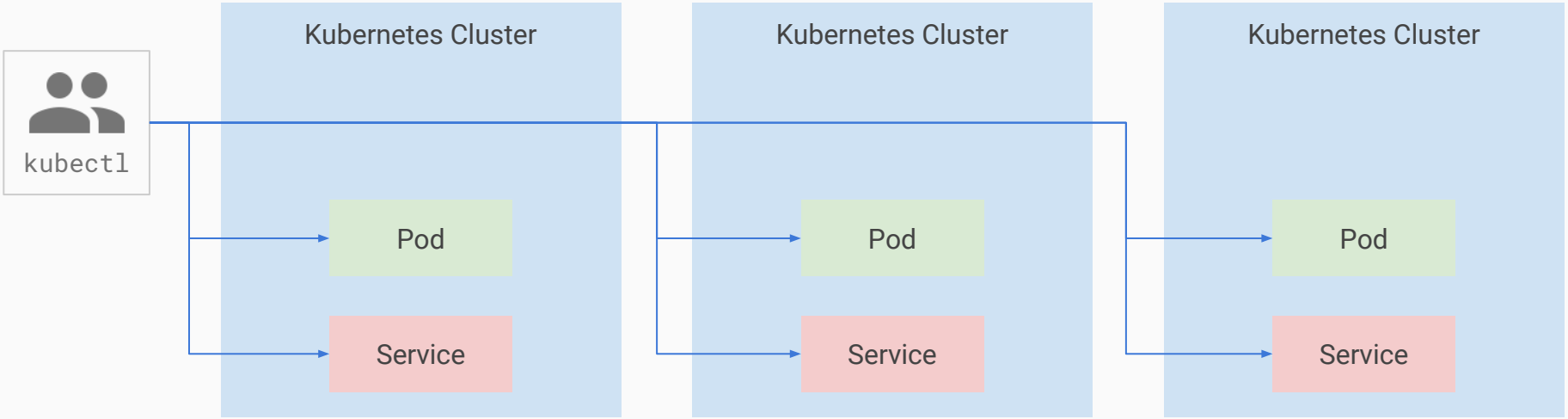
Good

- Startup/teardown ordering
- Stable hostname, available in DNS
- Peer discovery

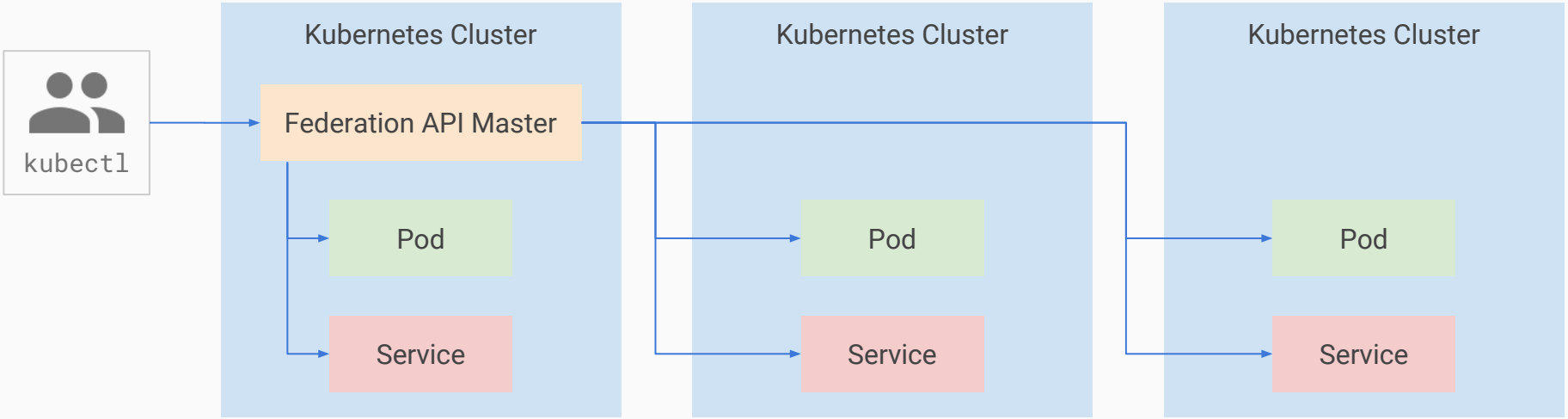
Not So Good

- Only so much disk bandwidth available in multi-pod nodes
- Might have snowflake nodes with one big pod per node
- Scaling/ops of certain systems might not match Kubernetes

Naive Deployment



Deploying With Federation



Federation

Why Federation

Sync resources across clusters

Cross-cluster service discovery

Highly available applications

Why Not Federation

Increased network bandwidth and cost

Reduced cross-cluster isolation

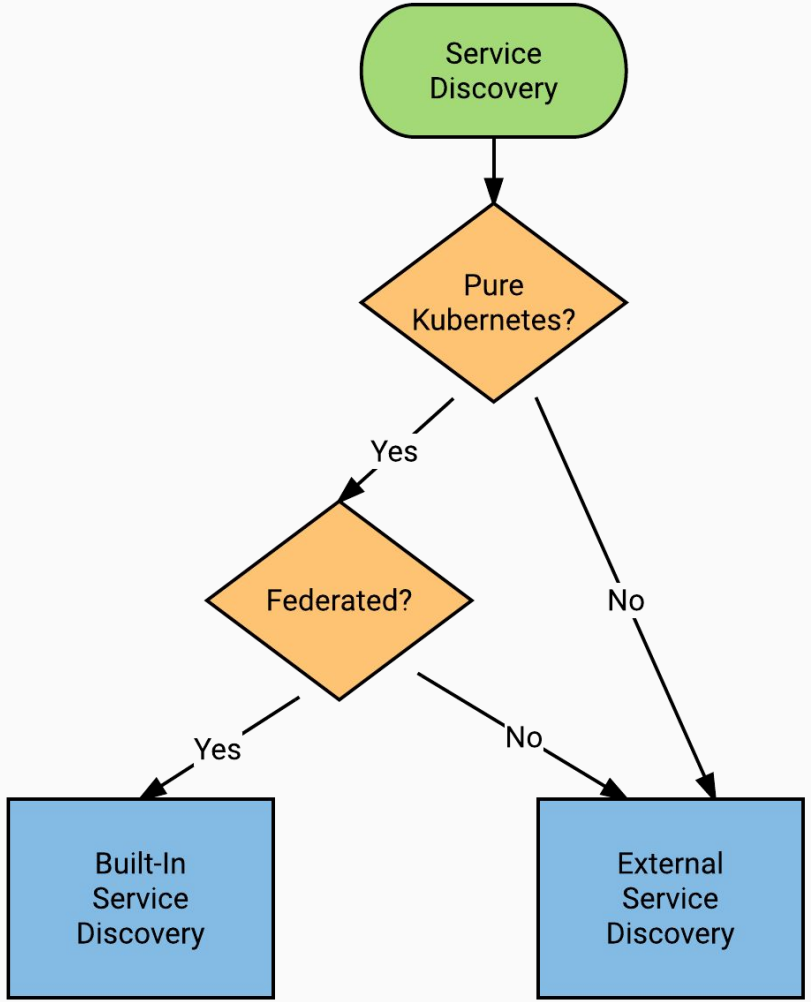
Each deployment is a snowflake

Service Discovery

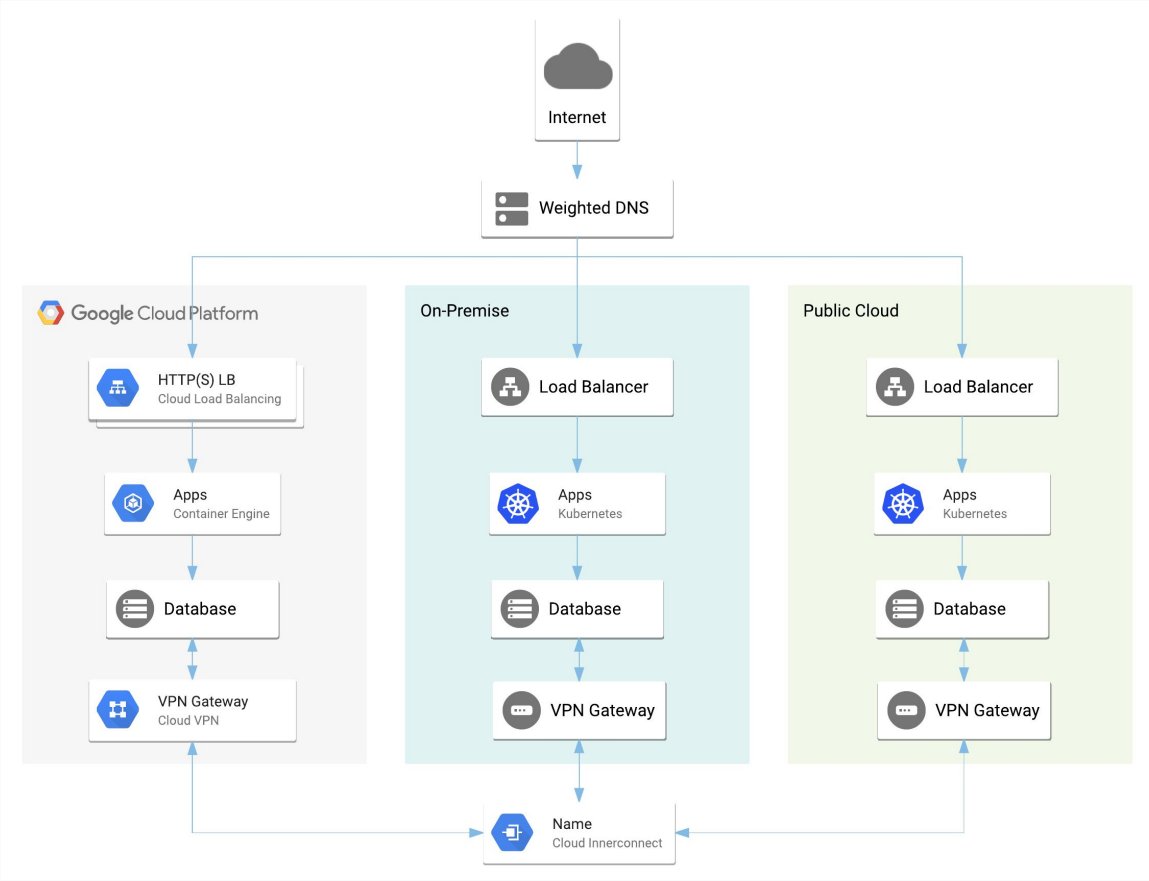
Consider long term deployment architecture

Cross-cloud networking is required

Shared services are important to consider as well



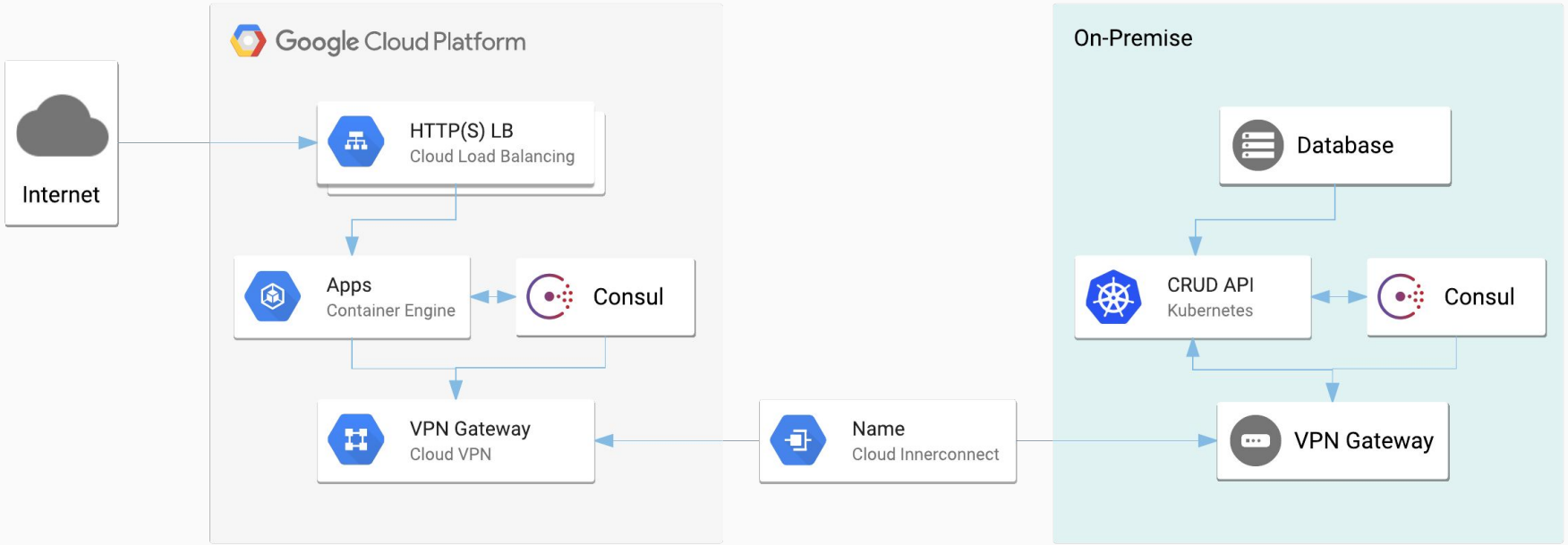
Heterogeneous Deployment



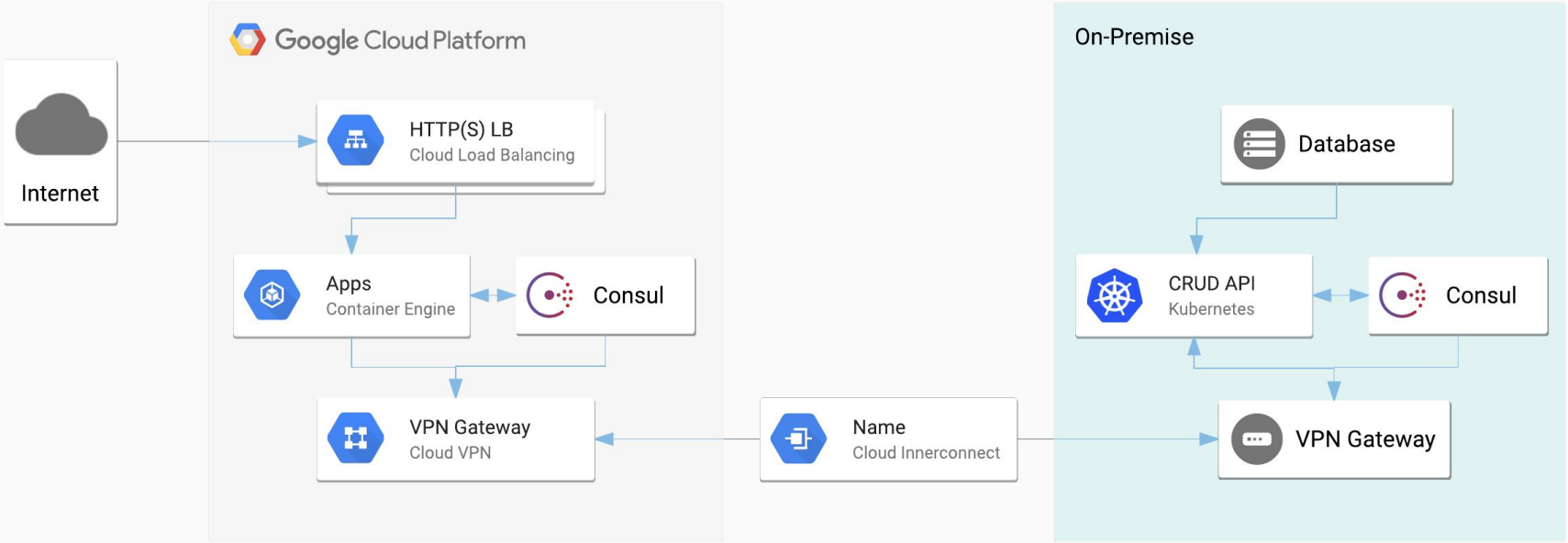
Fronting On-Premise Data

Cloud applications accessing on-premise (or private) data systems

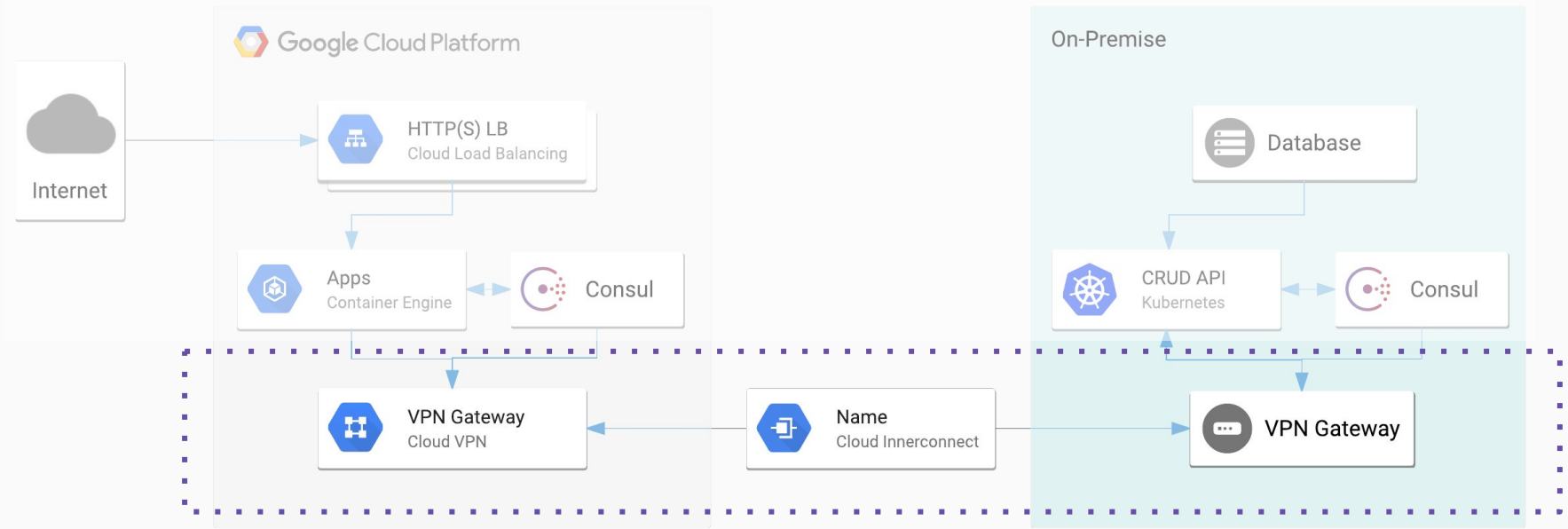
Deployment Architecture



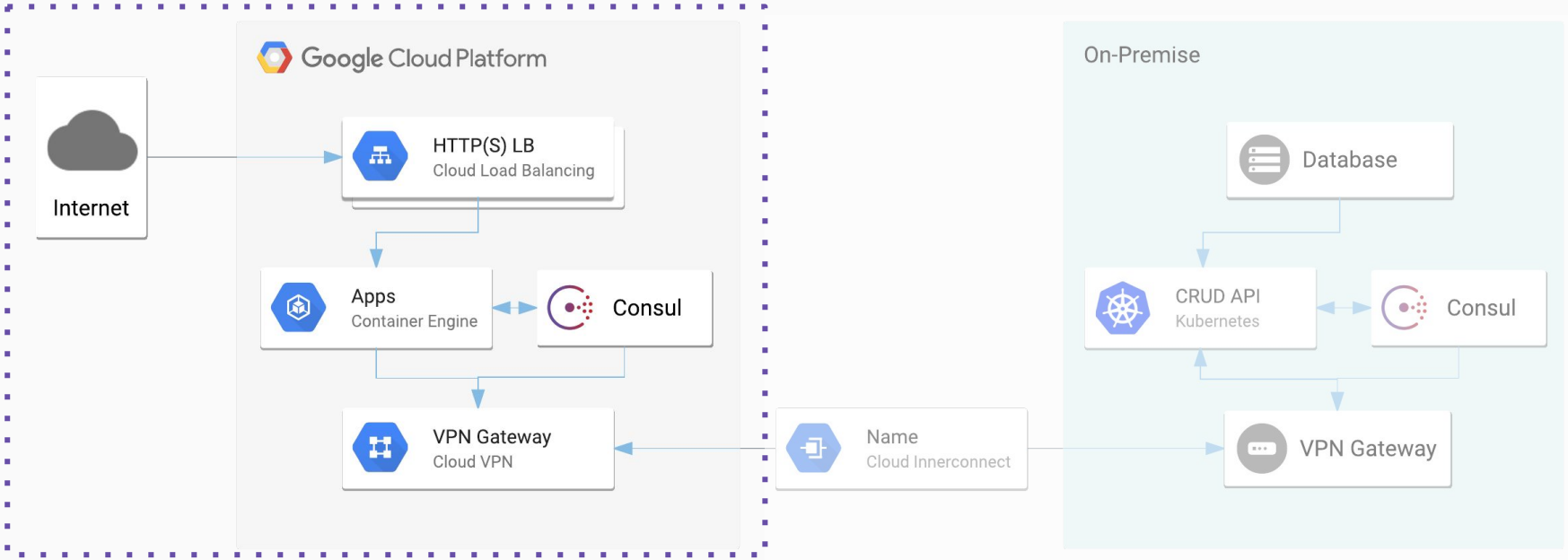
Component Review



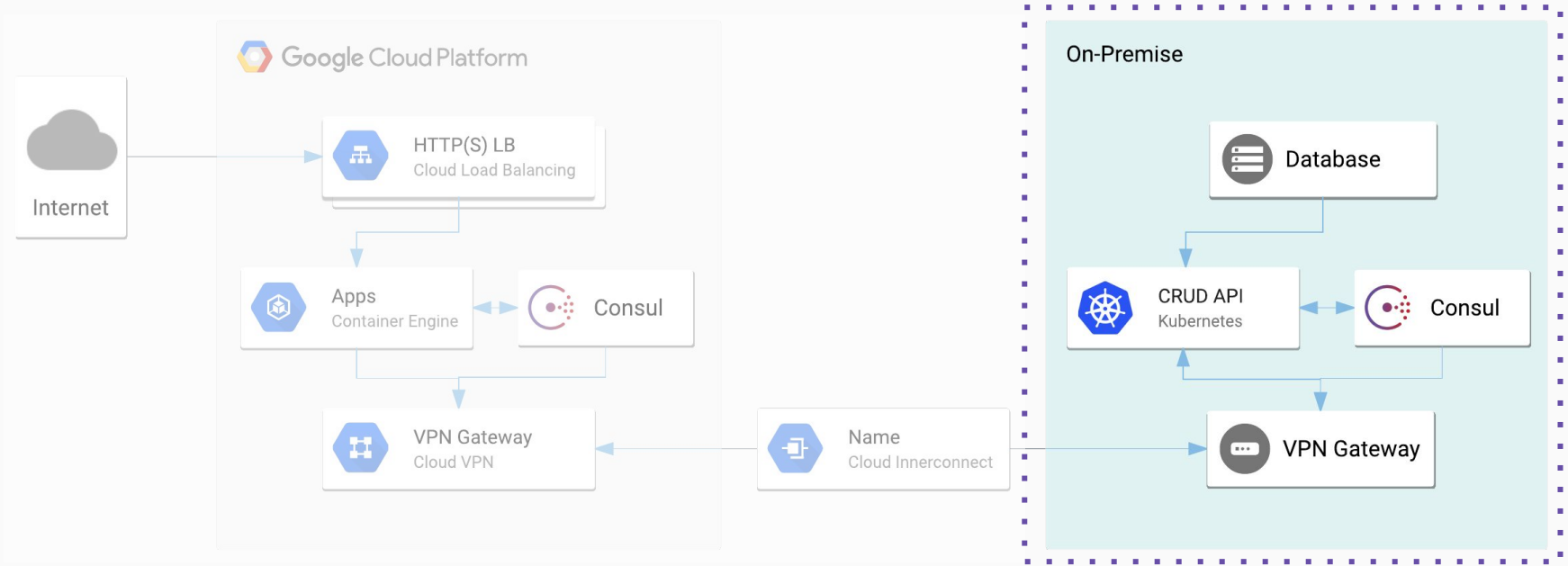
Networking


Google Cloud Platform
On-Premise
Internet
HTTP(S) LB
Cloud Load Balancing
Database
Apps
Container Engine
Consul
CRUD API
Kubernetes
Consul
VPN Gateway
Cloud VPN
Name
Cloud Innerconnect
VPN Gateway

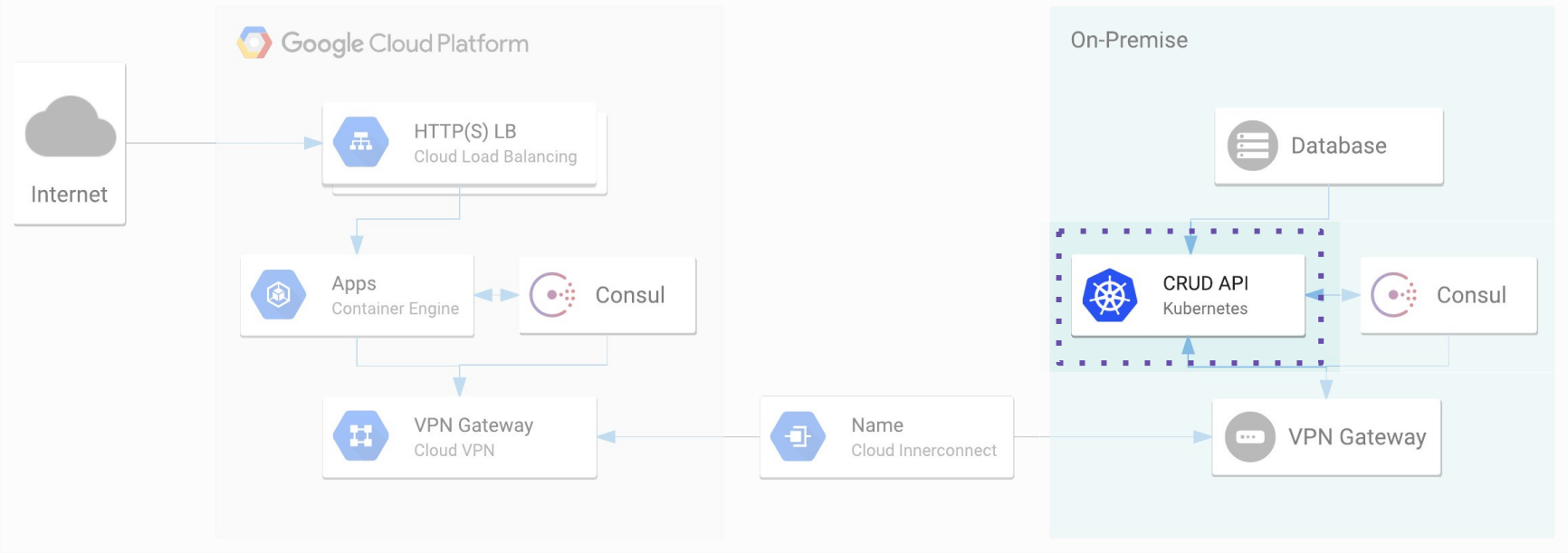
Cloud Architecture



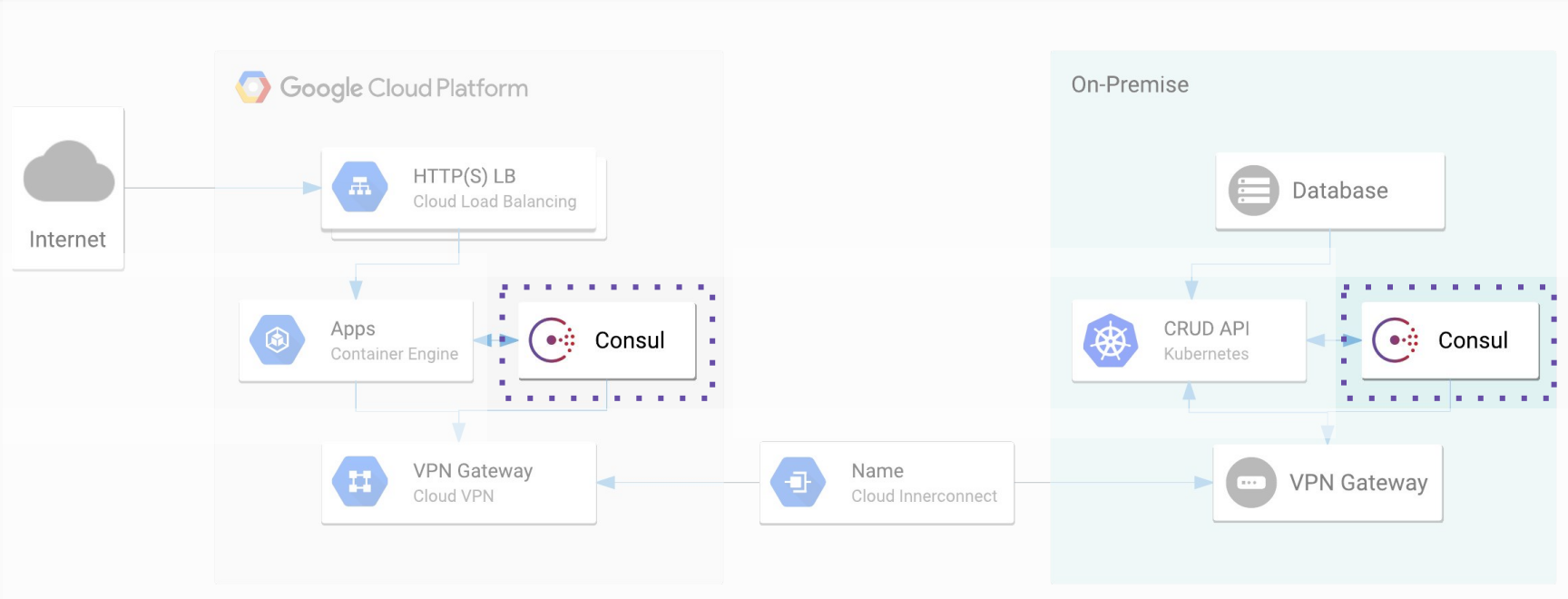
On-Premise Architecture



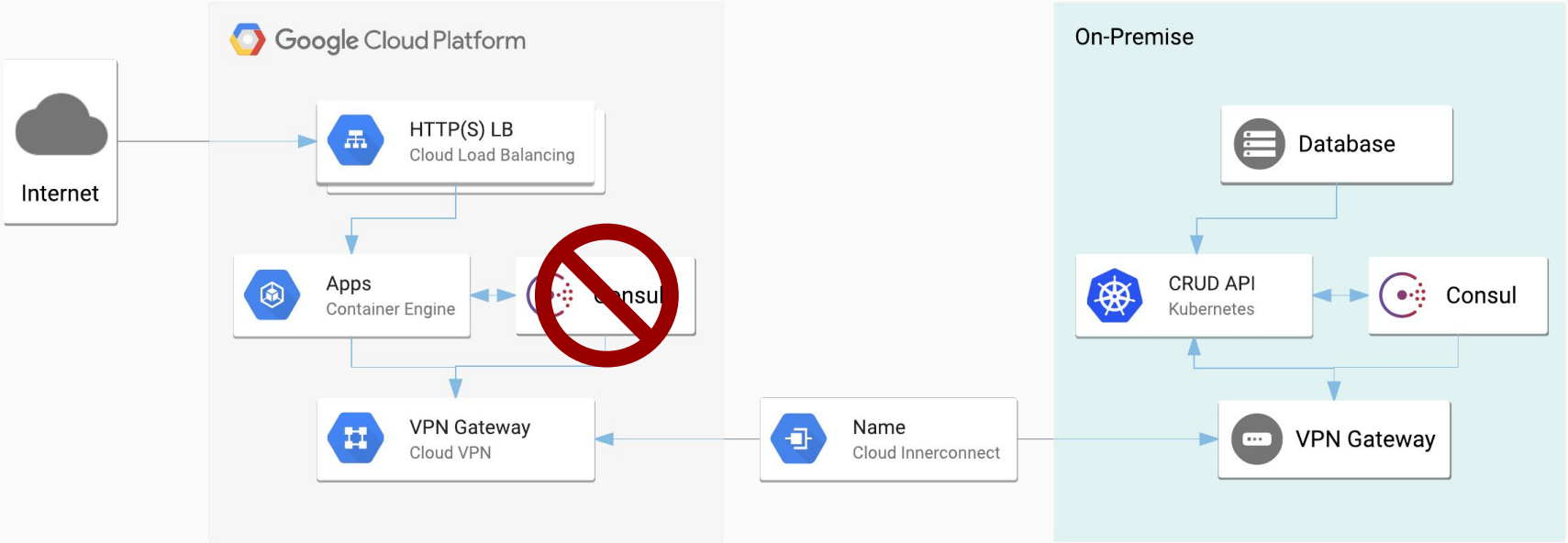
Kubernetes On-Premise



Service Discovery



Service Discovery with Kubernetes 1.6



https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG.md#dns

Considerations

Shared Services

- Each deployment is standalone
- Nothing (e.g. databases) shared across deployments
- ...Except Service Discovery (e.g. Consul, Linkerd, etc.)

Federation

- Not necessary here; each deployment is standalone
- Federated control plane would add unnecessary overhead

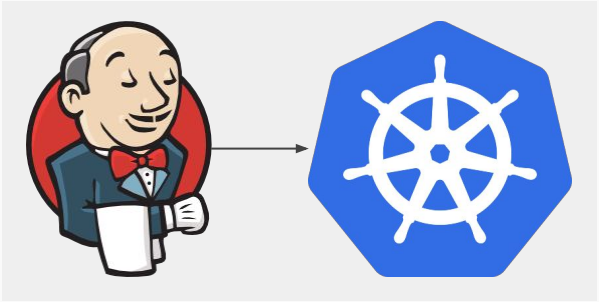
Short Term / Long Term

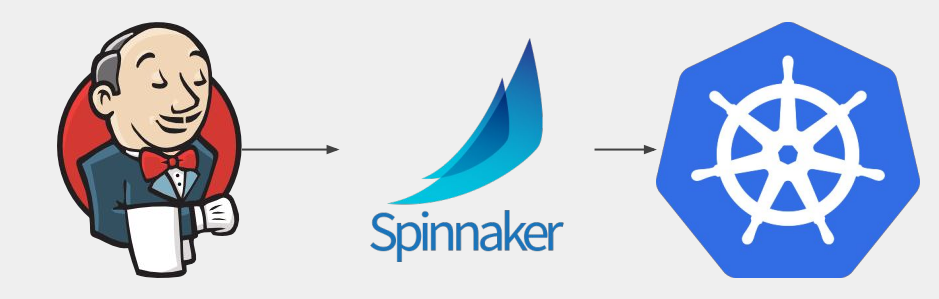
- CRUD has short and long term benefits
- Managing authn and authz back to database
- Measuring utilization and performance
- Building a path to (some) data migration

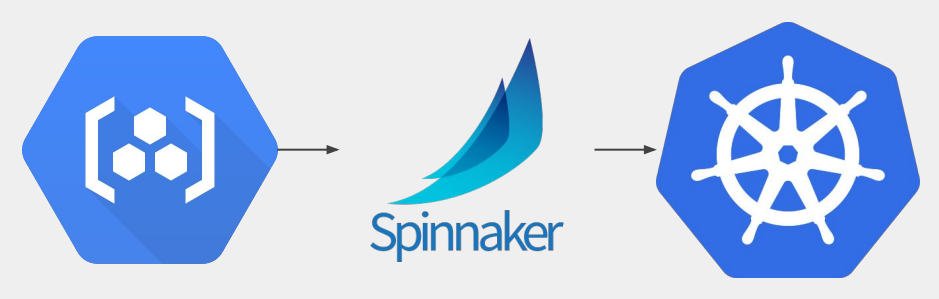
Hybrid Dev & Test Workloads

Using cloud to run build pipelines and orchestrate CI/CD workflows

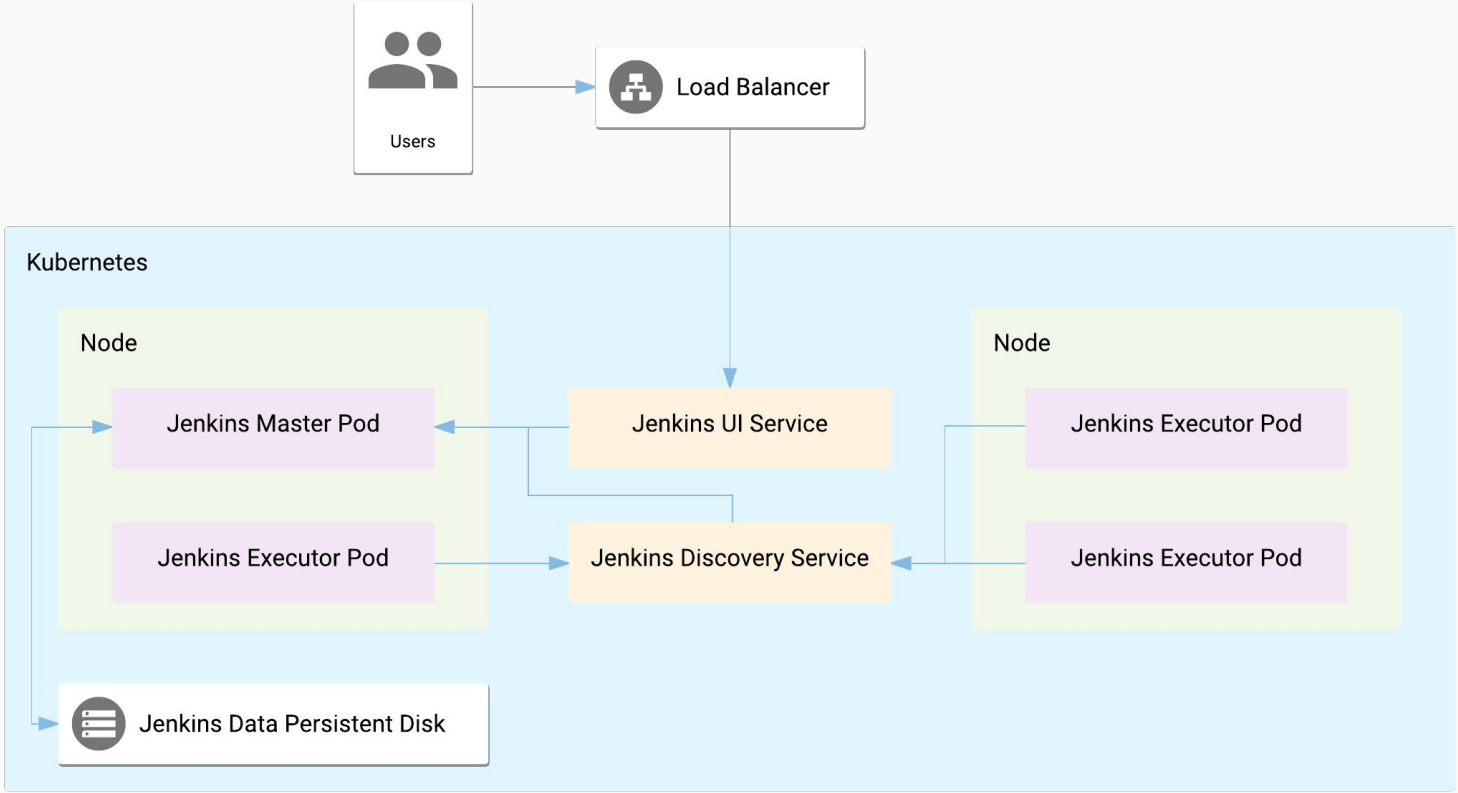
Approaches





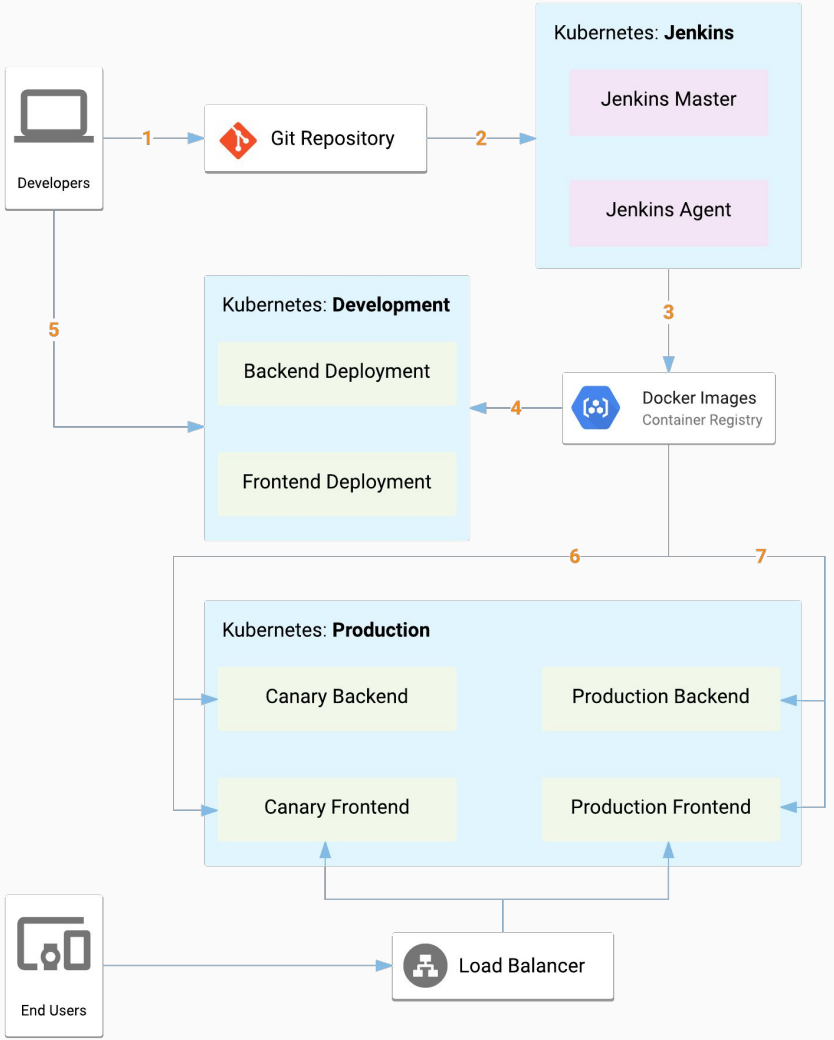


Jenkins and Kubernetes



Workflow

1. Developer commits code to development branch
2. Tests get kicked off and container image built
3. Container image uploaded to registry
4. Developer environment deployed
5. Iterate and test then commit to canary branch
6. Container image promoted to canary
7. Container image promoted to production



Configuration

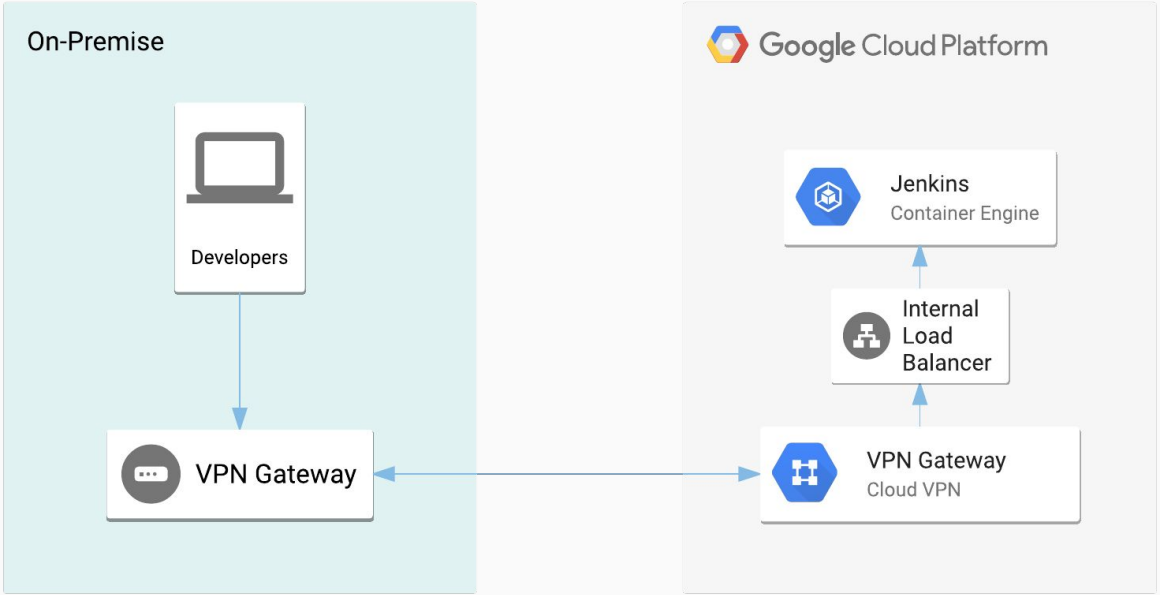
Master

- UI exposed via NodePort + Load Balancer
- Discovery internally via ClusterIP
- Replica Set of 1
- Resource limits!

Workers

- Jenkins Master -> 0 executors
- Add “volumes” for Docker and Docker socket
  `/usr/bin/docker`
  `/var/run/docker.sock`

Networking


On-Premise
Developers
VPN Gateway
Google Cloud Platform
Jenkins
Container Engine
Internal
Load
Balancer
VPN Gateway
Cloud VPN

Spinnaker

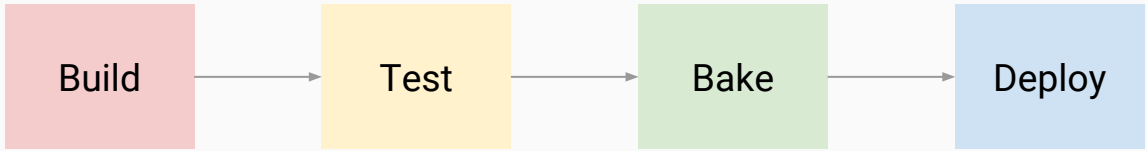
Orchestrating continuous delivery pipelines

Cluster Management

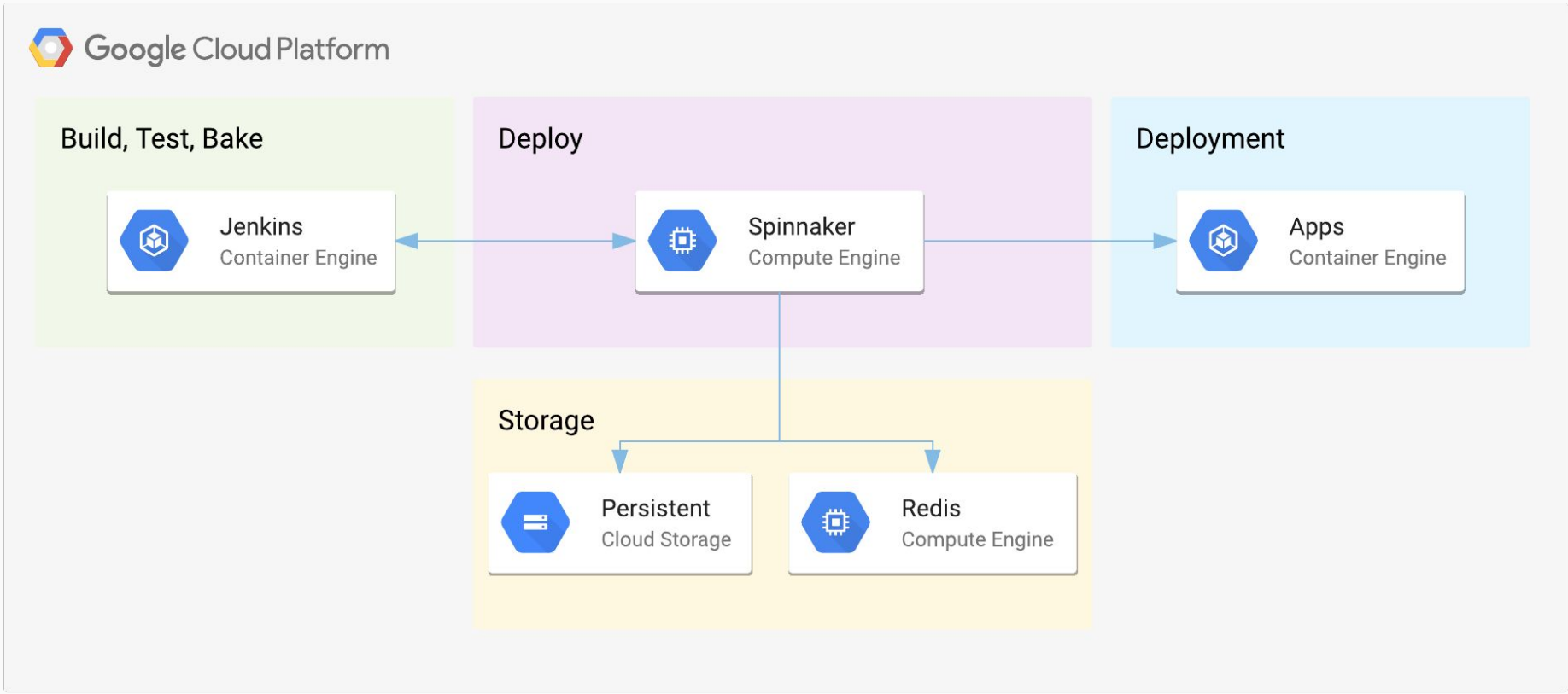
- Instance Groups
- Firewalls
- Load Balancers
- Instances

Deployment Management

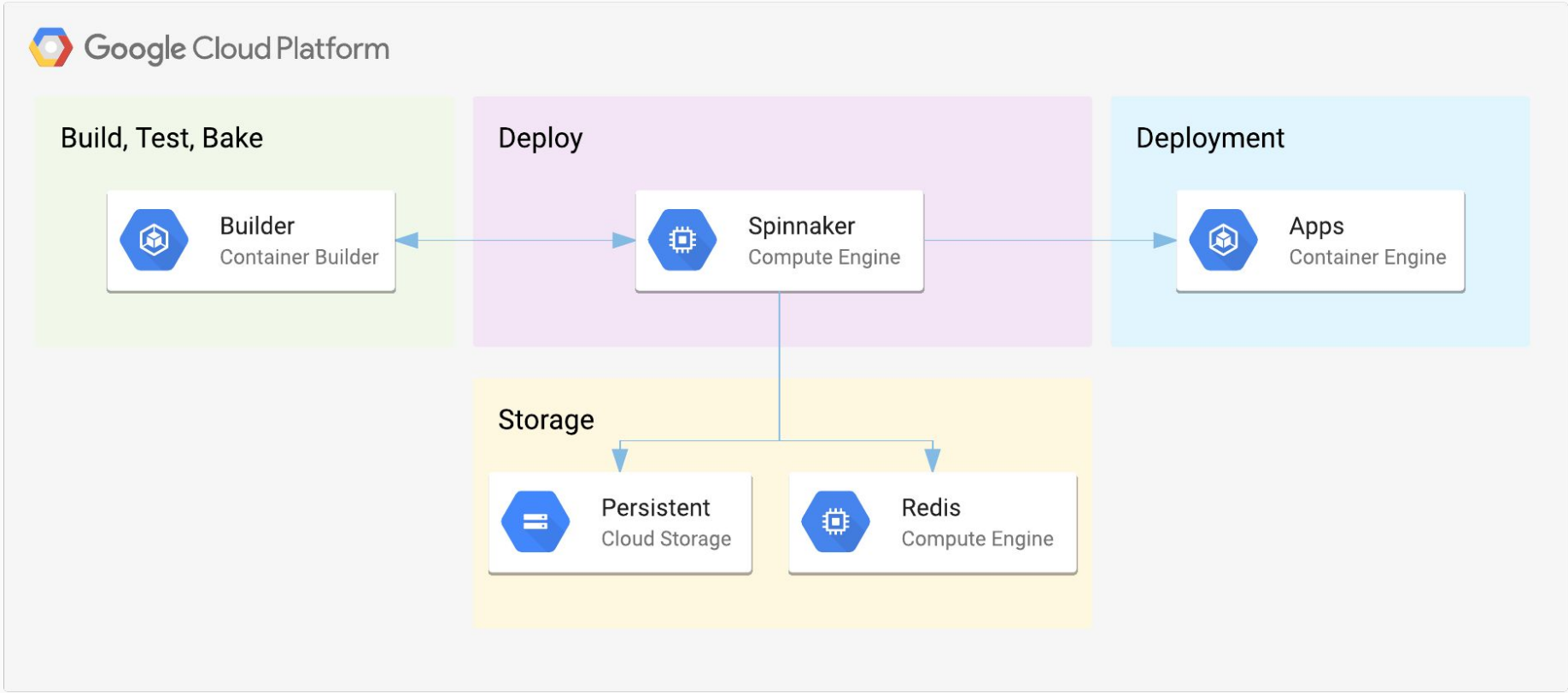
- Pipelines
- Stages
- Tasks



Jenkins, Spinnaker, and Kubernetes

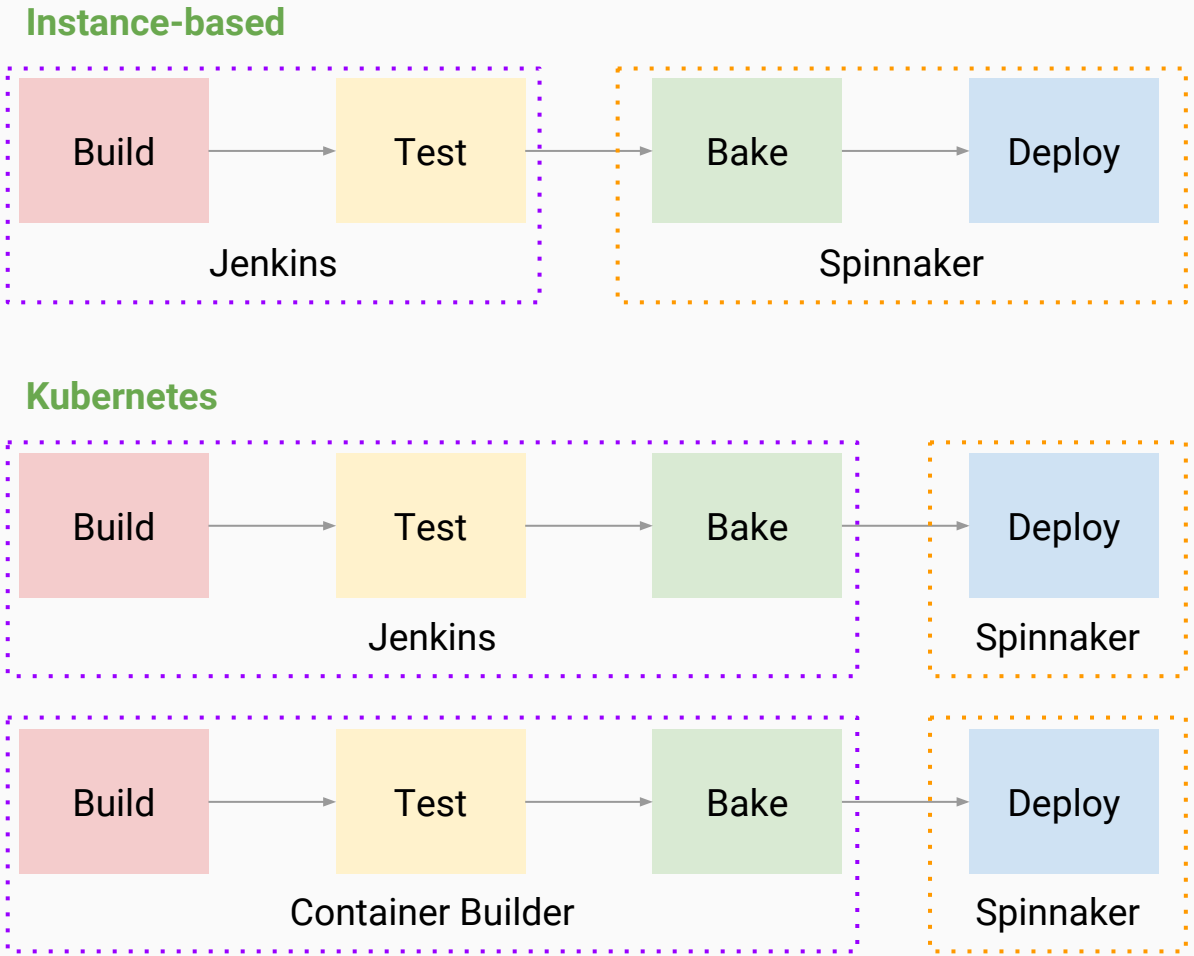


Container Builder, Spinnaker, and Kubernetes



Spinnaker

What does what and when



Container Builder

Container Builder executes your build by running commands in a Docker container.

Consistent and secure build environment

Built-in audit history and logging

Composable with external CI/CD workflows

Customizable build steps based on Docker images

Automated triggers for Github, BitBucket, and Cloud Source Repos

Concurrent Builds with Container Builder

```
steps:
- name: 'gcr.io/cloud-builders/go'
  args: ['generate']
- name: 'gcr.io/cloud-builders/go'
  args: ['test', './...']
- name: 'gcr.io/cloud-builders/go'
  args: ['install', 'mytarget']
  id: 'go-install'

- name: 'gcr.io/cloud-builders/gsutil'
  args: ['cp', '-r', 'gs://my-resource-bucket/somefiles', './somefiles']
  waitFor: ['-']  # The '-' indicates that this step begins immediately.
  id: 'fetch-resources'

- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/mytarget', '.']
  waitFor: ['go-install', 'fetch-resources']

images: ['gcr.io/$PROJECT_ID/mytarget']
```

Things to Remember

Things to Remember

Stateful Services

- Know the ops of your distributed systems really well
- Those ops might not match up to Kubernetes
- Don’t spend too much time fighting Kubernetes

Federation

- Great if you want the same thing everywhere
- Bad if you have a bunch of snowflake deployments

Security

- Authentication: figure out identity management
- Authorization: figure out access management
- Manage those secrets very closely with Cloud KMS, Kubernetes Secrets, or Vault

Getting Started

Minikube

Run single-node Kubernetes locally inside a VM on your laptop

Reuse your existing Docker installation with the minikube Docker daemon

Supports DNS, NodePorts, ConfigMaps, Secrets, Dashboards, Ingress

Addons can be added on :)

Low Hanging Fruit

Workloads with minimal dependencies

Skunkworks or Labs projects

Dev & test workloads

Questions?

Resources

Links

Getting Started with Minikube
Jenkins on Google Container Engine
Spinnaker on Google Compute Engine

Twitter

@crcsmnky