

Module - Overview

Overview

As data sets grow larger, in-memory processing on a single machine does not work well

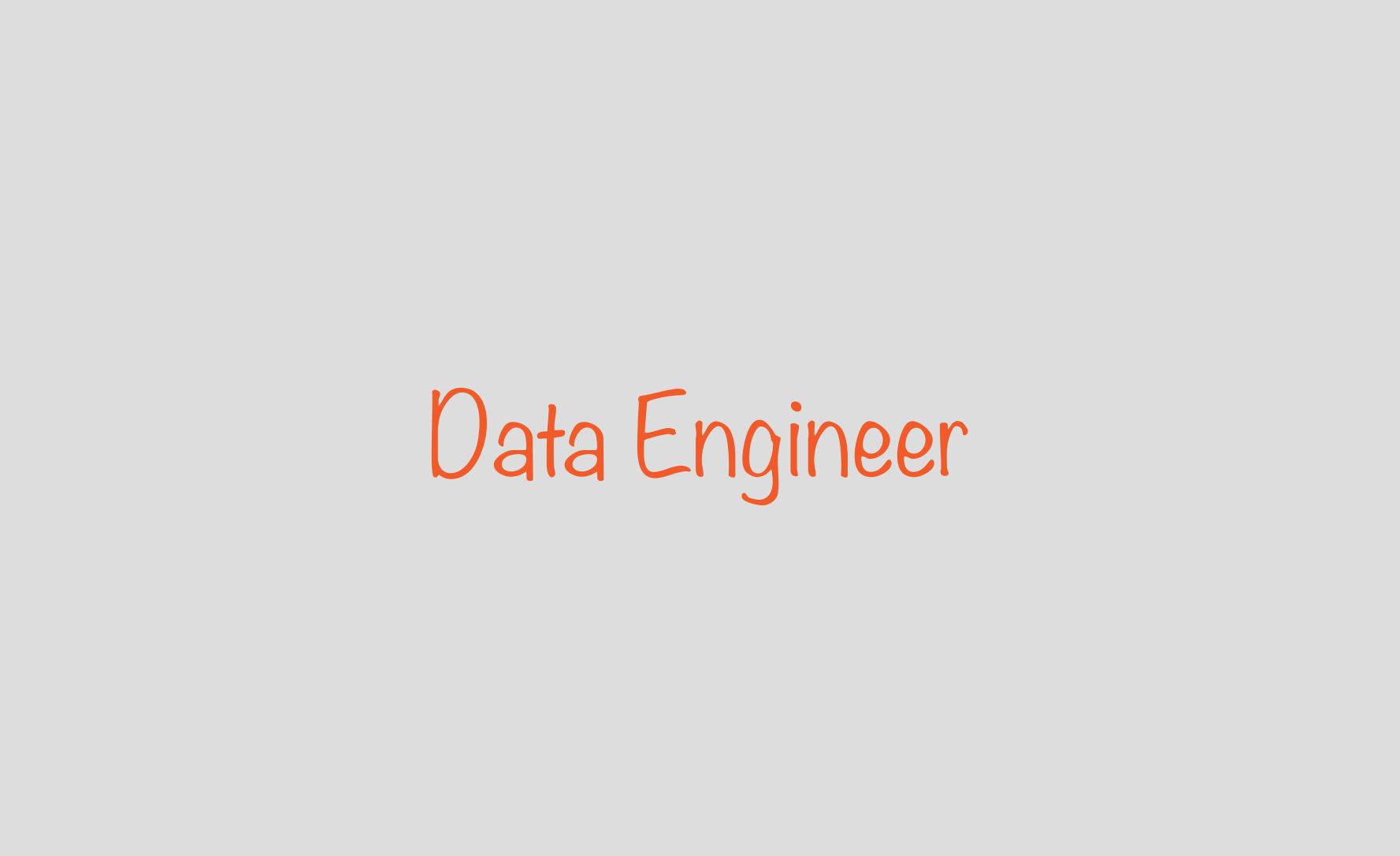
Distributed frameworks such as Hadoop/
MapReduce help

Configuring a cluster of distributed machines is complicated and expensive

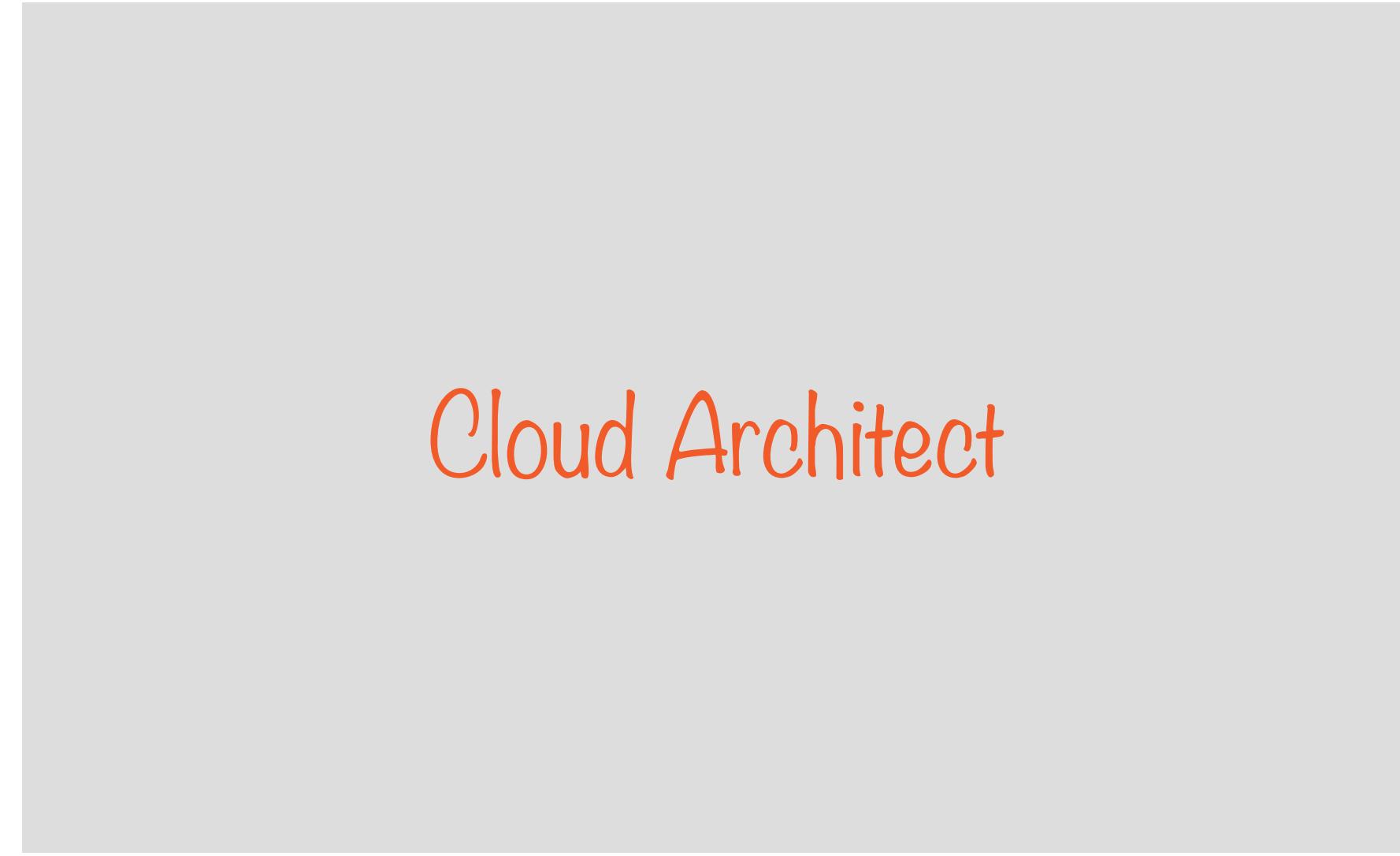
Cloud services such as GCP, AWS and Azure help

The Test

Two Tests



Data Engineer



Cloud Architect

Both - 2 hours, 50 questions, multiple-choice

Tests and Life



Data Engineer

Not easy to “game”

Big Data, ML, Hadoop..tough test

Cloud Architect

Definitely easier, more theoretical

Compute, Networking, Security,...

Data Engineer - Major Topics

(Also minor topics for Cloud Architect)

- Big Data

BigQuery, DataFlow, Pub/Sub

- Storage Technologies

Cloud storage, Cloud SQL, BigTable, Datastore

- Machine Learning

Concepts, TensorFlow, Cloud ML

Data Engineer - Minor Topics

(Also major topics
for Cloud Architect)

- Compute choices

AppEngine, Compute Engine, Containers

- Logging and monitoring

Stackdriver

- Security, networking

API keys, load balancing...

Required Context

- Hadoop, Spark, MapReduce...
- Hive, HBase, Pig
- RDBMS, indexing, hashing

Drills and Labs

- Syntax is tested too
- Implementation knowledge essential

Don't try to “prep for the test” (famous last words)

Why Cloud Computing

Why Cloud Computing

Data is getting bigger

World is getting smaller

Nothing fits in-memory on a single machine

Big Data, Small World



Super-computer

Cluster of generic
computers

Big Data, Small World



Monolithic

Distributed



Distributed

Lots of cheap hardware
Replication & fault tolerance
Distributed computing



Distributed

Lots of cheap hardware

HDFS

Replication & fault tolerance

YARN

Distributed computing

MapReduce



Distributed

“Clusters”

“Nodes”

“Server Farms”

“Server Farms”



Many Faces of Tim Duncan

[r/so-memes | tumblr](#)

All of these servers need to be
co-ordinated by a single piece
of software

Single Co-ordinating Software



Many Faces of Tim Duncan

nba-memes | tumblr

Partition data

Co-ordinate computing tasks

Handle fault tolerance and recovery

Allocate capacity to processes

Single Co-ordinating Software

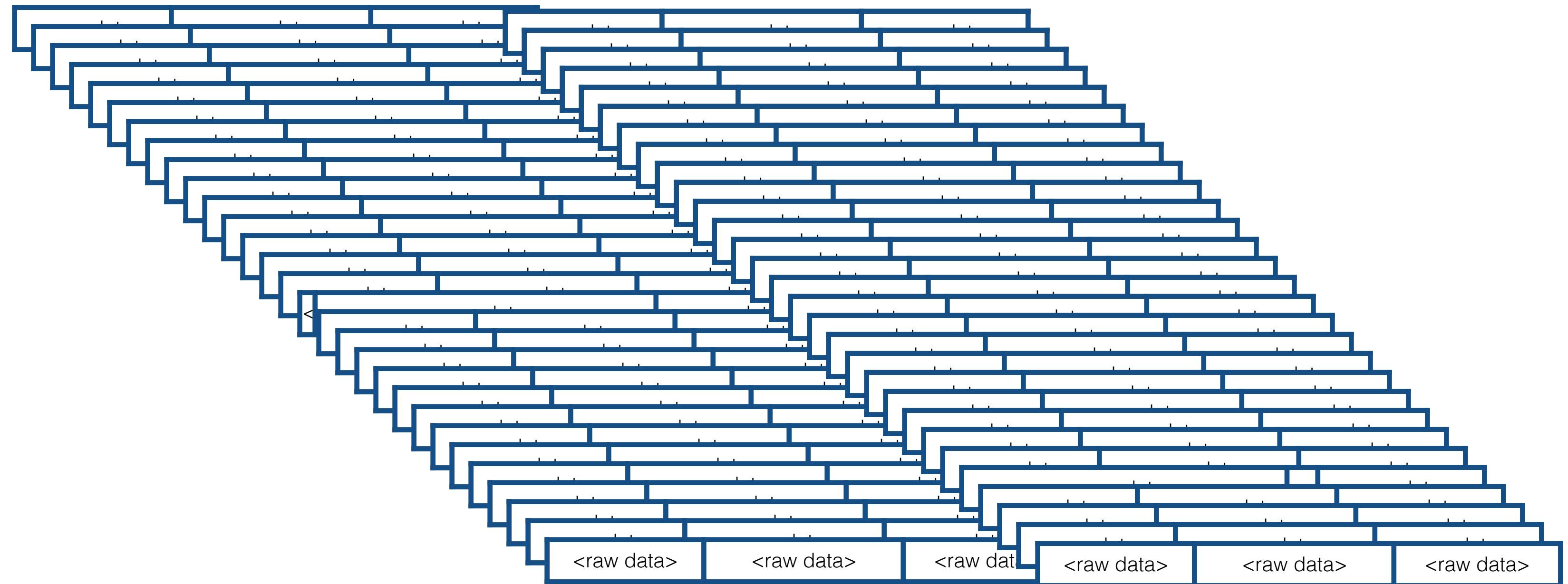


Many Faces of Tim Duncan

nba-memes | tumblr

Google developed proprietary
software to run on these
distributed systems

Single Co-ordinating Software



First: store millions of records on multiple machines

Single Co-ordinating Software



Many Faces of Tim Duncan

nba-memes | tumblr



Second: run processes on all these machines to crunch data

Single Co-ordinating Software

Google File System

To solve distributed
storage

MapReduce

To solve distributed
computing

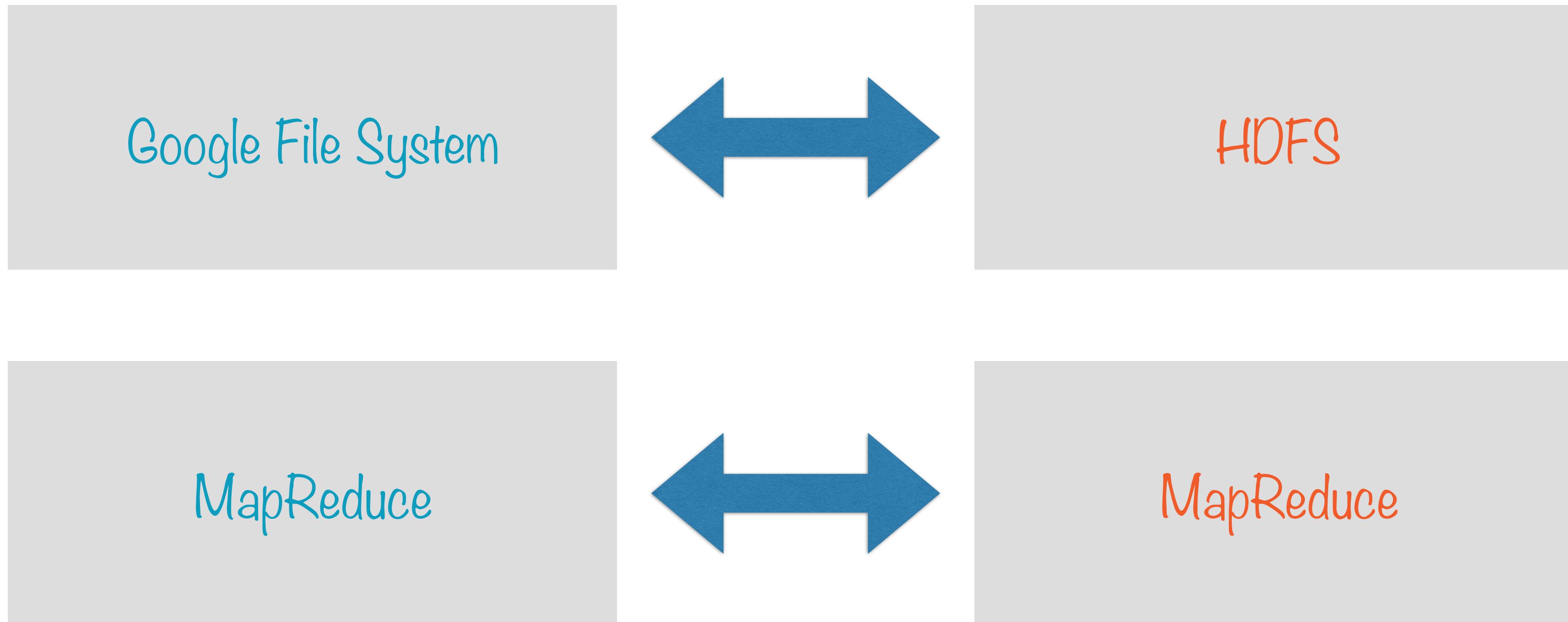
Single Co-ordinating Software

Google File System

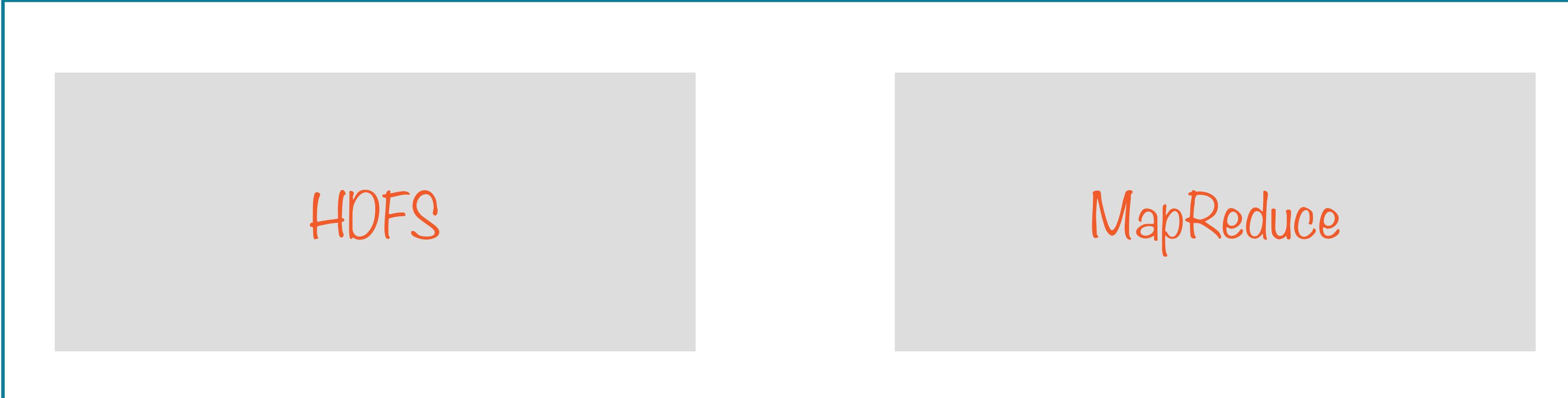
MapReduce

Apache developed open source
versions of these technologies

Single Co-ordinating Software



Hadoop



A diagram illustrating the Hadoop ecosystem. It features a large blue-bordered box containing two smaller gray boxes. The left gray box contains the text "HDFS" in orange. The right gray box contains the text "MapReduce" in orange.

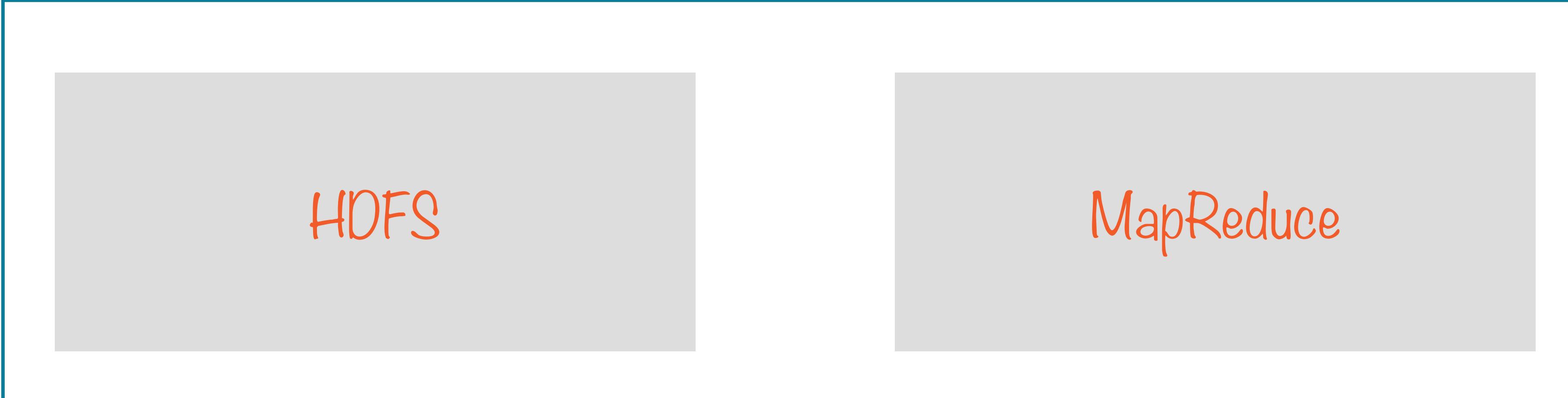
HDFS

MapReduce

A file system to manage
the storage of data

A framework to process
data across multiple servers

Hadoop



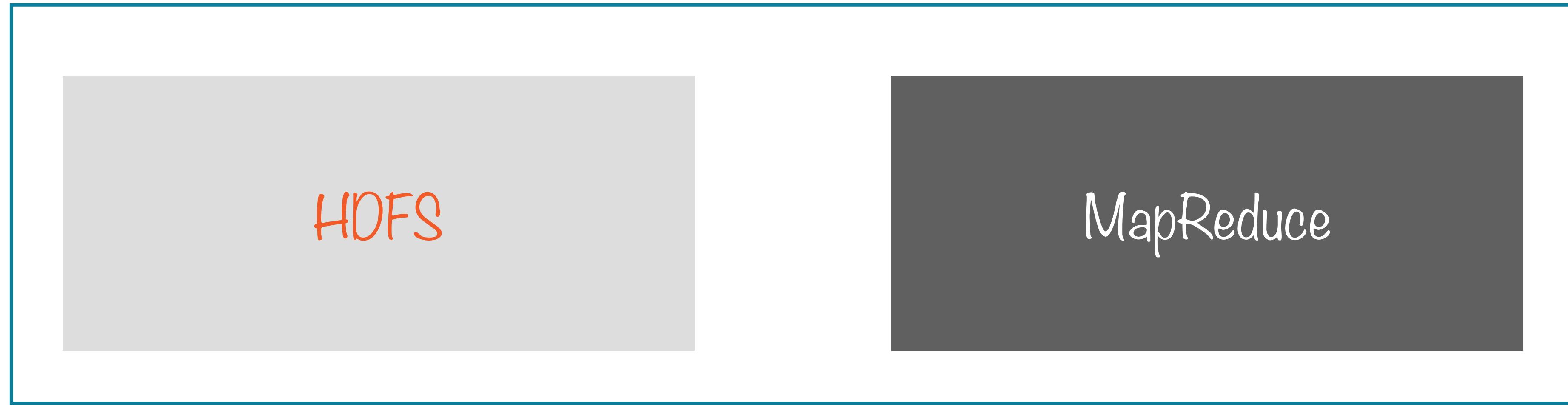
A diagram illustrating the Hadoop framework. It features a large blue-bordered rectangle representing the overall Hadoop system. Inside this rectangle, there are two gray rectangular boxes. The left box contains the text "HDFS" in orange. The right box contains the text "MapReduce" in orange.

HDFS

MapReduce

In 2013, Apache released Hadoop 2.0

Hadoop



MapReduce was broken into two separate
parts

Hadoop

HDFS

MapReduce

YARN

A framework to define
a data processing task

A framework to run
the data processing task

Hadoop

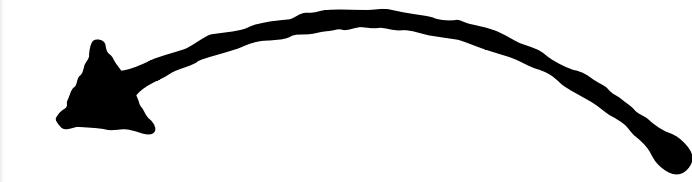
HDFS

MapReduce

YARN

Each of these components have corresponding configuration files

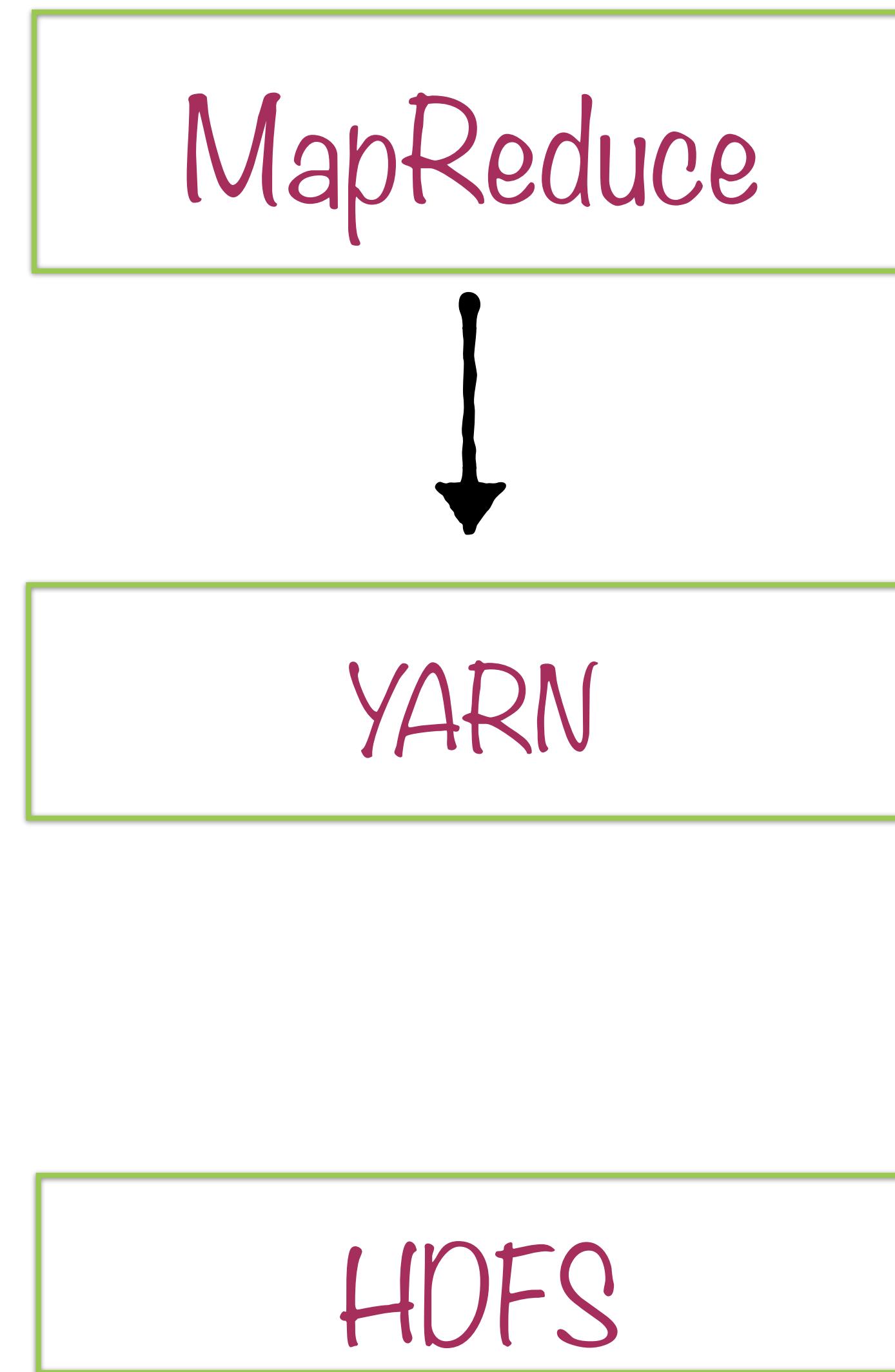
Co-ordination Between Hadoop Blocks



User defines map and
reduce tasks using the

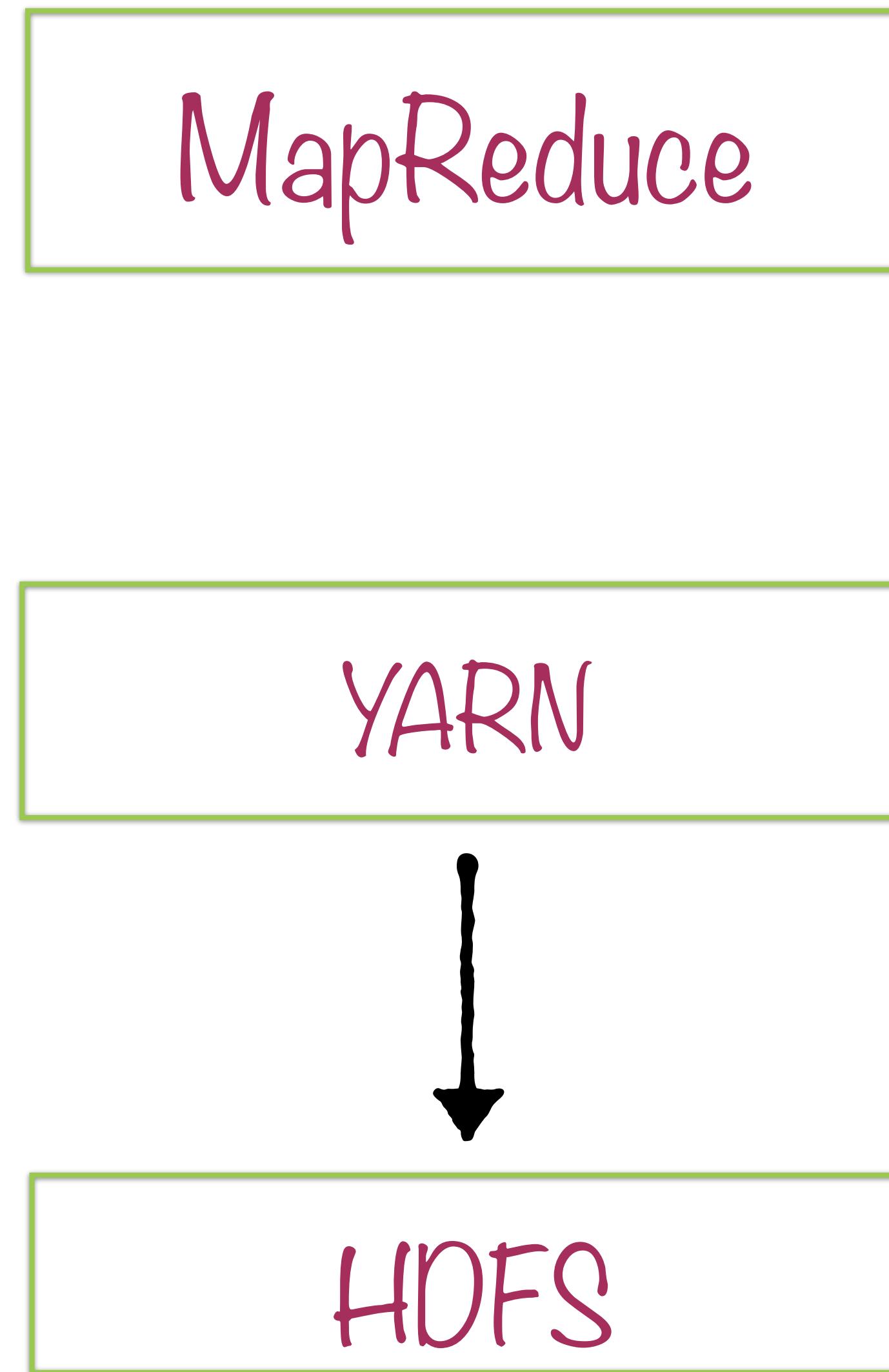
MapReduce API

Co-ordination Between Hadoop Blocks



A job is triggered on
the cluster

Co-ordination Between Hadoop Blocks



YARN figures out where and how to run the job, and stores the result in HDFS

Hadoop Ecosystem



Hadoop

An ecosystem of tools have sprung up around this
core piece of software

Hadoop Ecosystem

Hive

HBase

pig

Hadoop

Kafka

Spark

Oozie

Hadoop Ecosystem

Hive

HBase

pig

Kafka

Spark

Oozie

Hive

Provides an SQL interface to Hadoop

The bridge to Hadoop for folks who don't have
exposure to OOP in Java

HBase

A database management system on top of
Hadoop

Integrates with your application just like a
traditional database

Pig

A data manipulation language

Transforms unstructured data into a
structured format

Query this structured data using interfaces
like Hive

Spark

A distributed computing engine used along with Hadoop

Interactive shell to quickly process datasets

Has a bunch of built in libraries for machine learning, stream processing, graph processing etc.

Oozie

A tool to schedule workflows on all the
Hadoop ecosystem technologies

Kafka

Stream processing for unbounded datasets

Hadoop Ecosystem

Hive

HBase

pig

Hadoop

Kafka

Spark

Oozie

Hadoop Ecosystem

Hive

HBase

Pig

Setting up and running distributed software is
an expensive, complicated exercise

Kafka

Spark

Oozie

Distributed Computing Infrastructure

Who owns the machines?

Who deploys the software?

How does scaling happen?

Distributed Computing Infrastructure



On-premise



Colocation Services



Cloud Services

On-premise



Who owns the machines?

You do.

Who deploys the software?

You do.

How does scaling happen?

You **buy** machines and scale them in.

Colocation Services



Who owns the machines?

You (usually and mostly) do

Who deploys the software?

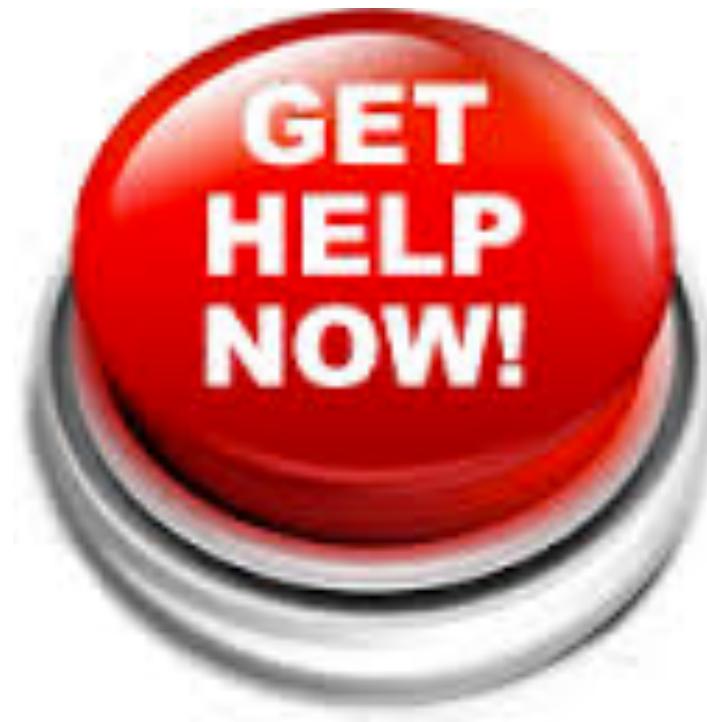
You do.

How does scaling happen?

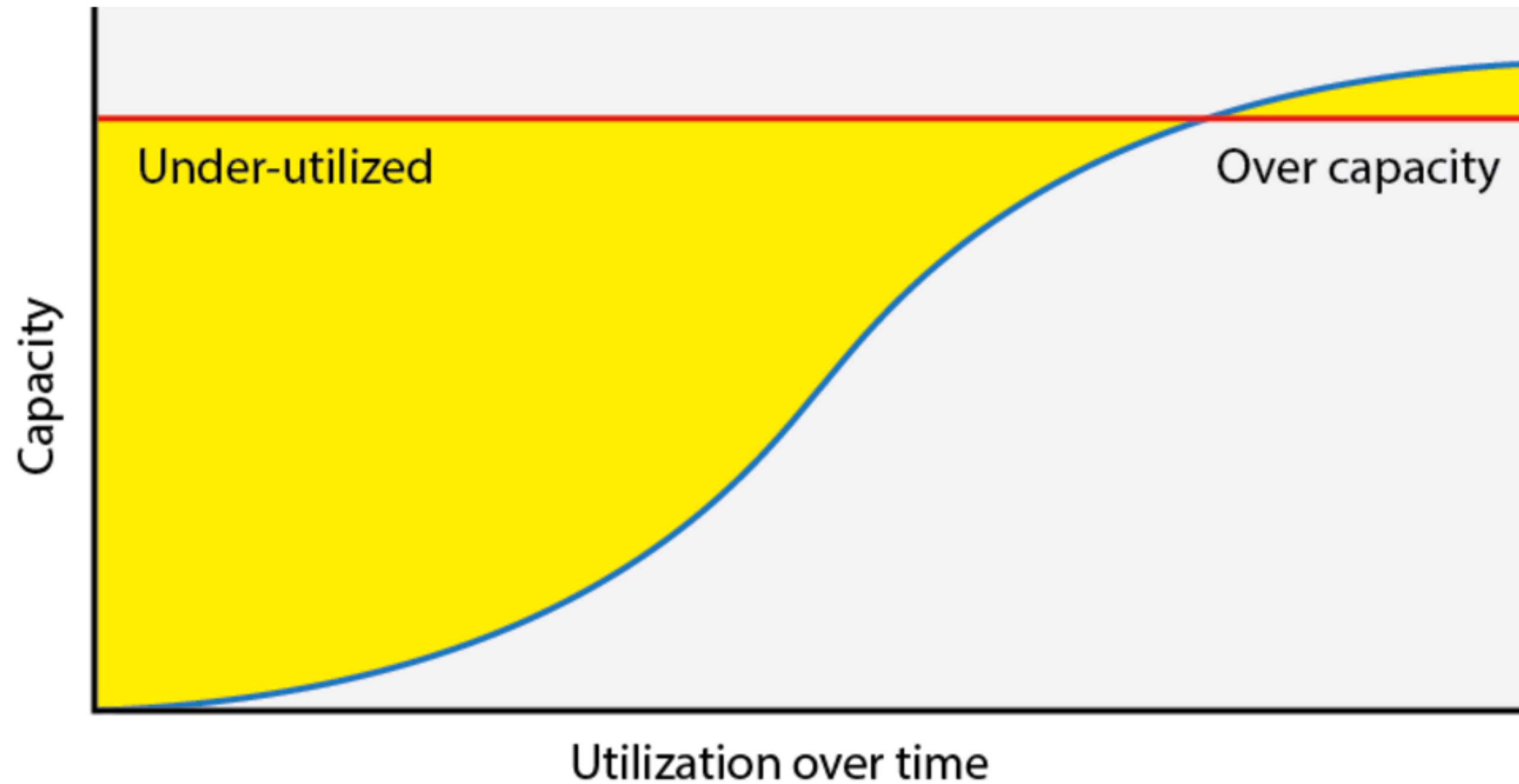
You **buy or lease** machines.

On-premise

Colocation Services



Utilisation planning is super-important



Else, end up with a white elephant data centre



Cloud Services

Who owns the machines?

Google, Amazon or Microsoft does

Who deploys the software?

GCP, AWS or Azure does it for you

How does scaling happen?

You **dynamically** add machines



Cloud Services: Financial Implications

OpEx rather than CapEx

Conserve cash

Watch for nickel-and-diming

Cloud Services: Operational Implications



Utilisation planning very simple indeed

Little chance of a white elephant data center

GCP Overview

Google Cloud Platform

“Use Resources”

Hardware (VMs, disks) and
Software (BigQuery, BigTable...)

“Pay for resources”

Billed for usage per-project

Google Cloud Platform

Resources

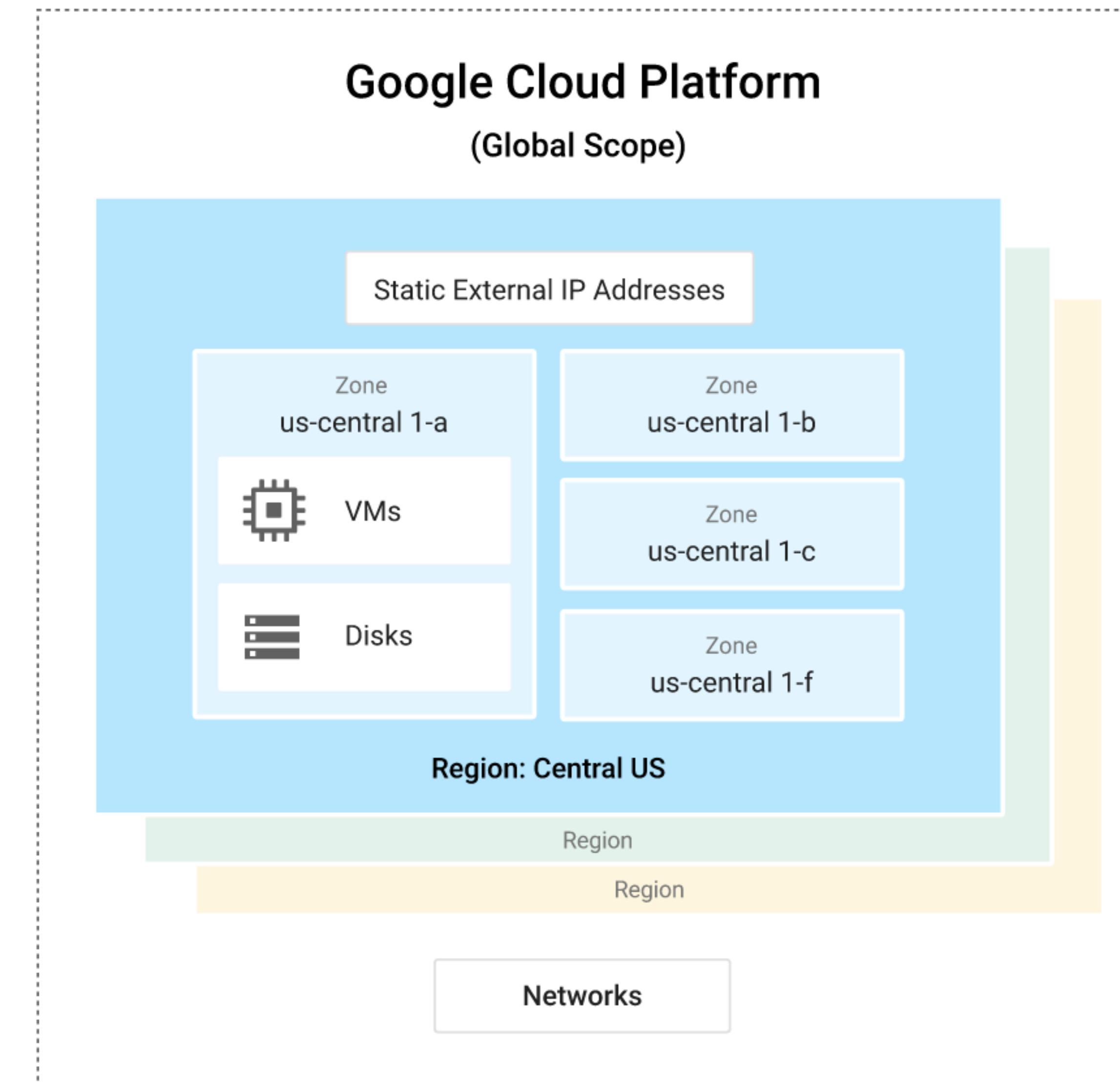
Billing

Google Cloud Platform

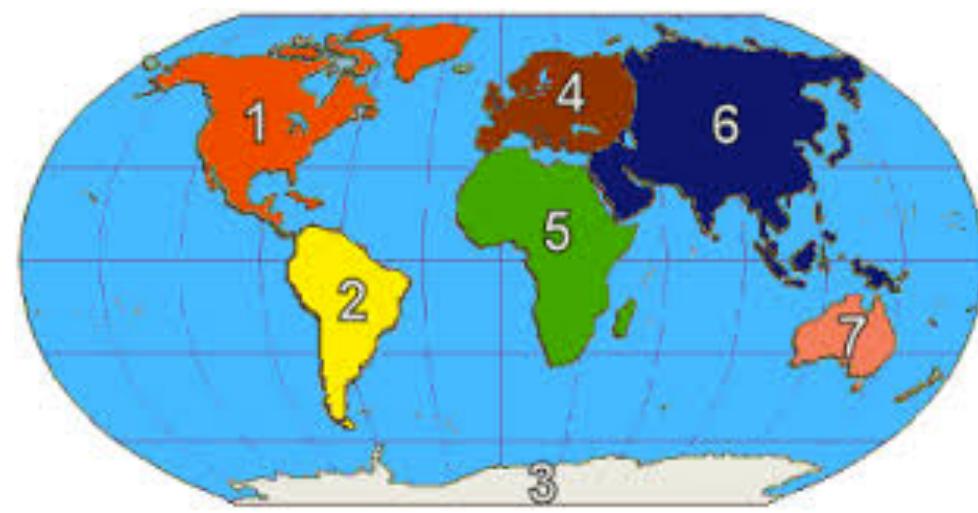
Resources

Billing

Resources

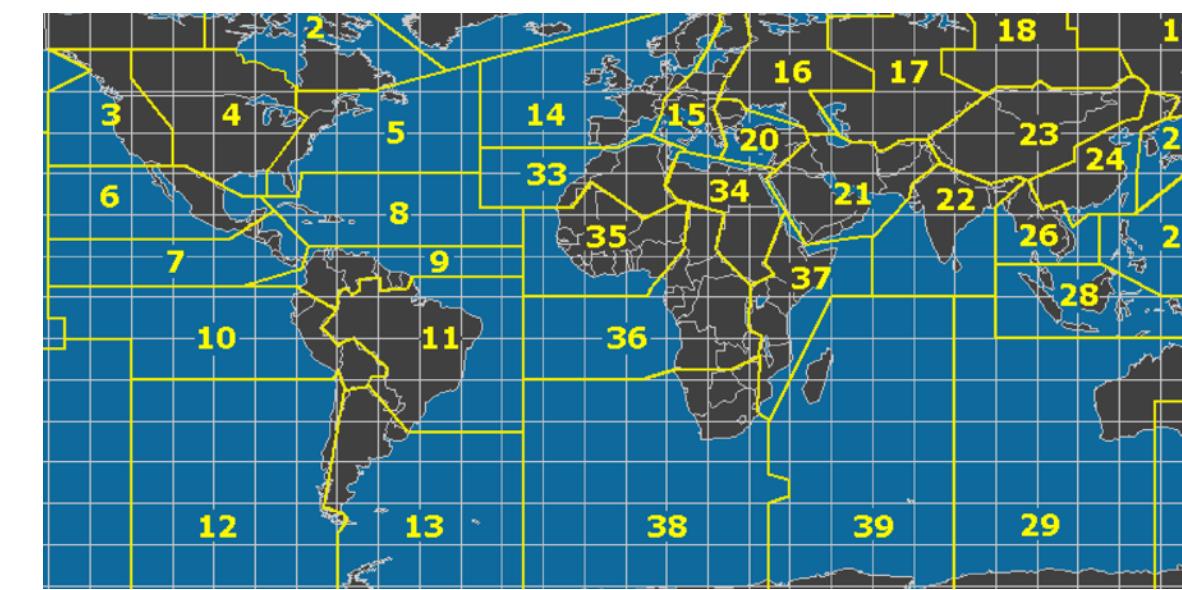


Location Hierarchy



“Regions”

e.g. Central US, Western Europe,
and East Asia



“Zones”

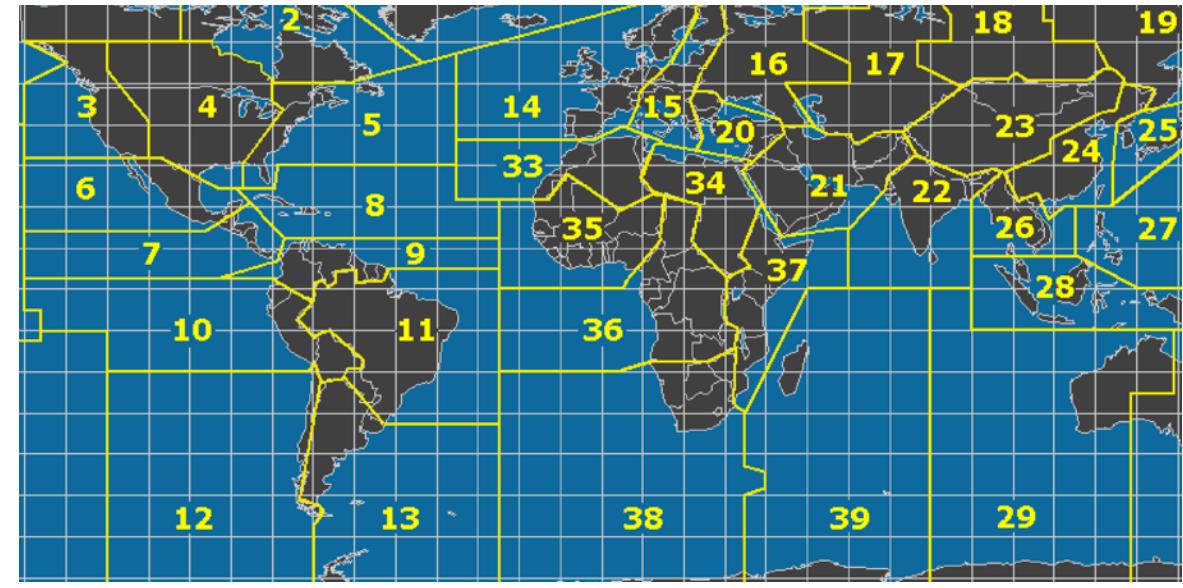
Basically, data centres within
regions



“Regions”

e.g. Central US, Western Europe, and East Asia

Within region, locations usually have network latencies of less than 5ms



“Zones”

Basically, data centres within regions

“Single failure domain” within a region
Identified as region-name + letter

asia-east1-a

Zones



Resources

Hardware

computers, hard disks

Software

Virtual machines (VMs)

Zones



Resources

Global aka multi-regional

Cloud Storage, DataStore, BigQuery

Regional

AppEngine instances

Zonal

VM (Compute Engine) instances, disks

Resources

Hardware

computers, hard disks

Software

Virtual machines (VMs),
Software services

Resources

Hardware

Compute choices,
Storage technologies...

Software

Big data,
machine learning ...

Major Topics

- Big Data

BigQuery, DataFlow, Pub/Sub

- Storage Technologies

Cloud storage, Cloud SQL, BigTable, Datastore

- Machine Learning

Concepts, TensorFlow, Cloud ML

Minor Topics

- Compute choices
AppEngine, Compute Engine, Containers
- Logging and monitoring
Stackdriver
- Security, networking
API keys, load balancing...

Resources

Hardware

Compute choices,
Storage technologies...

Software

Big data,
machine learning ...

Consumption Mechanisms

GCP Console

web front-end



Command-line
Interface

Client Libraries

gcloud utility (needs
SDK), or gcloud shell

Python, Java, Go...

Google Cloud Platform

Resources

Billing

Google Cloud Platform

Resources

Billing

Billing

All resources consumed associated with a project

Projects are associated with accounts

Billing happens per-project

Projects

Resources + Settings + Metadata

Resources within a project can easily interact

Project ~ Namespace

Project

Name, ID, number

ID is unique, forever

Project ~ Namespace

Getting Feet Wet

Summary

Cloud services have important advantages over on-premise or colocated data centers

GCP, Google's Cloud Platform, offers a suite of storage and compute solutions

TensorFlow and Cloud ML give GCP an edge in machine learning applications

The Google Cloud Data Engineer test is a fairly rigorous one

Module - Compute

Overview

GCP offers three Compute Options for running cloud apps

Google AppEngine is the PaaS option - serverless and ops-free

Google ComputeEngine is the IaaS option - fully controllable down to OS

Google Container Engine lies in between - clusters of machines running Kubernetes and hosting containers

Hosting Web Content: A Case Study of Compute Options

Cloud Use-Cases

Hosting a website

Running a Hadoop
cluster

Serving a
TensorFlow model

Relatively basic

“Big data”

“Machine
Learning”

Cloud Use-Cases

Hosting a website

Running a Hadoop
cluster

Serving a
TensorFlow model

Relatively basic

“Big data”

“Machine
Learning”

Hosting a Website

Static, No SSL

Plain HTML files

Just buy storage

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

SSL, CDN, Bells & whistles

Still quite static, but rich content

Need HTTPS serving, release management etc

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website



Buy disk space

Automatic scaling

No HTTPS, no CDN, no deployment help, nothing

“Google Cloud Storage”

Hosting on Cloud Storage

To host a static site, create a Cloud Storage Bucket and upload content

Can use “storage.googleapis.com” URL, or own domain

Hosting on Cloud Storage

Could write own HTML/CSS

Or use static generators

- Jekyll
- Ghost
- Hugo

Hosting on Cloud Storage

Can copy over content to bucket directly

- Web Console
- Cloud Shell

Or

- store on GitHub
- then use WebHook to run update script

Or

- use CI/CD tool like Jenkins
- Cloud Storage plug-in for post-build step

Hosting a Website



Buy disk space

Automatic scaling

No HTTPS, no CDN, no deployment help, nothing

“Google Cloud Storage”

Hosting a Website

“Google Cloud Storage”

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

SSL, CDN, Bells & whistles

Still quite static, but rich content

Need HTTPS serving, release management etc

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

“Google Cloud Storage”

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

SSL, CDN, Bells & whistles

Still quite static, but rich content

Need HTTPS serving, release management etc

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

SSL so that HTTPS serving is possible

CDN edges world-over

Atomic deployment, one-click rollback

SSL, CDN, Bells & whistles

Still quite static, but rich content

Need HTTPS serving, release management etc

“Firebase Hosting +
Google Cloud Storage”

Hosting a Website

“Google Cloud Storage”

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

“Firebase Hosting + Google Cloud Storage”

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

“Google Cloud Storage”

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

“Firebase Hosting +
Google Cloud Storage”

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

You'd like to control load balancing, scaling etc yourself

IAAS (“Infra-as-a-service”)

“Google Compute Engine”

Configuration, administration, management - all on you

No need to buy machines or install OS, dev stack, languages etc

Hosting with Compute Engine

Google Cloud Launcher

LAMP stack or WordPress in a few minutes

Cost estimates before deployment

Hosting with Compute Engine

You choose machine types, disk sizes before deployment

Can customise configuration, rename instances etc

After deployment, have full control of machine instances

Hosting with Compute Engine

Loads of storage options

- Cloud Storage Buckets
- Standard persistent disks
- SSD (solid state disks)
- Local SSD

After deployment, have full control of machine instances

Hosting with Compute Engine

Loads of **storage technologies** if you prefer

- Cloud SQL (MySQL, PostgreSQL)
- NoSQL
- GCP NoSQL tools - BigTable, Datastore

Hosting with Compute Engine

Load Balancing options

- Network load balancing: forwarding rules based on address, port, protocol
- HTTP load balancing: look into content, examine cookies, certain clients to one server...
- Internal: On private network not on internet
- Auto-scaled

Hosting with Compute Engine

DevOps - laundry list

Compute Engine Management with Puppet, Chef, Salt, and Ansible

Automated Image Builds with Jenkins, Packer, and Kubernetes

Distributed Load Testing with Kubernetes

Continuous Delivery with Travis CI

Managing Deployments with Spinnaker

**Hosting with
Compute Engine**

StackDriver for logging and monitoring

Hosting a Website

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

You'd like to control load balancing, scaling etc yourself

IaaS (“Infra-as-a-service”)

“Google Compute Engine”

Configuration, administration, management - all on you

No need to buy machines or install OS, dev stack, languages etc

Hosting a Website

“Google Cloud Storage”

“Google Compute Engine”

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

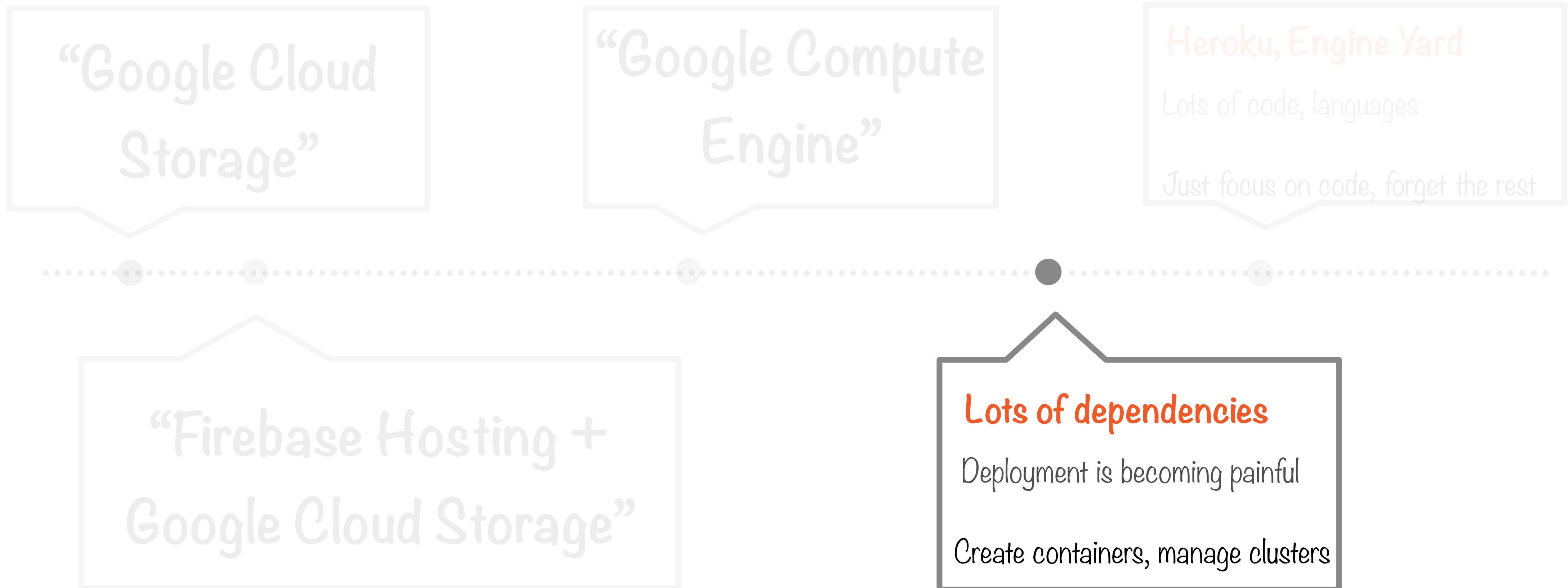
“Firebase Hosting +
Google Cloud Storage”

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website



Hosting a Website

You run separate web server, database

Separate containers to isolate from each other

Service-oriented architecture

Microservices

“Google Container Engine”



Lots of dependencies

Deployment is becoming painful

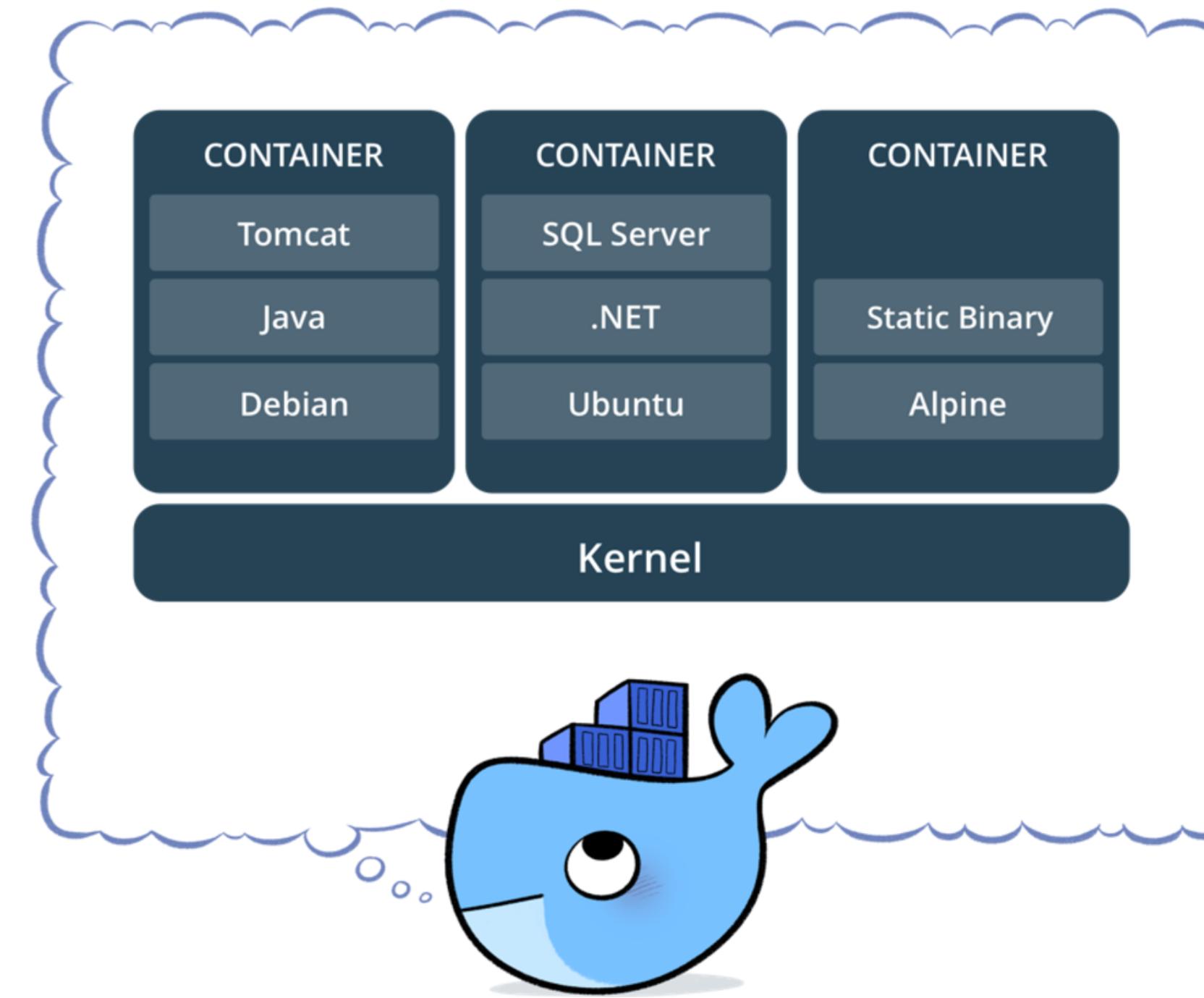
Create containers, manage clusters

Container

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

(www.docker.com)

Container

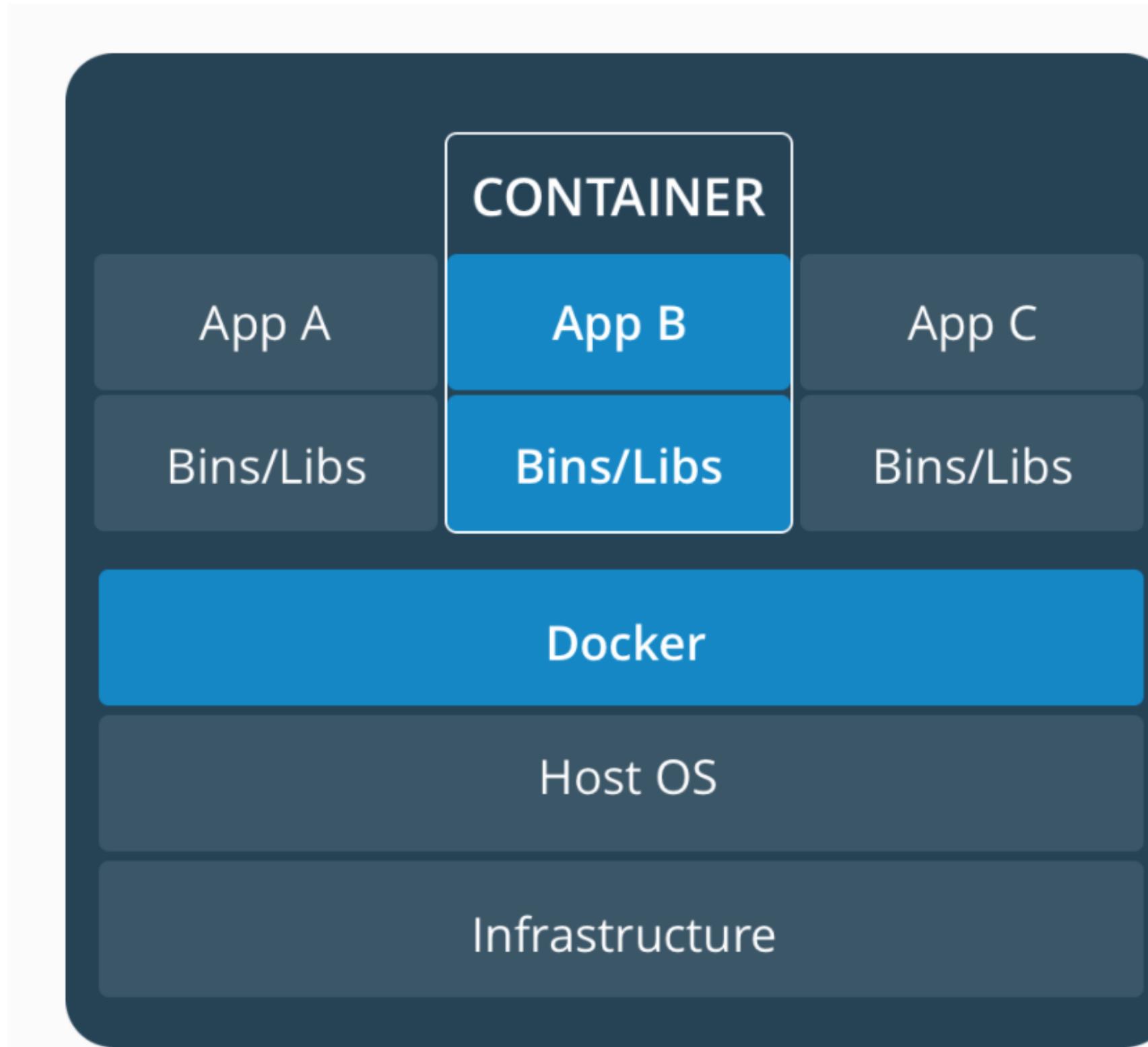


A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

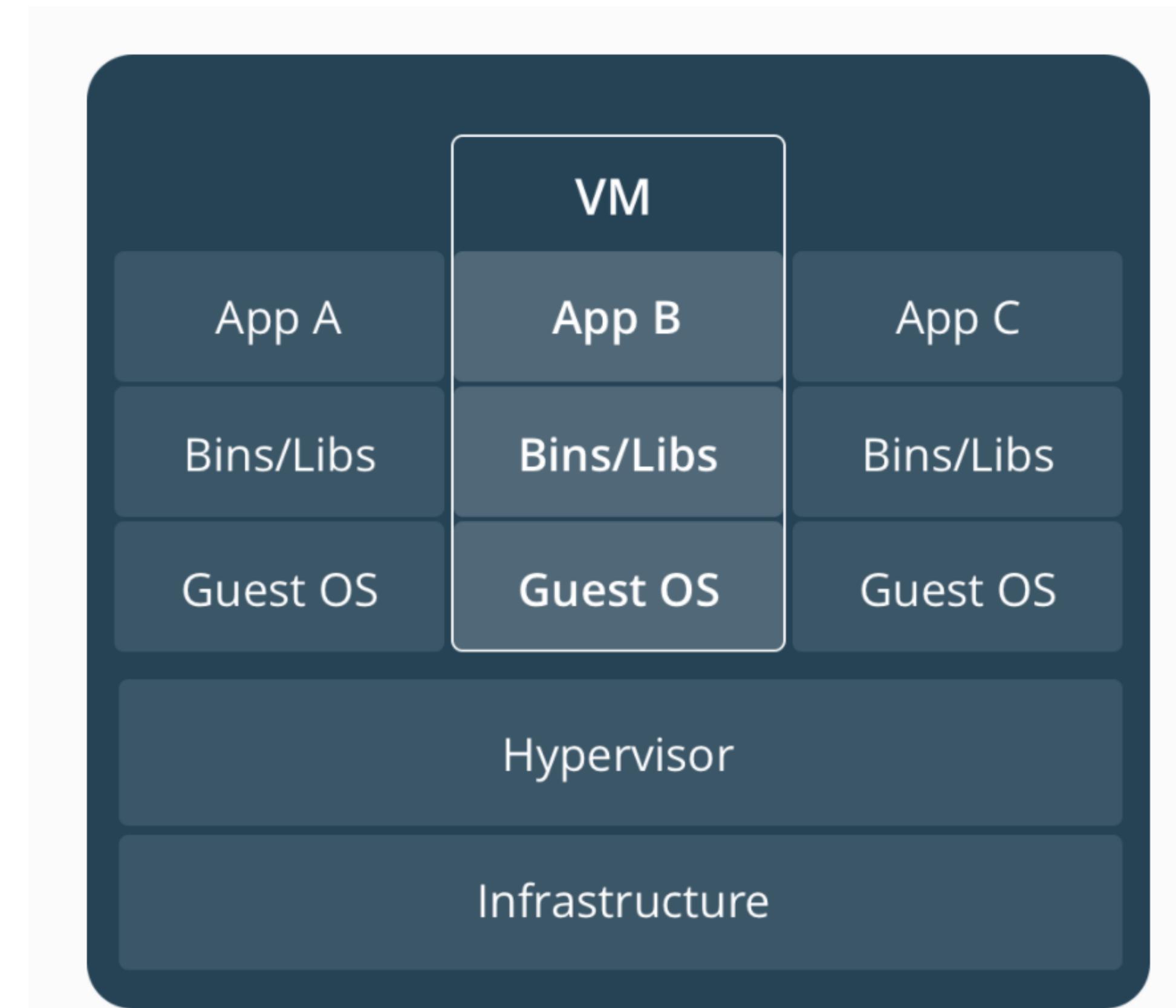
(www.docker.com)

Containers and VMs

Containers

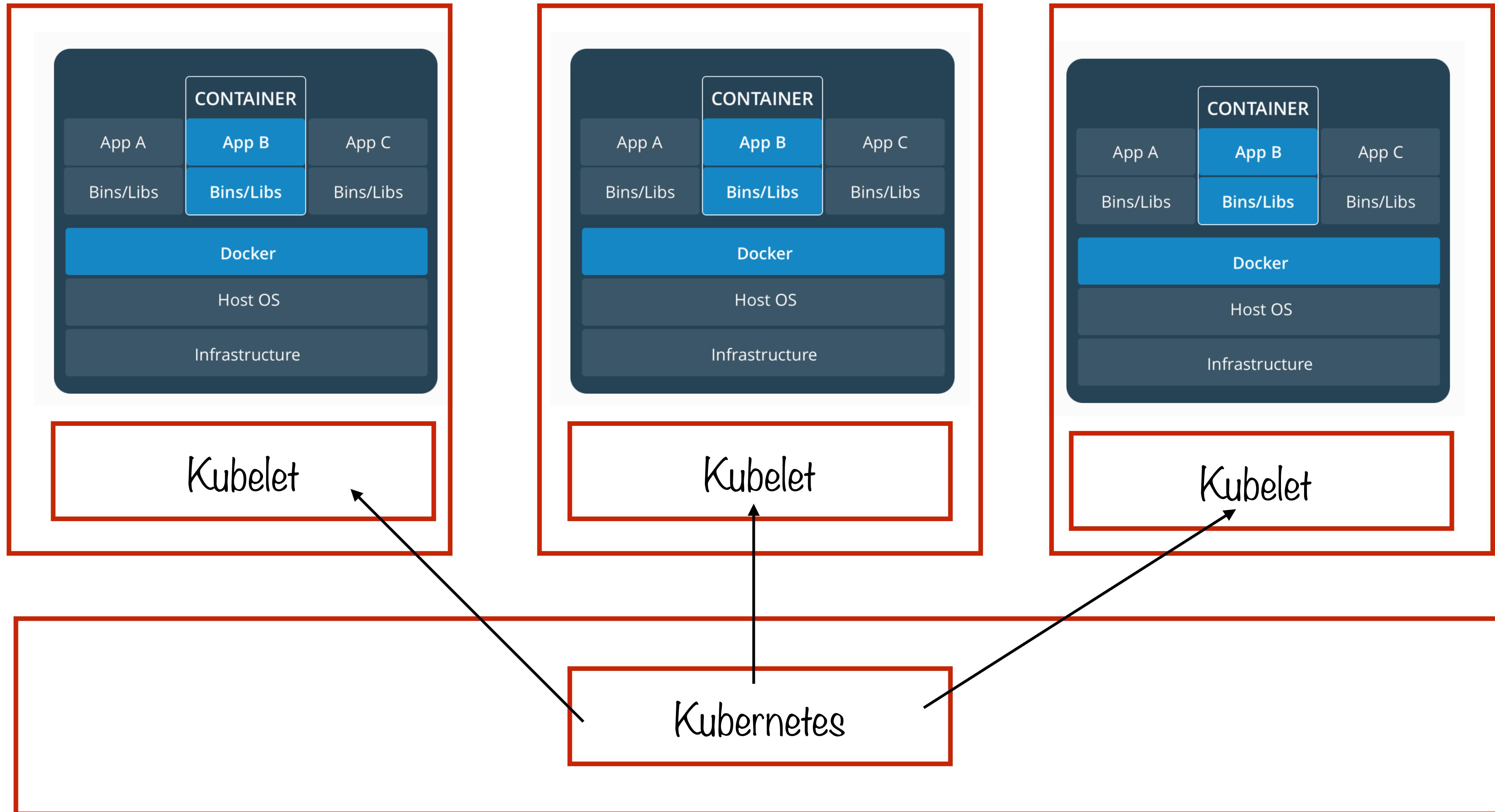


Virtual Machines

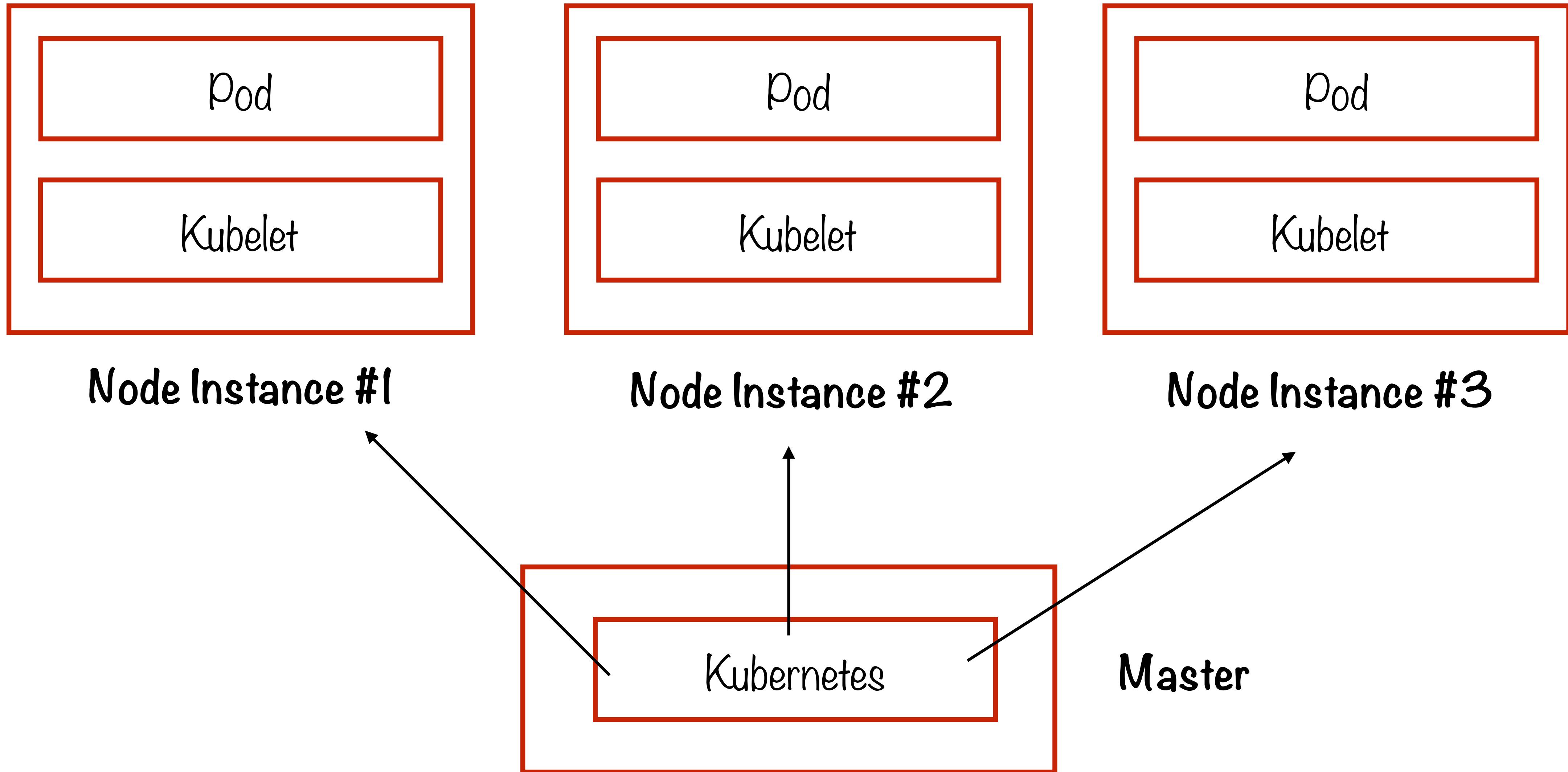


(www.docker.com)

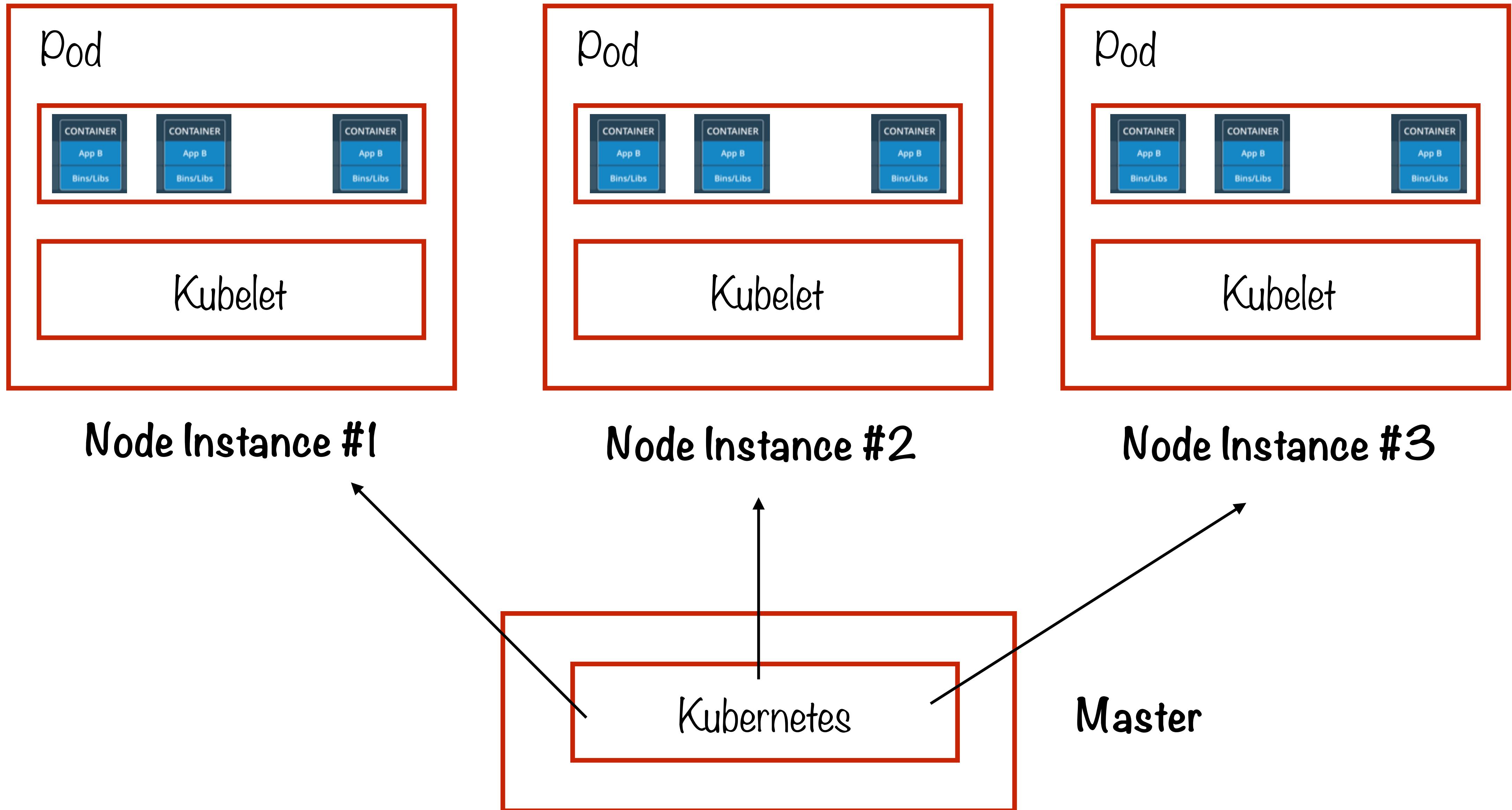
Container Cluster



Container Cluster



Container Cluster



Containers and VMs

Containers

Virtualise the Operating System

More portable

Quick to boot

Size - tens of MBs

Virtual Machines

Virtualise hardware

Less portable

Slow to boot

Size - tens of GBs

Hosting with Container Engine

DevOps - need largely mitigated
Can use Jenkins for CI/CD

**Hosting with
Container Engine**

StackDriver for logging and monitoring

Hosting a Website

You run separate web server, database

Separate containers to isolate from each other

Service-oriented architecture

Microservices

“Google Container Engine”



Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

“Google Cloud Storage”

“Google Compute Engine”

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

“Firebase Hosting +
Google Cloud Storage”

“Google Container
Engine”

Hosting a Website

Heroku, Engine Yard

Lots of code, languages

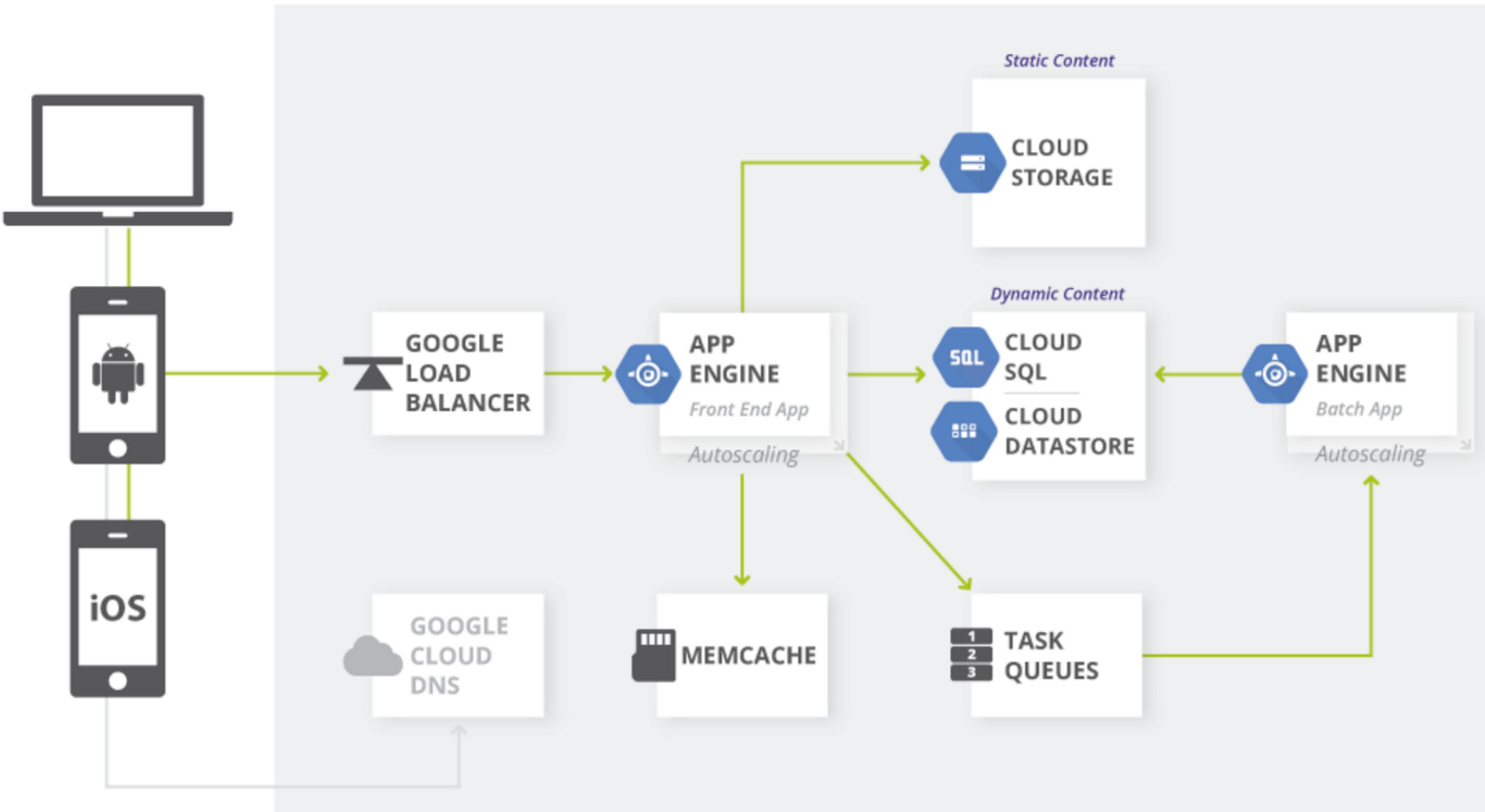
Just focus on code, forget the rest

Just write the code - leave the rest to platform

PaaS (“Platform-as-a-Service”)

“Google AppEngine”

Hosting on AppEngine



Hosting a Website

“Google Cloud Storage”

“Google Compute Engine”

“Google AppEngine”

“Firebase Hosting +
Google Cloud Storage”

“Google Container
Engine”

Compute Choices

App Engine

A flexible, zero ops
(serverless!) platform for
building highly available apps

Container Engine

Logical infrastructure powered by
Kubernetes, the open source
container orchestration system.

Compute Engine

Virtual machines running in
Google's global data center
network

App Engine

You want to focus on writing code, and never want to touch a server, cluster, or infrastructure.

Container Engine

You want to increase velocity and improve operability dramatically by separating the app from the OS.

Compute Engine

You need complete control over your infrastructure and direct access to high-performance hardware such as GPUs and local SSDs.

App Engine

You neither know nor care about the OS running your code

Container Engine

You don't have dependencies on a specific operating system.

Compute Engine

You need to make OS-level changes, such as providing your own network or graphic drivers, to squeeze out the last drop of performance.

App Engine

Support for Java, Python, PHP, Go, Ruby (beta) and Node.js (beta) ... or bring your own app runtime.

Container Engine

Run the same application on your laptop, on premise and in the cloud.

Compute Engine

Direct access to GPUs that you can use to accelerate specific workloads.

App Engine

Web sites; Mobile app and
gaming backends

RESTful APIs

Internet of things (IoT) apps.

Container Engine

Containerized workloads

Cloud-native distributed systems.

Hybrid applications.

Compute Engine

Any workload requiring a specific
OS or OS configuration

Currently deployed, on-premises
software that you want to run in
the cloud.

Can't be containerised easily; or
need existing VM images

Mix-and-Match

- Use App Engine for the front end serving layer, while running Redis in Compute Engine.
- Use Container Engine for a rendering microservice that uses Compute Engine VMs running Windows to do the actual frame rendering.
- Use App Engine for your web front end, Cloud SQL as your database, and Container Engine for your big data processing.

App Engine

Environments

Standard

Pre-configured with: Java 7,
Python 2.7, Go, PHP

Flexible

More choices: Java 8, Python
3.x, .NET

- Serverless!
- Instance classes determine price, billing
- Laundry list of services - pay for what you use

Environments

Standard

Pre-configured with: Java 7,
Python 2.7, Go, PHP

Flexible

More choices: Java 8, Python
3.x, .NET

- Serverless!
- Instance classes determine price, billing
- Laundry list of services - pay for what you use

AppEngine Standard Environment

- Based on container instances running on Google's infrastructure
- Preconfigured with one of several available runtimes (Java 7, Python 2.7, Go and PHP)
- Each runtime also includes libraries that support App Engine Standard APIs
- Maybe all you need

AppEngine Standard Environment

- Applications run in a secure, sandboxed environment
- App Engine standard environment distributes requests across multiple servers, and scaling servers to meet traffic demands
- Your application runs within its own secure, reliable environment that is independent of the hardware, operating system, or physical location of the server.

Environments

Standard

Pre-configured with: Java 7,
Python 2.7, Go, PHP

Flexible

More choices: Java 8, Python
3.x, .NET

- Serverless!
- Instance classes determine price, billing
- Laundry list of services - pay for what you use

Environments

Standard

Pre-configured with: Java 7,
Python 2.7, Go, PHP

Flexible

More choices: Java 8, Python
3.x, .NET

- Serverless!
- Instance classes determine price, billing
- Laundry list of services - pay for what you use

AppEngine Flexible Environment

- Allows you to customize your runtime and even the operating system of your virtual machine using Dockerfiles
- Under the hood, merely instances of Google Compute Engine VMs

Environments

Standard

Pre-configured with: Java 7,
Python 2.7, Go, PHP

Flexible

More choices: Java 8, Python
3.x, .NET

- Serverless!
- Instance classes determine price, billing
- Laundry list of services - pay for what you use

Cloud Functions

- Serverless execution environment for building and connecting cloud services
- Write simple, single-purpose functions
- Attached to events emitted from your cloud infrastructure and services
- Cloud Function is triggered when an event being watched is fired

Cloud Functions

- Your code executes in a fully managed environment
- No need to provision any infrastructure or worry about managing any servers
- Cloud Functions are written in Javascript
- Run it in any standard Node.js runtime

Compute Engine

Hosting a Website

Load balancing, scaling

Currently on VMs or servers

Get VMs, manage yourself

You'd like to control load balancing, scaling etc yourself

IAAS (“Infra-as-a-service”)

“Google Compute Engine”

Configuration, administration, management - all on you

No need to buy machines or install OS, dev stack, languages etc

Image Types

- Public images for Linux and Windows Server that Google provides
- Private images that you create or import to Compute Engine
- Images of other OSes OK too

Creation

- Creator has full root privileges, SSH capability
 - Can share with other users
- While creating instance specify
 - zone
 - OS
 - machine type

Projects and Instances

- Each instance belongs to a project
- Projects can have any number of instances
- Projects can have upto 5 VPC (Virtual Private Networks)
- Each instance belongs in one VPC
 - instances within VPC communicate on LAN
 - instances across VPC communicate on internet

Machine Types

- Standard
- High-memory
- High-CPU
- Shared-core (small, non-resource intensive)
- Can attach GPU dies to most machine types

Preemptible Instances

- Much much cheaper than regular Compute Engine instances
- But, might be terminated (preempted) at any time if Compute Engine needs the resources
- Use for fault-tolerant applications

Preemptible Instances

- Will definitely be terminated after running for 24 hours
- Probability of termination varies by day/zone etc
- Cannot live migrate (stay alive during updates) or auto-restart on maintenance

Preemptible Instances

- Step 1 in Preemption: Compute Engine sends a Soft Off signal
- Step 2: Hopefully, you have a shutdown script to clean up and give up control within 30 seconds
- Step 3: If not, Compute Engine sends a Mechanical Off signal
- Compute Engine transitions to Terminated state

Storage Options

- Each instance comes with a small root persistent disk containing the OS
- Add additional storage options
 - Persistent disks
 - Standard
 - SSD
 - Local SSDs
 - Cloud Storage

Storage Options

	Standard persistent disks	SSD persistent disks	Local SSDs	Cloud Storage buckets
Storage type	Efficient and reliable block storage	Fast and reliable block storage	High-performance local block storage	Affordable object storage
Price per GB/month	\$0.040 - \$0.052	\$0.170 - \$0.221	\$0.218 - \$0.283	\$0.007 - \$0.026
Maximum space per instance	64 TB	64 TB	3 TB	Almost infinite
Scope of access	Zone	Zone	Instance	Global
Data redundancy	Yes	Yes	No	Yes
Encryption at rest	Yes	Yes	Yes	Yes
Custom encryption keys	Yes	Yes	No	Yes
Machine type support	All machine types	All machine types	Most machine types	All machine types

Persistent Disks

- Durable network storage devices that instances can access like physical disks in a desktop or a server
- Compute Engine manages physical disks and data distribution to ensure redundancy and optimize performance
- Encrypted (custom encryption possible)
- Built-in redundancy
- Restricted to the zone where instance is located

Persistent Disks

- Two types - Standard and SSD
- Standard Persistent - regular hard disks - cheap - OK for sequential access
- SSD Persistent - expensive - fast for random access

Local SSD

- Physically attached to the server that hosts your virtual machine instance
- Local SSDs have higher throughput and lower latency
- The data that you store on a local SSD persists only until you stop or delete the instance
- Small - each local SSD is 375 GB in size, but you can attach up to eight local SSD devices for 3 TB of total local SSD storage space per instance.

Local SSD

- Very high IOPS and low latency
- Unlike persistent disks, you must manage the striping on local SSDs yourself
- Encrypted, custom encryption not possible

Cloud Storage Buckets

- use when latency and throughput are not a priority
- and
- when you must share data easily between multiple instances or zones.

Cloud Storage Buckets

- Flexible, scalable, durable
- ~Infinite size possible
- Performance depends on storage class
 - Multi-regional
 - Regional
 - Nearline
 - Coldline

Containers

Hosting a Website

“Google Cloud Storage”

“Google Compute Engine”

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

“Firebase Hosting + Google Cloud Storage”

Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

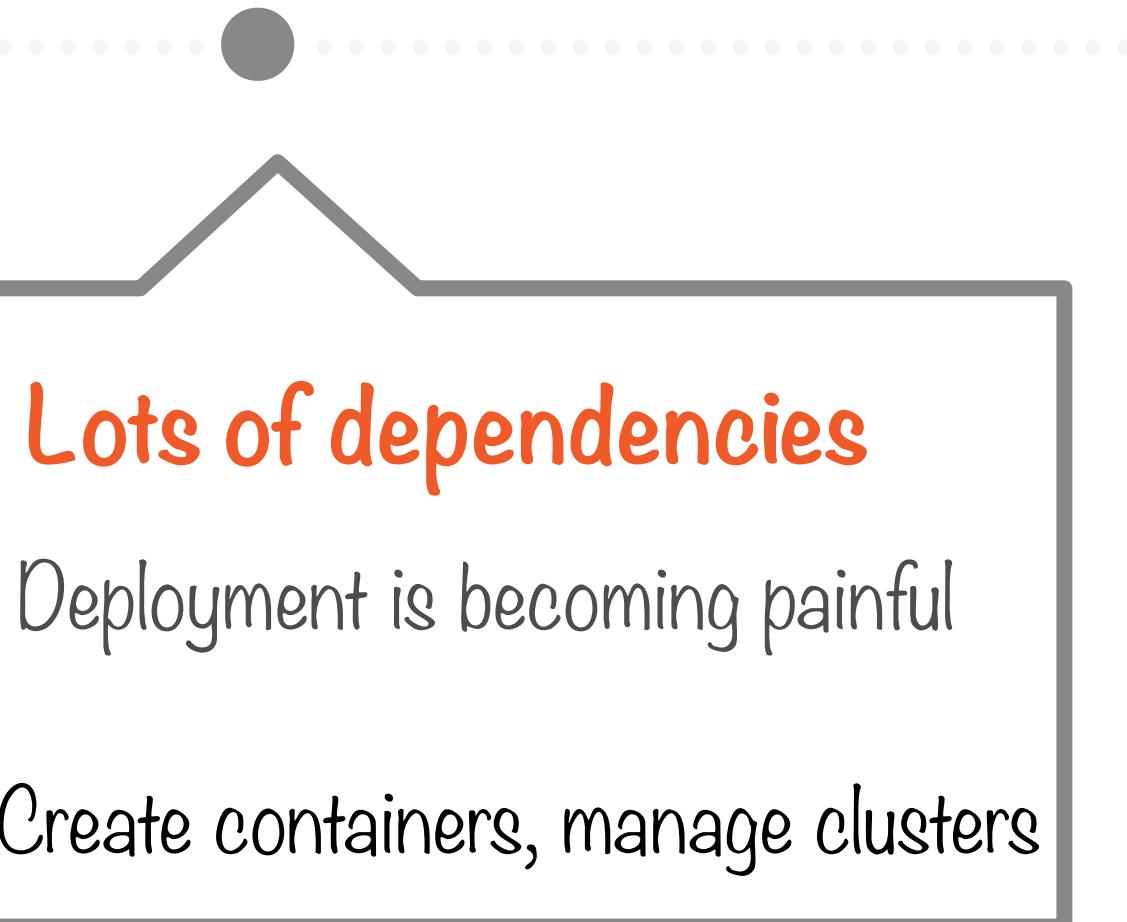
You run separate web server, database

Separate containers to isolate from each other

Service-oriented architecture

Microservices

“Google Container Engine”

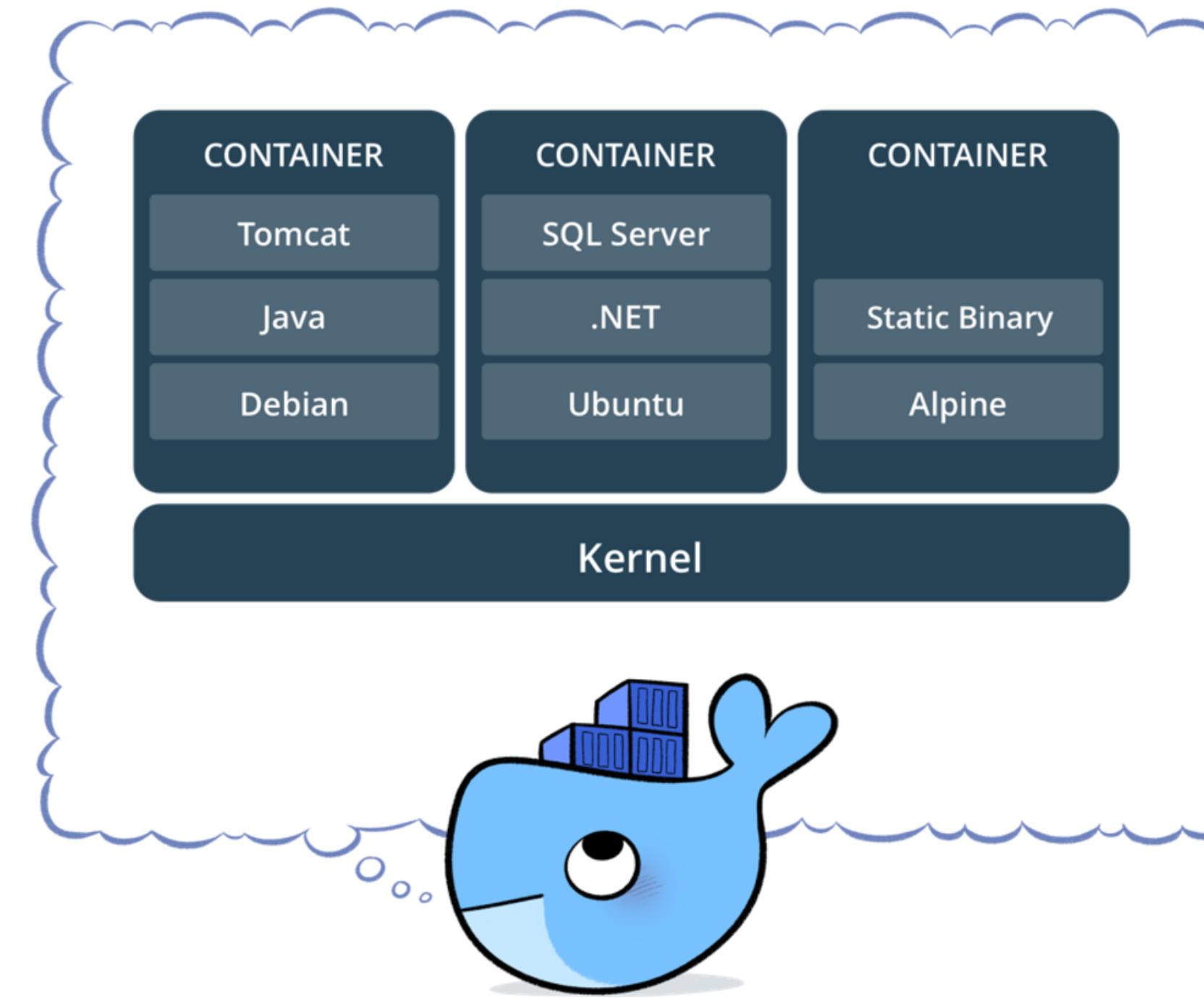


Container

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

(www.docker.com)

Container

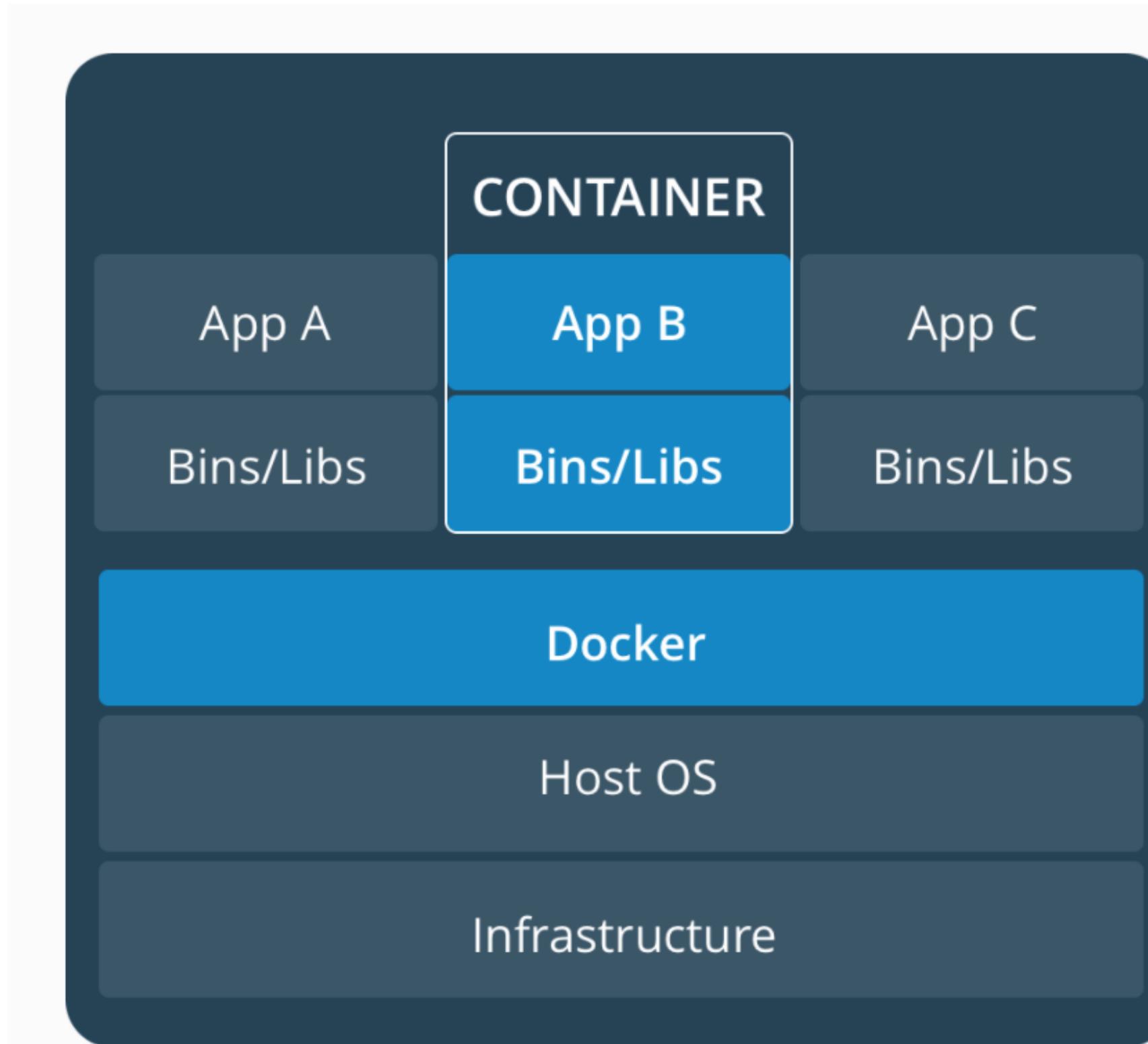


A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

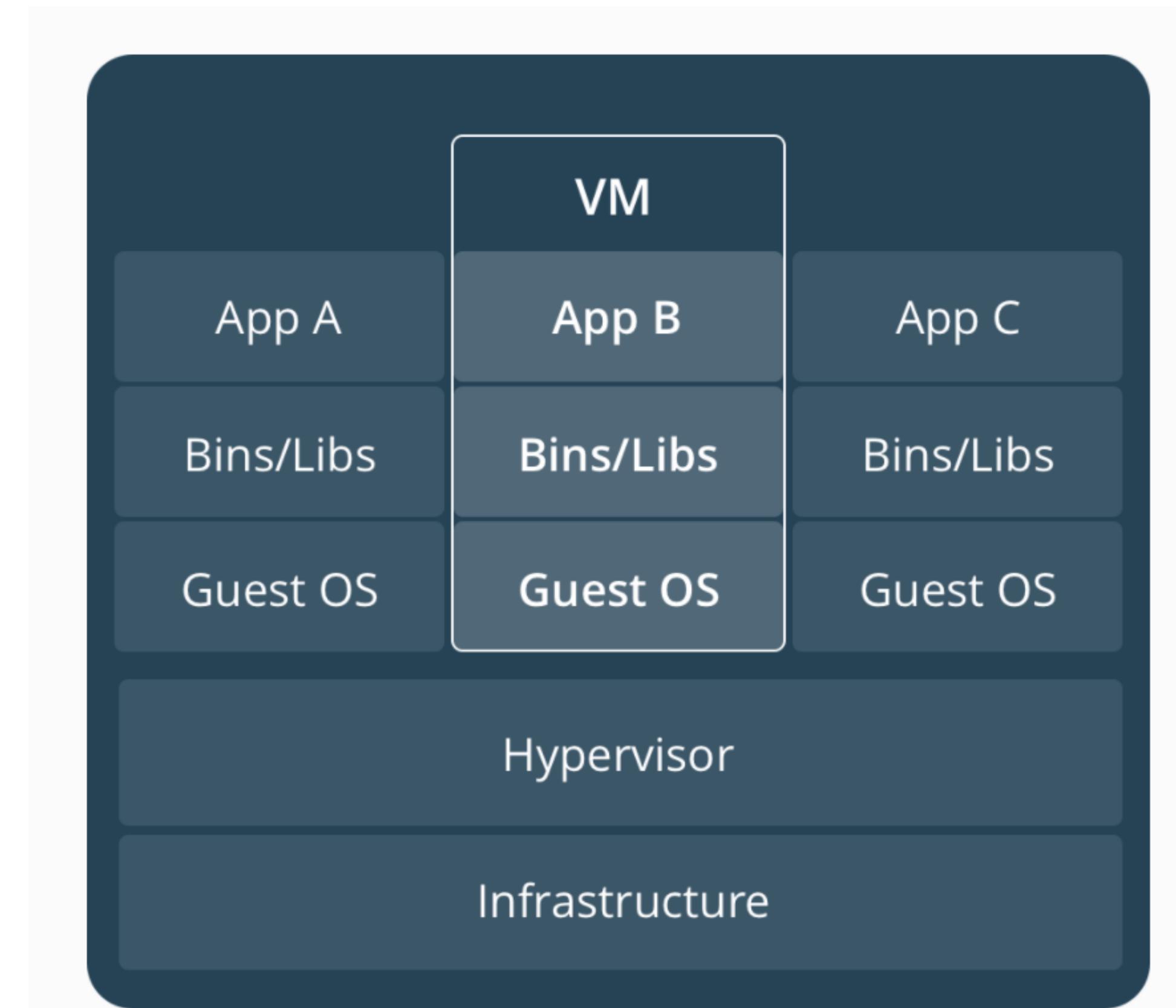
(www.docker.com)

Containers and VMs

Containers



Virtual Machines



(www.docker.com)

Containers and VMs

Containers

Virtualise the Operating System

More portable

Quick to boot

Size - tens of MBs

Virtual Machines

Virtualise hardware

Less portable

Slow to boot

Size - tens of GBs

Hosting a Website

You run separate web server, database

Separate containers to isolate from each other

Service-oriented architecture

Microservices

“Google Container Engine”



Lots of dependencies

Deployment is becoming painful

Create containers, manage clusters

Hosting a Website

“Google Cloud Storage”

“Google Compute Engine”

Heroku, Engine Yard

Lots of code, languages

Just focus on code, forget the rest

“Firebase Hosting +
Google Cloud Storage”

“Google Container
Engine”

Advantages

Componentization - microservices

Portability

Rapid deployment

Advantages

Orchestration - Kubernetes clusters

Image registration - Pull images from container registry

Flexibility - mix-and-match with other cloud providers, on-premise

Storage options

Storage options as with Compute Engine

However, remember that container disks are ephemeral

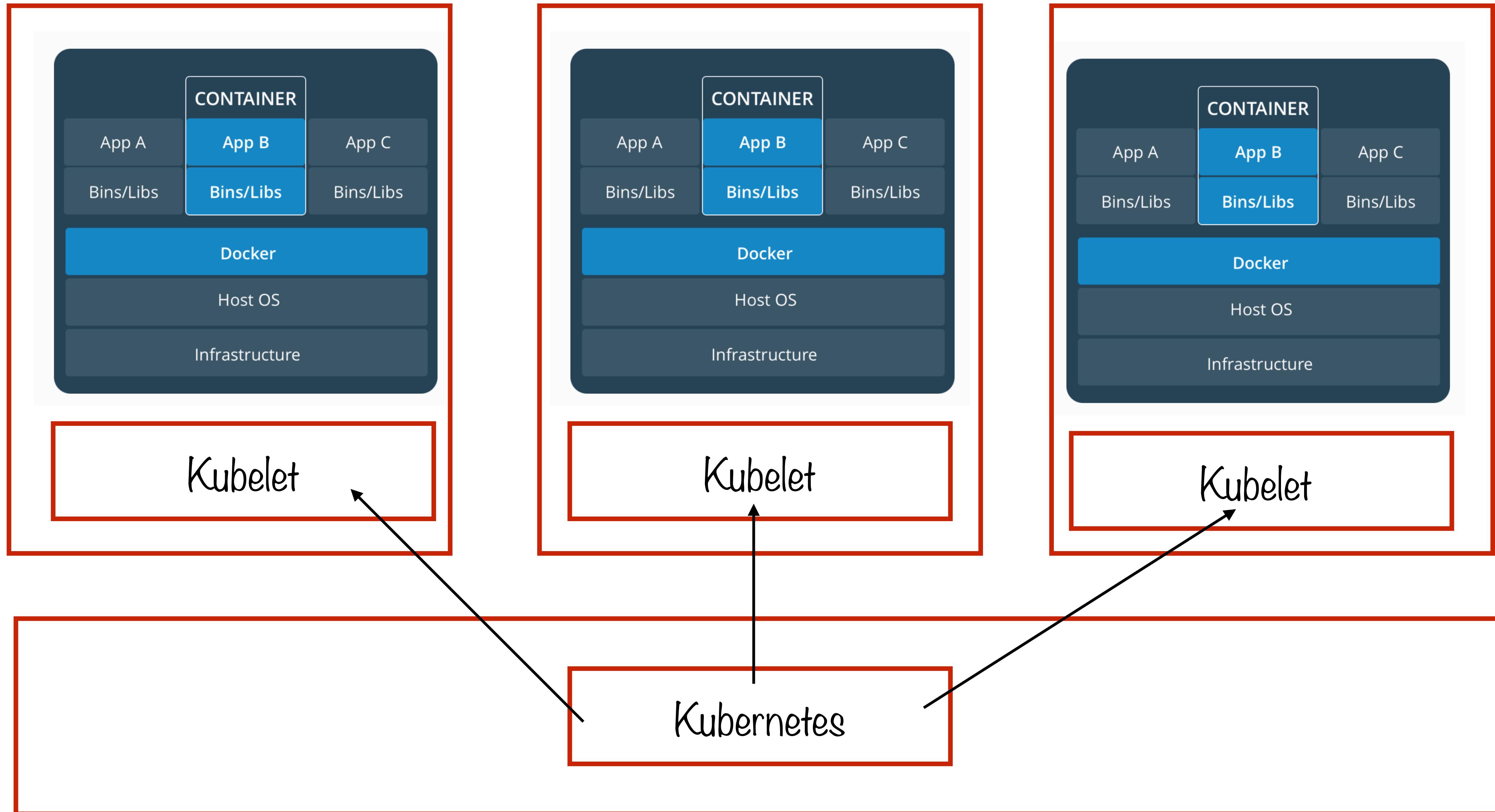
Need to use `gcePersistentDisk` abstraction for persistent disk

Load Balancing

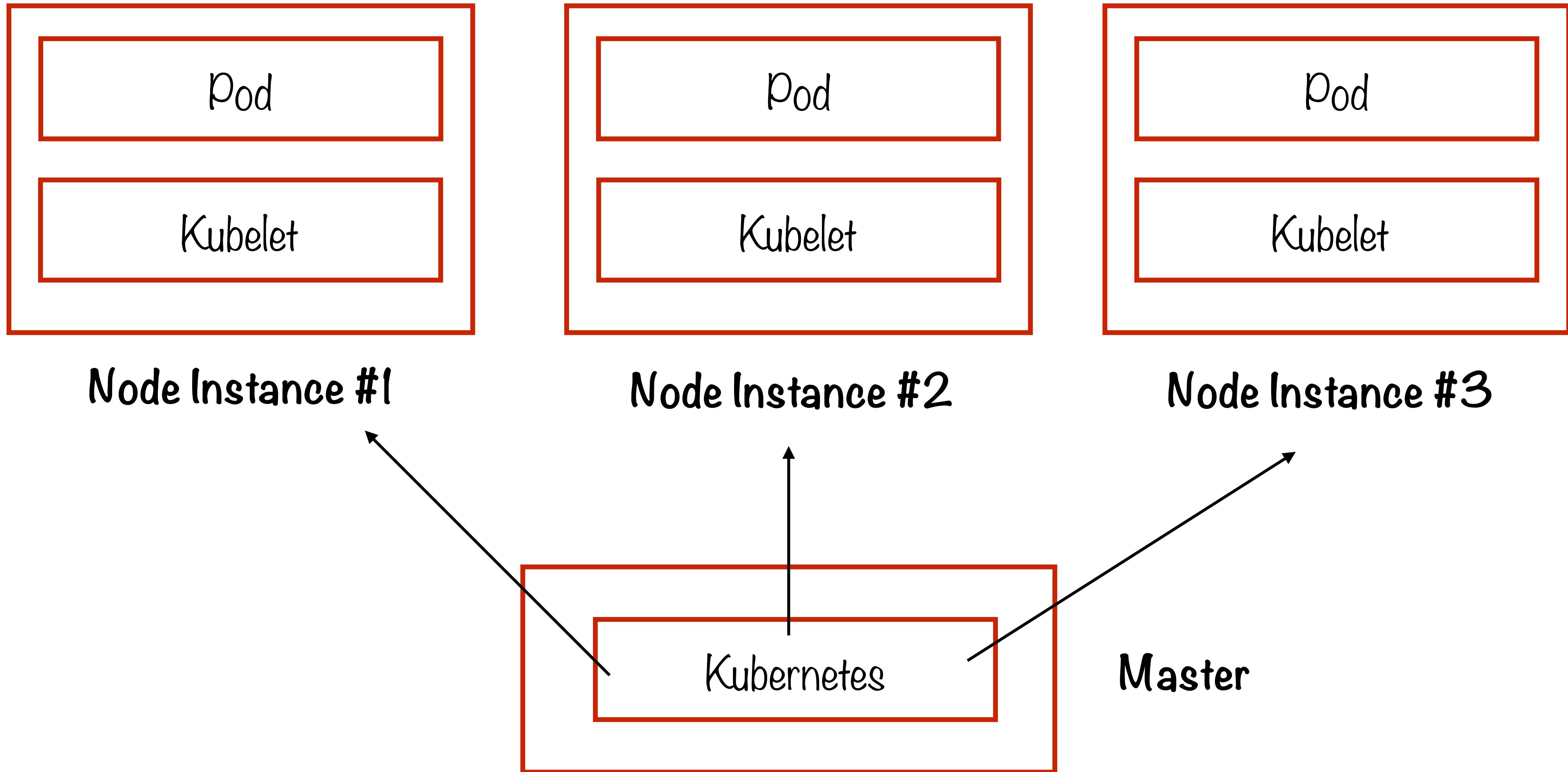
Network load balancing works out-of-box with Container Engine

For HTTP load balancing, need to integrate with Compute Engine
load balancing

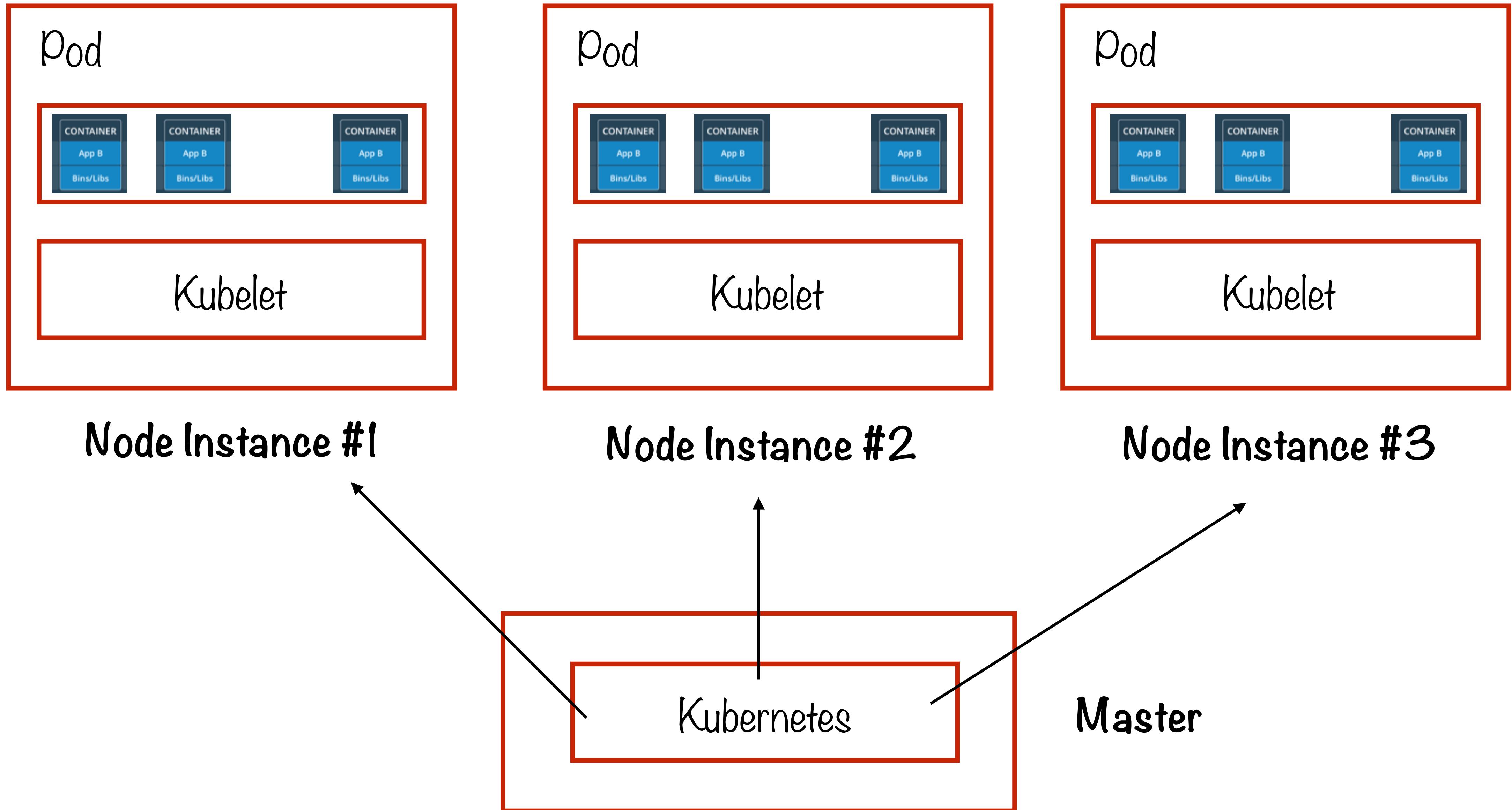
Container Cluster



Container Cluster



Container Cluster



Container Cluster

- Group of Compute Engine instances running Kubernetes.
- It consists of
 - one or more node instances, and
 - a managed Kubernetes master endpoint.

Node Instances

- Managed from the master
- Run the services necessary to support Docker containers
- Each node runs the Docker runtime and hosts a Kubelet agent, which manages the Docker containers scheduled on the host

Master Endpoint

- Managed master also runs the Kubernetes API server, which
 - services REST requests
 - schedules pod creation and deletion on worker nodes
 - synchronizes pod information (such as open ports and location)

Node Pool

- Subset of machines within a cluster that all have the same configuration.
- Useful for customizing instance profiles in your cluster
- You can also run multiple Kubernetes node versions on each node pool in your cluster, update each node pool independently, and target different node pools for specific deployments.

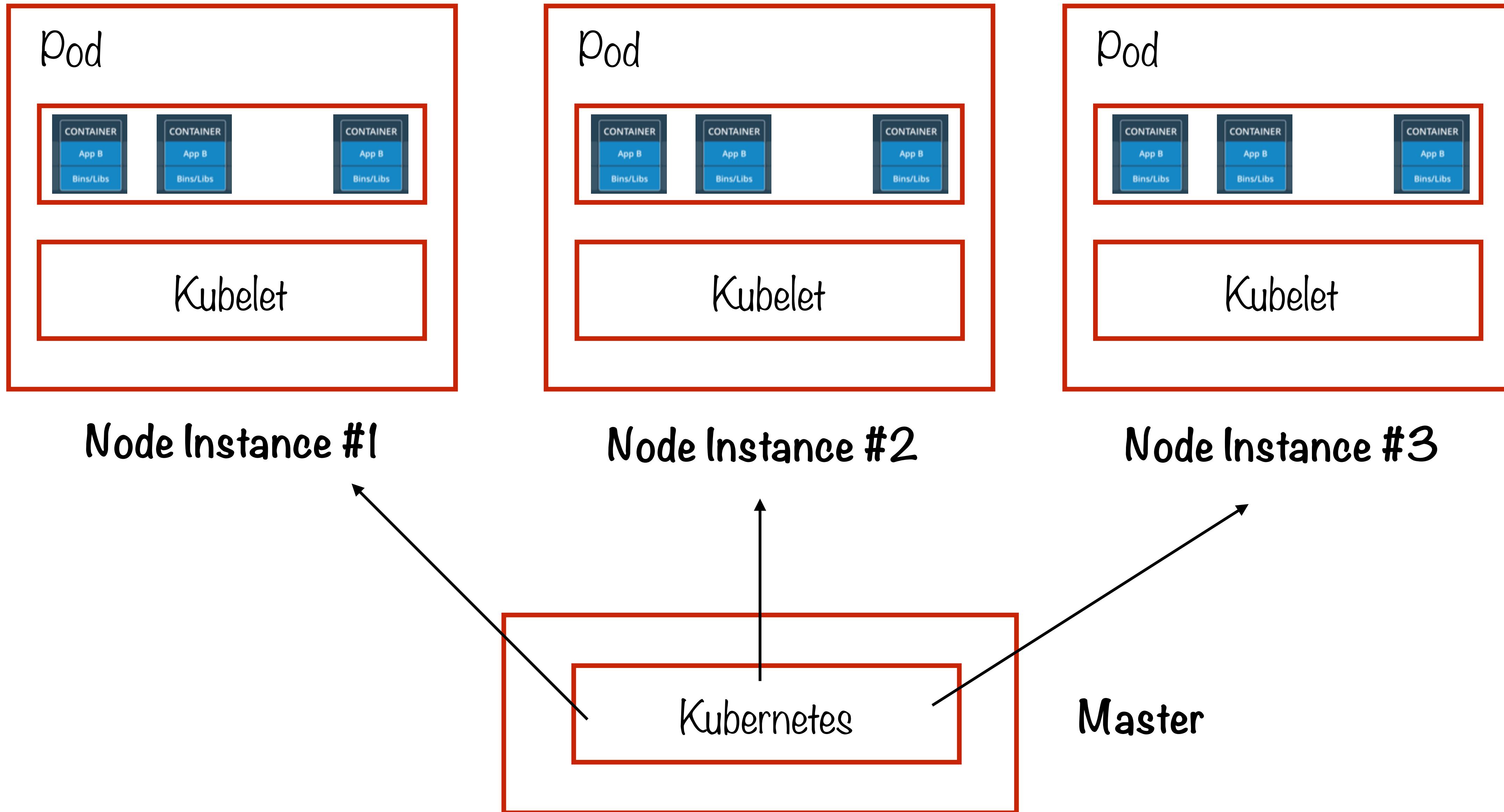
Container Builder

- Tool that executes your container image builds on Google Cloud Platform's infrastructure
- Working:
 - import source code from a variety of repositories or cloud storage spaces
 - execute a build to your specifications
 - produce artifacts such as Docker containers or Java archives.

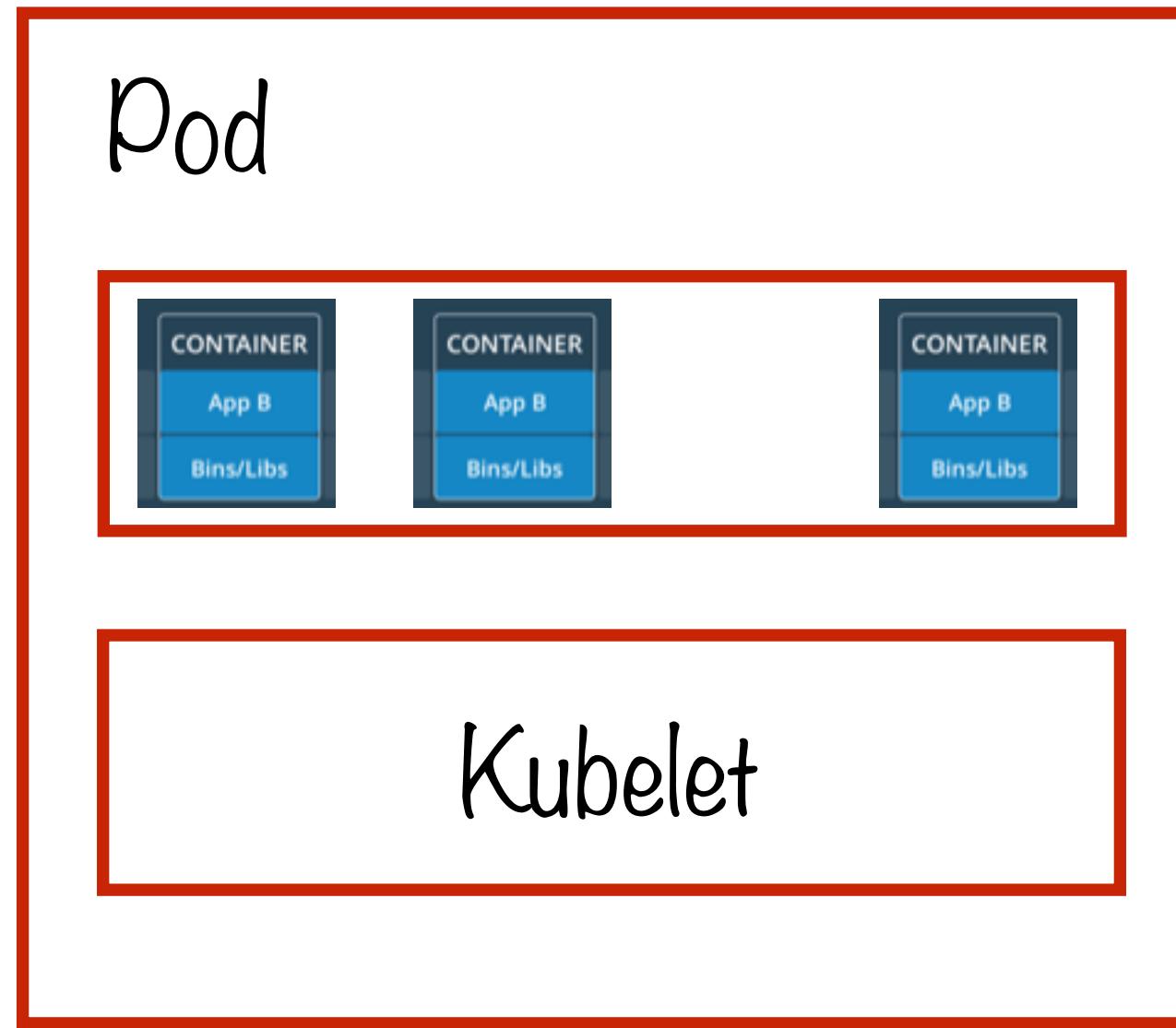
Container Registry

- Private registry for Docker images
- Can access Container Registry through secure HTTPS endpoints, which lets you push, pull, and manage images from any system, whether it's a Compute Engine instance or your own hardware
- Can use the Docker credential helper command-line tool to configure Docker to authenticate directly with Container Registry
- Can use third-party cluster management, continuous integration, or other solutions outside of Google Cloud Platform

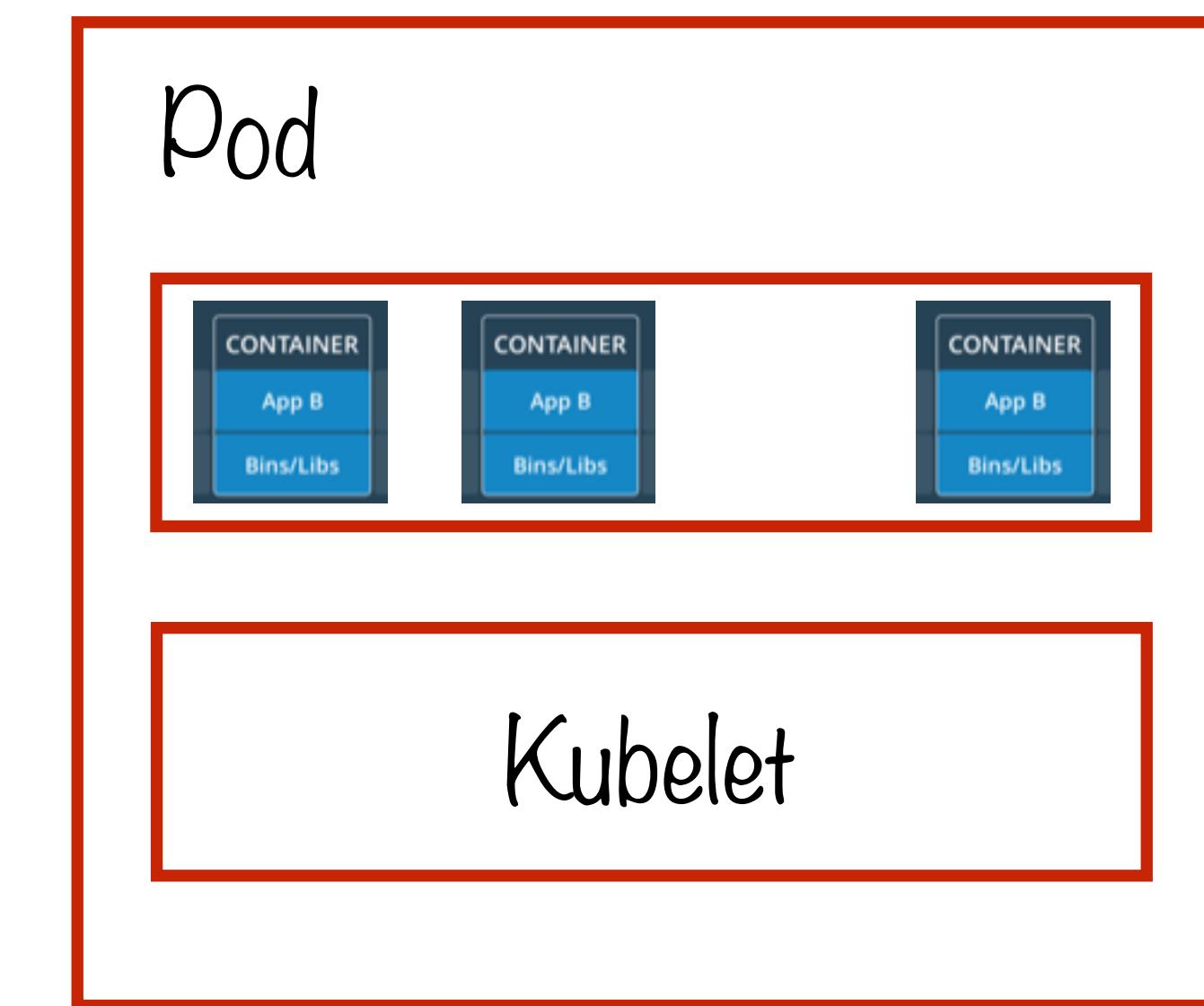
Autoscaling



Autoscaling



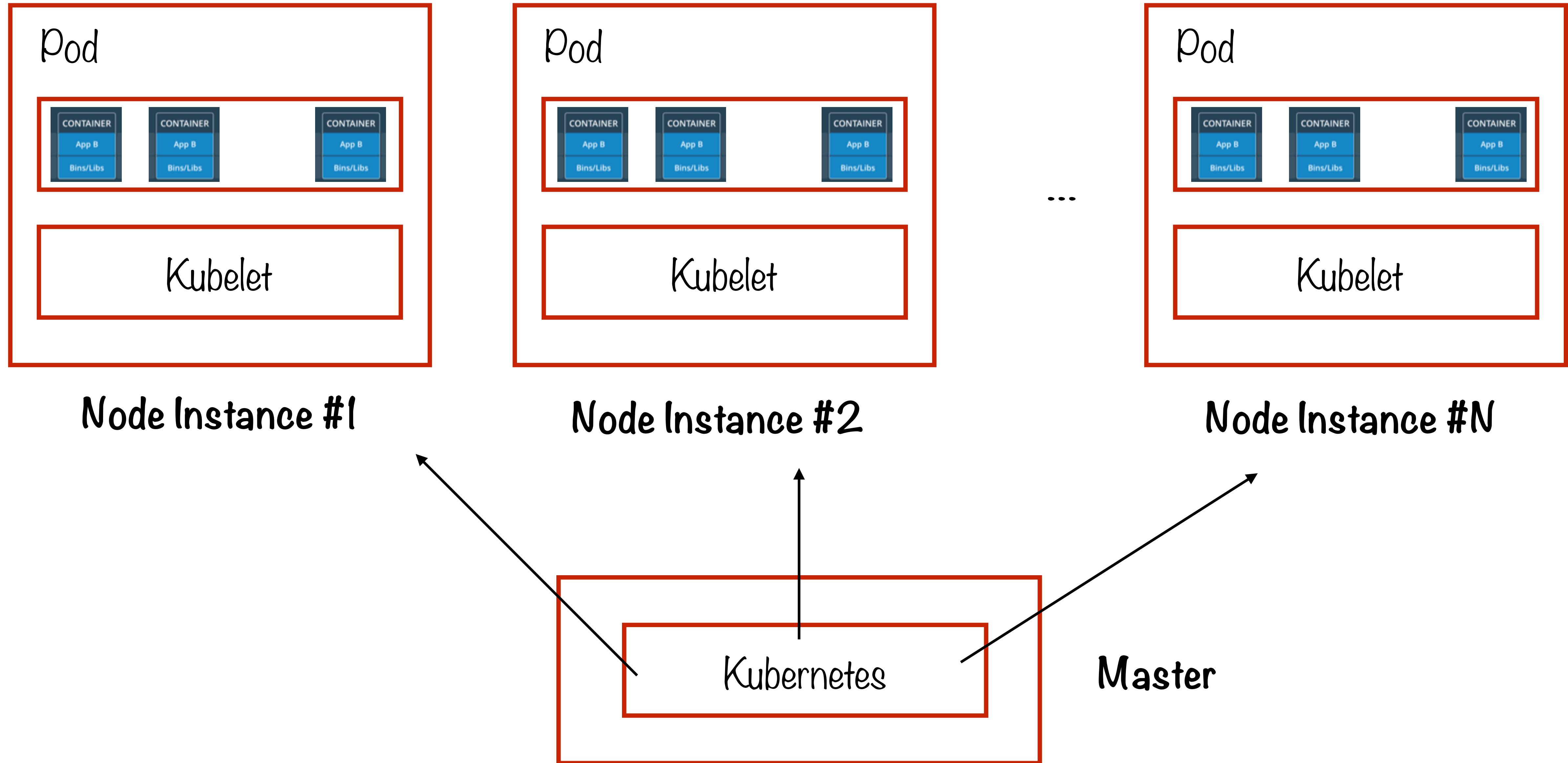
Node Instance #1



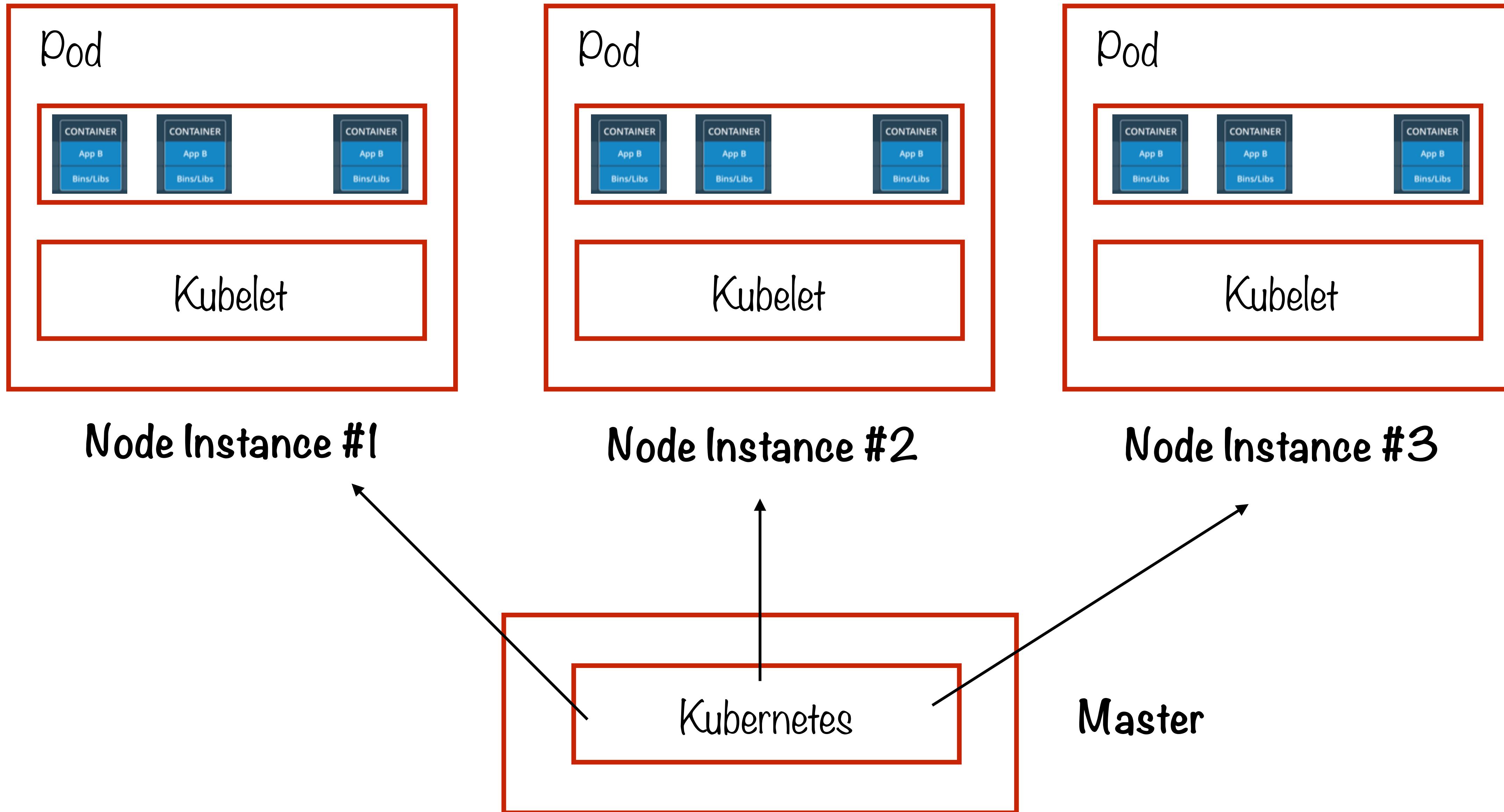
Node Instance #2



Autoscaling



Autoscaling



Autoscaling

- Automatic resizing of clusters with **Cluster Autoscaler**
- Periodically checks whether there are any pods waiting, resizes cluster if needed
- Also monitors usage of nodes and deletes nodes if all its pods can be scheduled elsewhere

Summary

AppEngine, Compute Engine and Container Engine are GCP's 3 compute options

Google AppEngine is the PaaS option - serverless and ops-free

Google ComputeEngine is the IaaS option - fully controllable down to OS

Google Container Engine lies in between - clusters of machines running Kubernetes and hosting containers

Module - Storage

Overview

Block storage for compute VMs - persistent disks or SSDs

Immutable blobs like video/images - Cloud Storage

OLTP - Cloud SQL or Cloud Spanner

NoSQL Documents like HTML/XML - Datastore

NoSQL Key-values - BigTable (~HBase)

Getting data into Cloud Storage - Transfer service

Storage Options

Use-Cases

When you need

- Storage for Compute, Block Storage
- Storing media, Blob Storage
- SQL Interface atop file data
- Document database, NoSQL
- Fast scanning, NoSQL
- Transaction Processing (OLTP)
- Analytics/Data Warehouse (OLAP)

Use

- Persistent (hard disks), SSD
- File system - maybe HDFS
- Hive (SQL-like, but MapReduce on HDFS)
- CouchDB, MongoDB (key-value/indexed database)
- HBase (columnar database)
- RDBMS
- Hive (SQL-like, but MapReduce on HDFS)

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Mobile-Specific Use-Cases

When you need

Storage for Compute, Block Storage along with mobile SDKs

Fast random access with mobile SDKs

Use

Cloud Storage for Firebase

Firebase Realtime DB

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Block Storage

Data is not structured

Lowest level of storage - no abstraction at all

Meant for use from VMs

Location tied to VM location

Block Storage

Data stored in volumes (called blocks)

Remember the options available on Compute Engine VMs

- Persistent disks
 - Standard
 - SSD
 - Local SSDs
 - (Also Cloud Storage - more in a bit)

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Cloud Storage

- Create buckets to store data
- Buckets are globally unique
 - Name (globally unique)
 - Location
 - Storage Class

Bucket Storage Classes

- Multi-regional - frequent access from anywhere in the world
- Regional - frequent access from specific region
- Nearline - accessed once a month at max
- Coldline - accessed once a year at max

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

BigQuery

- Latency bit higher than BigTable, DataStore - prefer those for low latency
- No ACID properties - can't use for transaction processing (OLTP)
- Great for analytics/business intelligence/data warehouse (OLAP)
- Recall that OLTP needs strict write consistency, OLAP does not

BigQuery

- Superficially similar in use-case to Hive
- SQL-like abstraction for non-relational data
- Underlying implementation actually quite different from Hive though

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Cloud SQL, Cloud Spanner

- Relational databases - super-structured data, constraints etc
- ACID properties - use for transaction processing (OLTP)
- Too slow and too many checks for analytics/BI/warehousing (OLAP)
- Recall that OLTP needs strict write consistency, OLAP does not

Cloud SQL, Cloud Spanner

- Cloud Spanner is Google proprietary, more advanced than Cloud SQL
- Cloud Spanner offers “horizontal scaling” - i.e. bigger data, more instances, replication etc
- Under the hood, Cloud Spanner has a surprising design - more later

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

DataStore

- Document data - eg XML or HTML - has a characteristic pattern
- Key-value structure, i.e. structured data
- Typically not used either for OLTP or OLAP
- Fast lookup on keys is the most common use-case

DataStore

- Speciality of DataStore is that query execution time depends on size of returned result (not size of data set)
- So, a returning 10 rows will take the same length of time whether dataset is 10 rows, or 10 billion rows
- Ideal for “needle-in-a-haystack” type applications, i.e. lookups of non-sequential keys

DataStore

- Indices are always fast to read, slow to write
- So, don't use for write-intensive data

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

BigTable

- Fast scanning of sequential key values - use BigTable
- Columnar database, good for sparse data
- Sensitive to hot spotting - need to design key structure carefully
- Similar to HBase

Cloud Storage

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Cloud Storage

- Create buckets to store data
- Buckets are globally unique
 - Name (globally unique)
 - Location
 - Storage Class

Bucket Storage Classes

- Multi-regional - frequent access from anywhere in the world
- Regional - frequent access from specific region
- Nearline - accessed once a month at max
- Coldline - accessed once a year at max

Bucket Storage Classes

Storage Class	Characteristics	Use Cases	Price (per GB per month)**
Multi-Regional Storage	<ul style="list-style-type: none">• 99.95% availability SLA*• Geo-redundant	Storing data that is frequently accessed ("hot" objects) around the world, such as serving website content, streaming videos, or gaming and mobile applications.	\$0.026

- Frequently accessed ("hot" objects), such as serving website content, interactive workloads, or mobile and gaming applications.
- Highest availability of the storage classes
- Geo-redundant - Cloud Storage stores your data redundantly in at least two regions separated by at least 100 miles within the multi-regional location of the bucket.

Bucket Storage Classes

Storage Class	Characteristics	Use Cases	Price (per GB per month)**
Regional Storage	<ul style="list-style-type: none">• 99.9% availability SLA*• Lower cost per GB stored• Data stored in a narrow geographic region	Storing frequently accessed in the same region as your Google Cloud DataProc or Google Compute Engine instances that use it, such as for data analytics.	\$0.02

- Appropriate for storing data that is used by Compute Engine instances.
- Better performance for data-intensive computations, as opposed to storing your data in a multi-regional location

Bucket Storage Classes

Storage Class	Characteristics	Use Cases	Price (per GB per month)**
Nearline Storage	<ul style="list-style-type: none">• 99.0% availability SLA*• Very low cost per GB stored• Data retrieval costs• Higher per-operation costs• 30-day minimum storage duration	Data you do not expect to access frequently (i.e., no more than once per month). Ideal for back-up and serving long-tail multimedia content.	\$0.01

- Slightly lower availability
- 30-day minimum storage duration
- Data you plan to read or modify on average once a month or less
- Data backup, disaster recovery, and archival storage.

Bucket Storage Classes

Storage Class	Characteristics	Use Cases	Price (per GB per month)**
Coldline Storage	<ul style="list-style-type: none">• 99.0% availability SLA*• Lowest cost per GB stored• Data retrieval costs• Higher per-operation costs• 90-day minimum storage duration	Data you expect to access infrequently (i.e., no more than once per year). Typically this is for disaster recovery, or data that is archived and may or may not be needed at some future time.	\$0.007

- Unlike other "cold" storage services, same throughput and latency (i.e. not slower to access)
- 90-day minimum storage duration, costs for data access, and higher per-operation costs
- Infrequently accessed data, such as data stored for legal or regulatory reasons

Working with Cloud Storage

- XML and JSON APIs
- Command line (gsutil)
- GCP Console (web)
- Client SDK

Domain-Named Buckets

- Cloud Storage considers bucket names that contain dots to be domain names
- Must be syntactically valid DNS names
 - E.g. bucket...example.com is not valid because it contains three dots in a row
- End with a currently-recognized top-level domain, such as .com
- Pass domain ownership verification
 - E.g. Team member creating bucket must be domain owner or manager

Domain Verification

- Number of ways to demonstrate ownership of a site or domain, including:
 - Adding a special Meta tag to the site's homepage.
 - Uploading a special HTML file to the site.
 - Verifying ownership directly from Search Console.
 - Adding a DNS TXT or CNAME record to the domain's DNS configuration.

Mobile-Specific Use-Cases

When you need

Storage for Compute, Block Storage along with mobile SDKs

Fast random access with mobile SDKs

Use

Cloud Storage for Firebase

Firebase Realtime DB

Cloud SQL

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Cloud SQL, Cloud Spanner

- Relational databases - super-structured data, constraints etc
- ACID properties - use for transaction processing (OLTP)
- Too slow and too many checks for analytics/BI/warehousing (OLAP)
- Recall that OLTP needs strict write consistency, OLAP does not

Relational Data

StudentID	Student Name
1	Jane Doe
2	John Walsh
3	Raymond Wu

CourseID	Course Name
CS101	Introduction to Computer Science
EE275	Logic Circuits
CS183	Computer Architecture

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Cloud SQL, Cloud Spanner

- Cloud Spanner is Google proprietary, more advanced than Cloud SQL
- Cloud Spanner offers “horizontal scaling” - i.e. bigger data, more instances, replication etc
- Under the hood, Cloud Spanner has a surprising design - more later

Cloud SQL

- MySQL - fast and the usual
- PostgreSQL - complex queries

Instances

- Instances need to be created explicitly
 - Not serverless
 - Specify region while creating instance
- First vs. second generation instances
 - Second generation instances allow proxy support - no need to whitelist IP addresses or configure SSL
 - Higher availability configuration
 - Maintenance won't take down the server

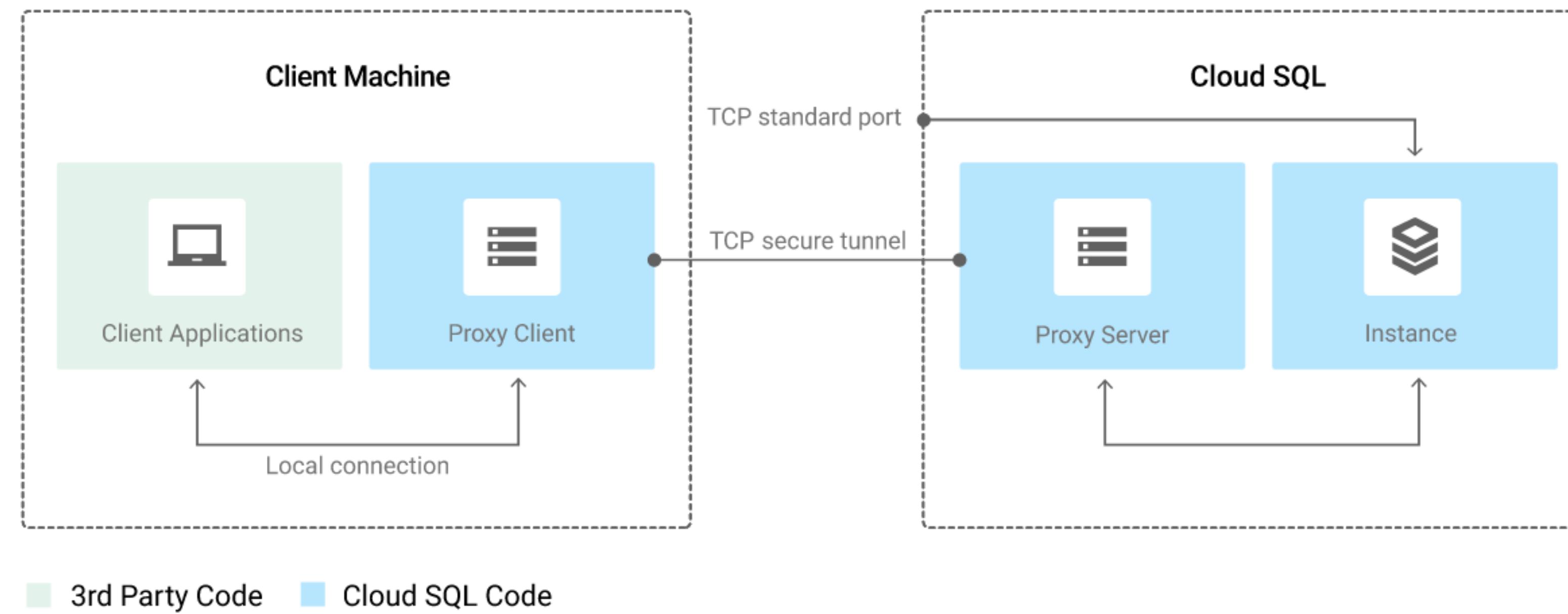
High Availability Configuration

- A Second Generation instance is in an high availability configuration when it has a failover replica
- The failover replica must be in a different zone than the original instance, also called the master
- All changes made to the data on the master, including to user tables, are replicated to the failover replica using semisynchronous replication.

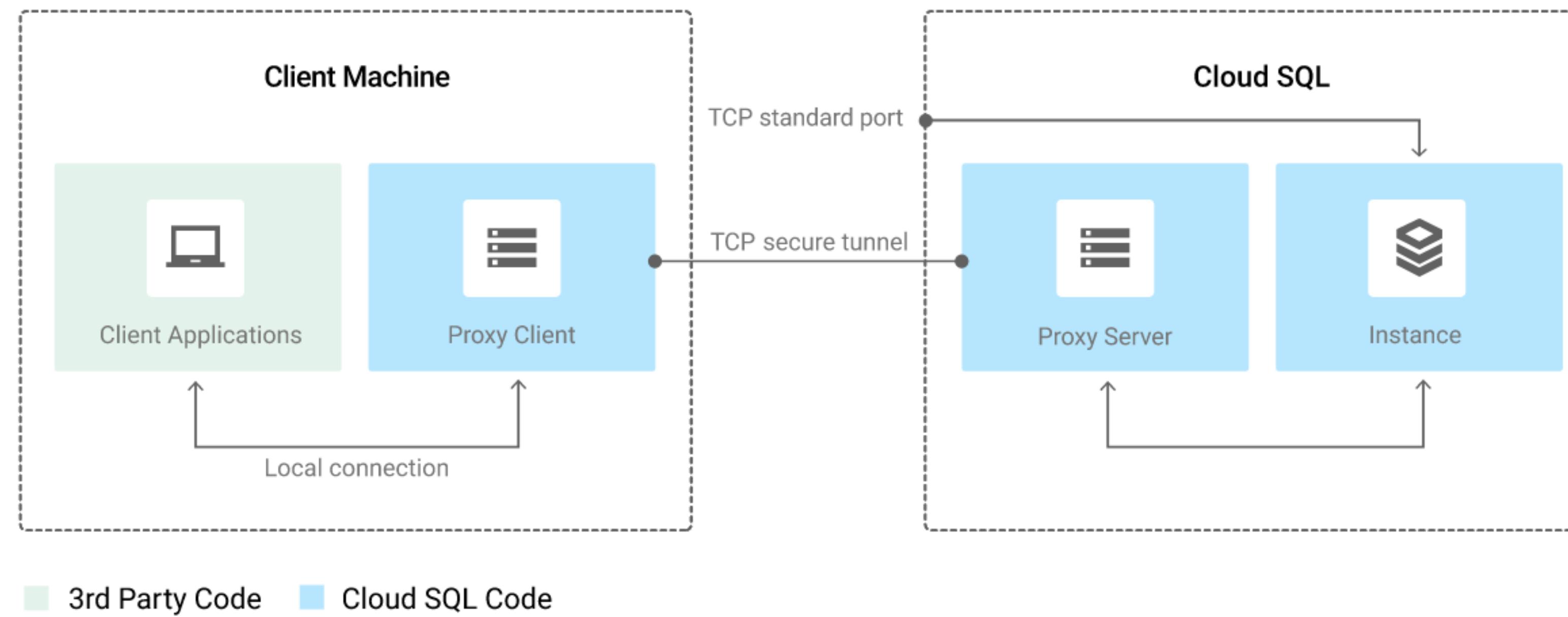
Cloud Proxy

- Provides secure access to your Cloud SQL Second Generation instances without having to whitelist IP addresses or configure SSL.
- Secure connections: The proxy automatically encrypts traffic to and from the database; SSL certificates are used to verify client and server identities.
- Easier connection management: The proxy handles authentication with Google Cloud SQL, removing the need to provide static IP addresses.

Cloud Proxy

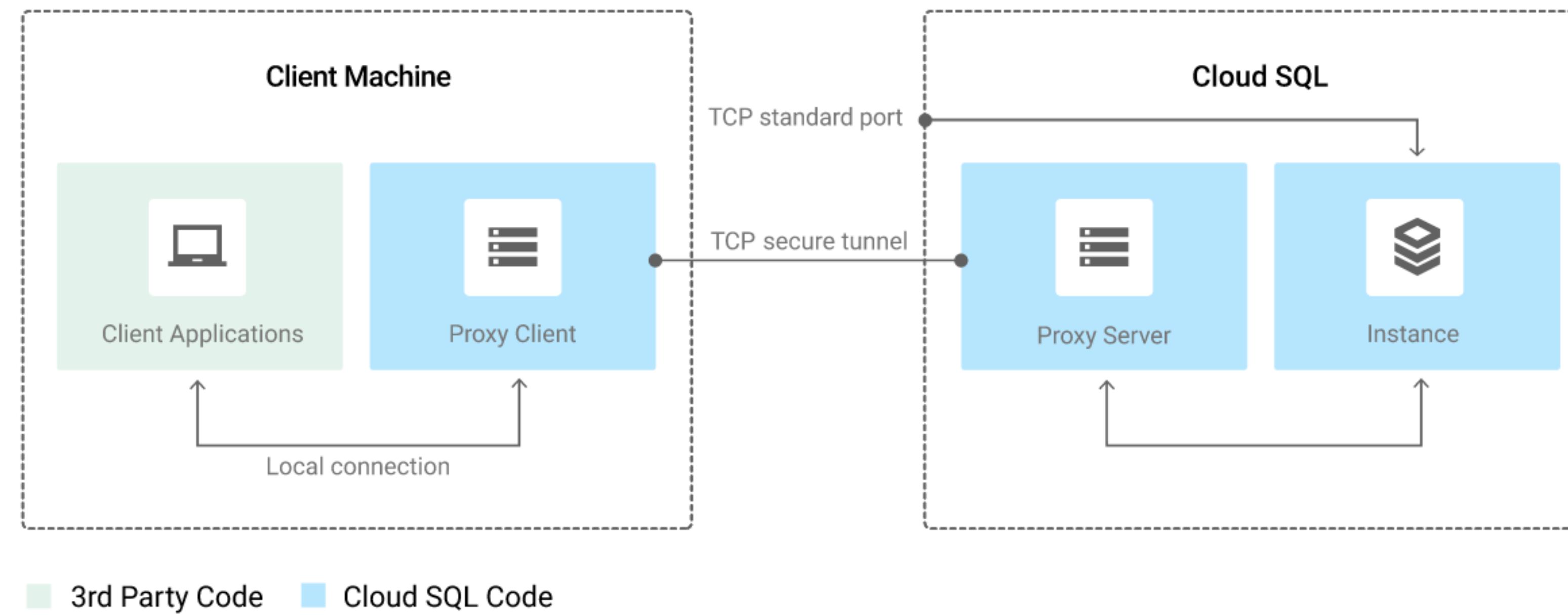


Cloud Proxy



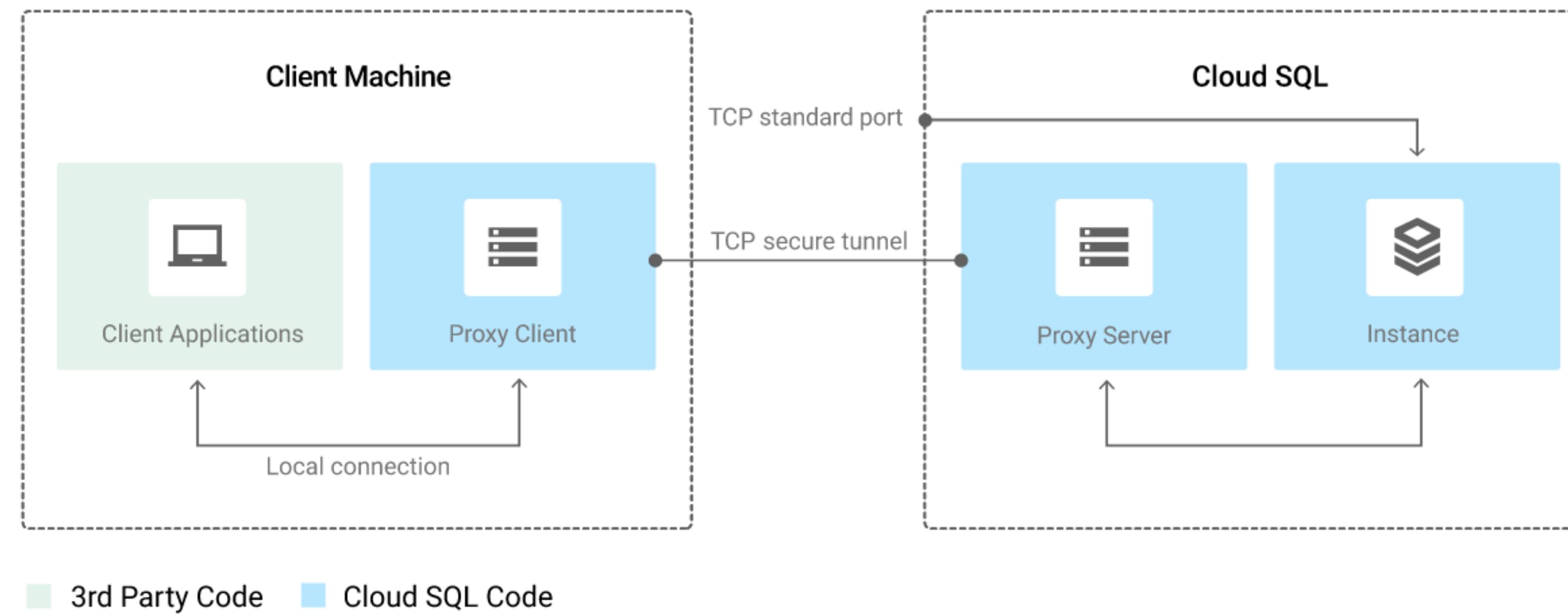
The Cloud SQL Proxy works by having a local client, called the proxy, running in the local environment

Cloud Proxy



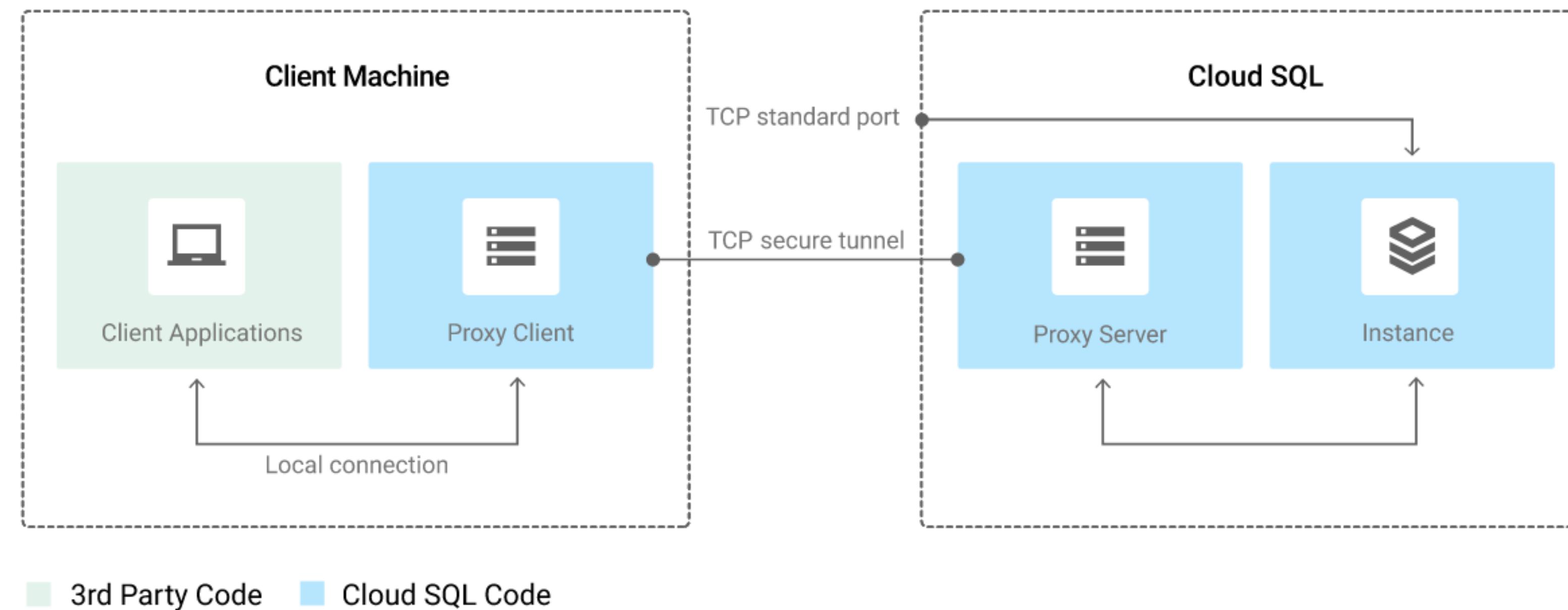
Your application communicates with the proxy with the standard database protocol
used by your database

Cloud Proxy



The proxy uses a secure tunnel to communicate with its companion process running on the server.

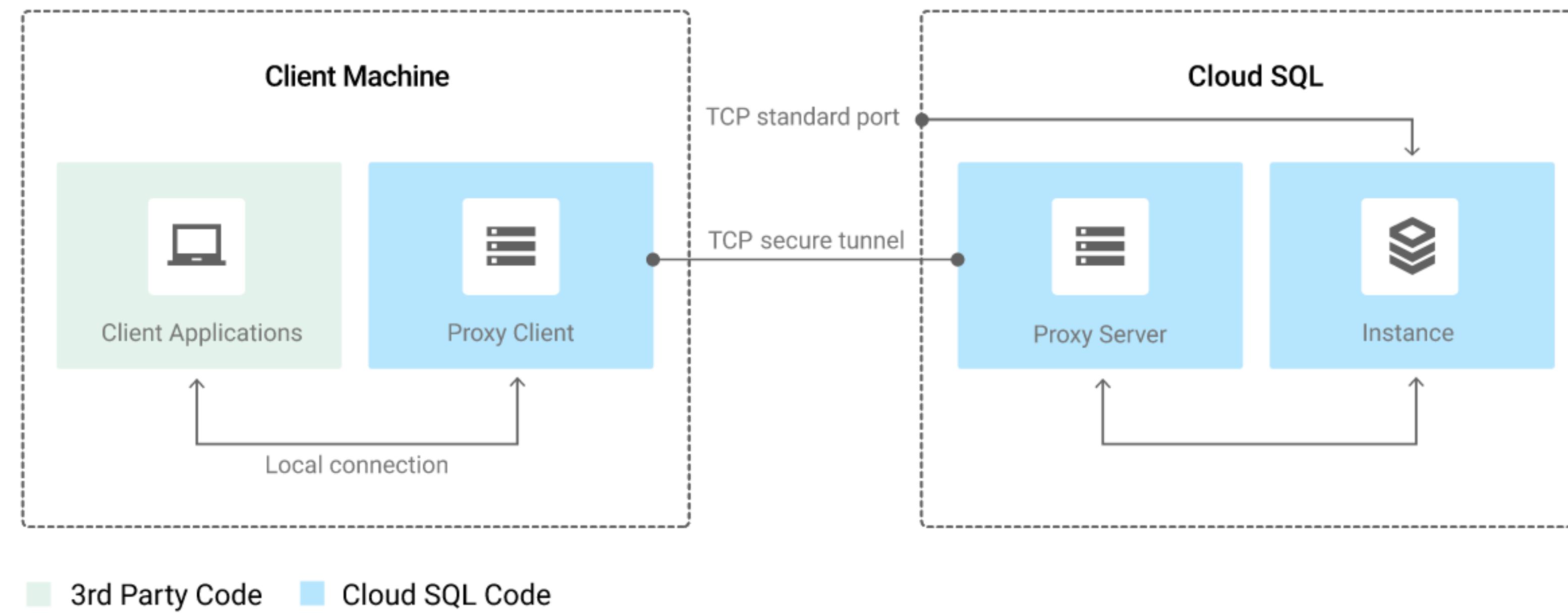
Cloud Proxy



When you start the proxy, need to tell it

- What Cloud SQL instances it should establish connections to
- Where it will listen for data coming from your application to be sent to Cloud SQL
- Where it will find the credentials it will use to authenticate your application to Cloud SQL

Cloud Proxy



You can install the proxy anywhere in your local environment. The location of the proxy binaries does not impact where it listens for data from your application.

Cloud Spanner

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Cloud SQL, Cloud Spanner

- Relational databases - super-structured data, constraints etc
- ACID properties - use for transaction processing (OLTP)
- Too slow and too many checks for analytics/BI/warehousing (OLAP)
- Recall that OLTP needs strict write consistency, OLAP does not

Cloud SQL, Cloud Spanner

- Cloud Spanner is Google proprietary, more advanced than Cloud SQL
- Cloud Spanner offers “horizontal scaling” - i.e. bigger data, more instances, replication etc
- Under the hood, Cloud Spanner has a surprising design - more later

Cloud Spanner

- Use when
 - Need high availability
 - strong consistency
 - transactional reads and writes (especially writes!)
- Don't use if
 - Data is not relational, or not even structured
 - Want an open source RDBMS
 - Strong consistency and availability is overkill

Data Model

- Databases contain tables (as usual)
- Tables 'look' relational - rows, columns, strongly typed schemas
- But...

Relational Data

StudentID	Student Name
1	Jane Doe
2	John Walsh
3	Raymond Wu

CourseID	Course Name
CS101	Introduction to Computer Science
EE275	Logic Circuits
CS183	Computer Architecture

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Relational Data

StudentID	Student Name
1	Jane Doe
2	John Walsh
3	Raymond Wu

- Usually query student and course grades together
- Most common query = get transcript
- Specify a parent-child relationship for efficient storage

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Interleaved Representation

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
1	Jane Doe	CS183	Summer 2012	A+
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+

Parent- Child

- Parent-child relationships between tables
- These cause physical location for fast access
- If you query Students and Grades together, make Grades child of Students
- Data locality will be enforced between 2 independent tables!

Parent- Child

- Every table **must** have primary keys
- To declare table is child of another...
- Prefix parent's primary key onto primary key of child
- (This storage model resembles HBase btw)

Interleaving

- Rows are stored in sorted order of primary key values
- Child rows are inserted between parent rows with that key prefix
- “Interleaving”
- Fast sequential access - like HBase

Hotspotting

- As in HBase - need to choose Primary key carefully
- Do not use monotonically increasing values, else writes will be on same locations - hot spotting
- Use hash of key value if you naturally monotonically ordered keys
- Under the hood, Cloud Scanner divides data among servers across key ranges

Hotspotting

StudentID	Student Name
1	Jane Doe
2	John Walsh
3	Raymond Wu

- Usually query student and course grades together
- Most common query = get transcript
- Specify a parent-child relationship for efficient storage

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Interleaved Representation

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
1	Jane Doe	CS183	Summer 2012	A+
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+

Hotspotting

StudentID	Student Name
X23	Jane Doe
F23	John Walsh
B99	Raymond Wu

- Change key so that it is not monotonically increasing
- Hash StudentID values

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Interleaved Representation

StudentID	Student Name	CourseID	Term	Grade
X23	Jane Doe	CS101	Fall 2015	A-
X23	Jane Doe	CS183	Summer 2012	A+
F23	John Walsh	CS294	Spring 2016	B+
B99	Raymond Wu	ME101	Winter 2015	C+

Splits

- Parent-child relationships can get complicated - up to 7 layers deep!
- CloudScanner is distributed - uses “splits”
- A split is a range of rows that can be moved around independent of others
- Splits are added to distribute high read-write data (to break up hotspots)
- Splits are, of course, influenced by parent-child relationships

Splits

StudentID	Student Name	CourseID	Term	Grade
X23	Jane Doe	CS101	Fall 2015	A-
X23	Jane Doe	CS183	Summer 2012	A+
F23	John Walsh	CS294	Spring 2016	B+
B99	Raymond Wu	ME101	Winter 2015	C+

Secondary Indices

- Like in HBase, key-based storage ensures fast sequential scan of keys
 - Remember that tables must have primary keys
- Unlike in HBase, can also add secondary indices
 - Might cause same data to be stored twice
 - Fine-grained control on use of indices
 - Force query to use a specific index (index directives)
 - Force column to be copied into a secondary index (STORING clause)

Relational Data

StudentID	Student Name
1	Jane Doe
2	John Walsh
3	Raymond Wu

CourseID	Course Name
CS101	Introduction to Computer Science
EE275	Logic Circuits
CS183	Computer Architecture

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Primary Index - Interleaved Representation

StudentID	Student Name	CourseID	Term	Grade
X23	Jane Doe	CS101	Fall 2015	A-
X23	Jane Doe	CS183	Summer 2012	A+
F23	John Walsh	CS294	Spring 2016	B+
B99	Raymond Wu	ME101	Winter 2015	C+

Relational Data

CourseID	Course Name
CS101	Introduction to Computer Science
EE275	Logic Circuits
CS183	Computer Architecture

- Usually query student and course grades together
- But also query courses and grades
- So, create secondary index

StudentID	Student Name	CourseID	Term	Grade
1	Jane Doe	CS101	Fall 2015	A-
2	John Walsh	CS294	Spring 2016	B+
3	Raymond Wu	ME101	Winter 2015	C+
1	Jane Doe	CS183	Summer 2012	A+

Secondary Index - Interleaved Representation

CourseID	CourseName	StudentID	Term	Grade
CS101	Introduction to Computer Science	X23	Fall 2015	A-
CS101	Jane Doe	F23	Fall 2002	B+
EE275	Logic Circuits	EE275	Spring 2016	A-
EE275	John Walsh	EE275	Spring 2016	A-
CSI83	Computer Architecture	CSI83	Winter 2015	C+
CSI83	Raymond Wu	CSI83	Winter 2015	C+

Data Types

- Remember that tables are strongly-typed (schemas must have types)
- Non-normalized types such as ARRAY and STRUCT available too
- STRUCTs are not OK in tables, but can be returned by queries (eg if query returns ARRAY of ARRAYS)
- ARRAYS are OK in tables, but ARRAYS of ARRAYS are not

Transactions

- Supports serialisability
- Cloud Spanner transaction support is super-strong, even stronger than traditional ACID
 - Transactions commit in an order that is reflected in their commit timestamps
 - These commit timestamps are "real time" so you can compare them to your watch

Transactions

- Two transaction modes
 - Locking read-write (slow)
 - Read-only (fast)
- If making a one-off read, use something known as a “Single Read Call”
 - Fastest, no transaction checks needed!

Staleness

- Can set **timestamp bounds**
- Strong - “read latest data”
- Bounded Staleness - “read version no later than ...”
- Exact Staleness - “read at exactly ...”
 - (could be in past or future)
- Cloud Scanner has a version-gc that reclaims versions older than 1 hour old

Bigtable

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

BigTable

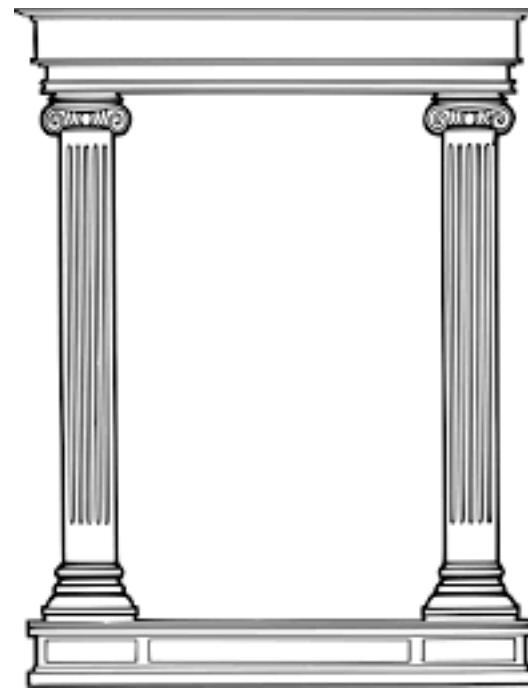
- Fast scanning of sequential key values - use BigTable
- Columnar database, good for sparse data
- Sensitive to hot spotting - need to design key structure carefully
- Similar to HBase

BigTable and HBase

- BigTable is basically GCP's managed HBase
 - This is a much stronger link than between say Hive and BigQuery!
- Usual advantages of GCP -
 - scalability
 - low ops/admin burden
 - cluster resizing without downtime
 - many more column families before performance drops (~100 OK)

HBase vs. Relational Databases

Properties of HBase



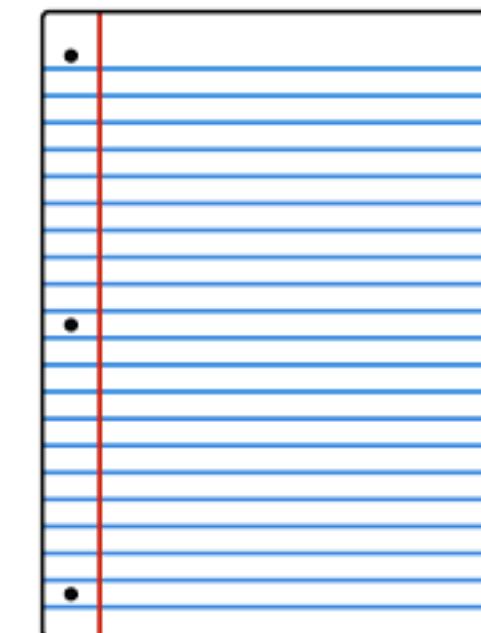
Columnar store



Denormalized storage

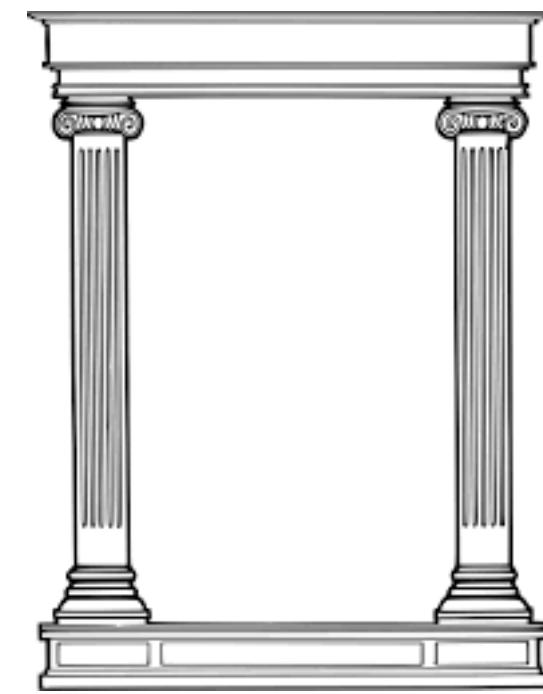


Only CRUD operations



ACID at the row level

A Notification Service

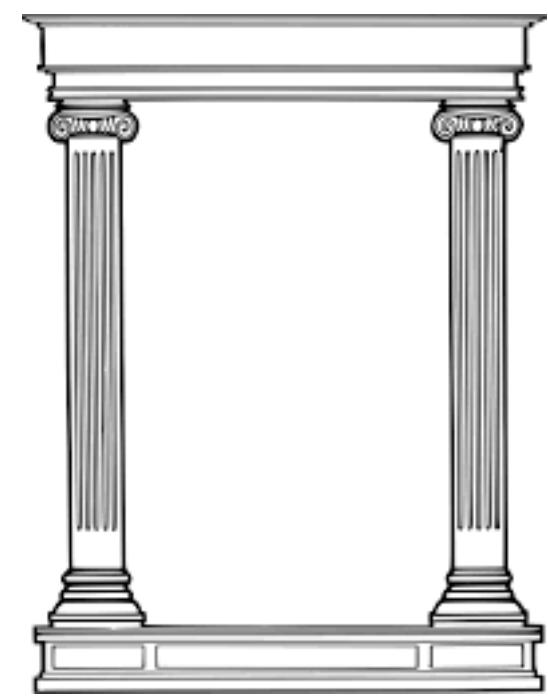


Columnar store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

Layout of a traditional relational database

A Notification Service

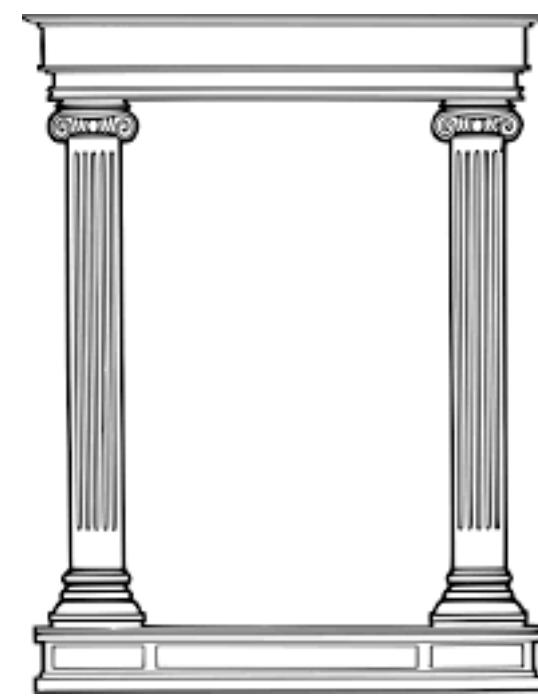


Columnar store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

Layout of a traditional relational database

A Notification Service



Columnar store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

Row = 3

Column = To

Columnar Store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Columnar Store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Columnar Store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

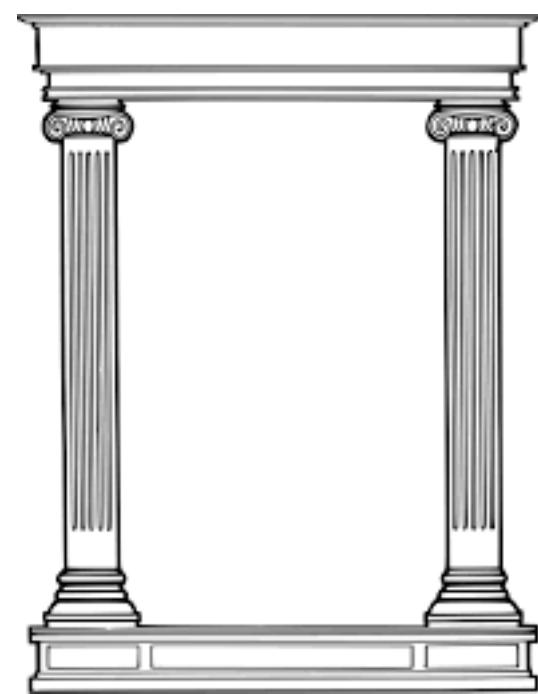
Columnar Store

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Advantages of a Columnar Store



Columnar store

Sparse tables: No wastage of space when storing sparse data

Dynamic attributes: Update attributes dynamically without changing storage structure

Sparse Tables

Id	To	Type	Content	Expiry
1	mike	offer	Offer on mobiles	2345689070
2	john	sale	Redmi sale	
3	jill	order	Order delivered	
4	megan	sale	Clothes sale	2456123989

Sale and offer notifications may have an
expiry time

Sparse Tables

Id	To	Type	Content	Expiry	Order Status
1	mike	offer	Offer on mobiles	2345689070	
2	john	sale	Redmi sale		
3	jill	order	Order delivered		Delivered
4	megan	sale	Clothes sale	2456123989	

Order related notifications may have
an order status

Sparse Tables

Id	To	Type	Content	Expiry	Order Status
1	mike	offer	Offer on mobiles	2345689070	
2	john	sale	Redmi sale		
3	jill	order	Order delivered		Delivered
4	megan	sale	Clothes sale	2456123989	

In a traditional database this results in a
change in **database structure**

Sparse Tables

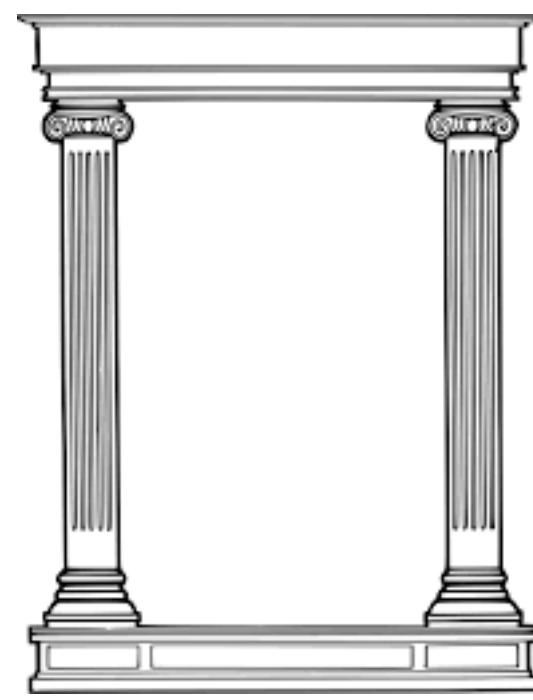
Id	To	Type	Content	Expiry	Order Status
1	mike	offer	Offer on mobiles	2345689070	
2	john	sale	Redmi sale		
3	jill	order	Order delivered		Delivered
4	megan	sale	Clothes sale	2456123989	

And empty cells when data is not applicable to certain rows

Sparse Tables

Id	To	Type	Content	Expiry	Order Status
1	mike	offer	Offer on mobiles	2345689070	
2	john	sale	Redmi sale		
3	jill	order	Order delivered		Delivered
4	megan	sale	Clothes sale	2456123989	

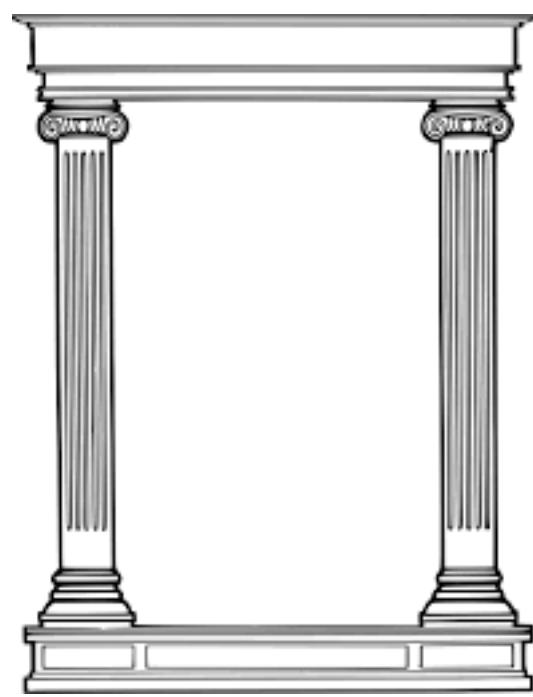
These cells still occupy space!



Columnar store

Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
1	Expiry	2345689070
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale
4	Expiry	2456123989

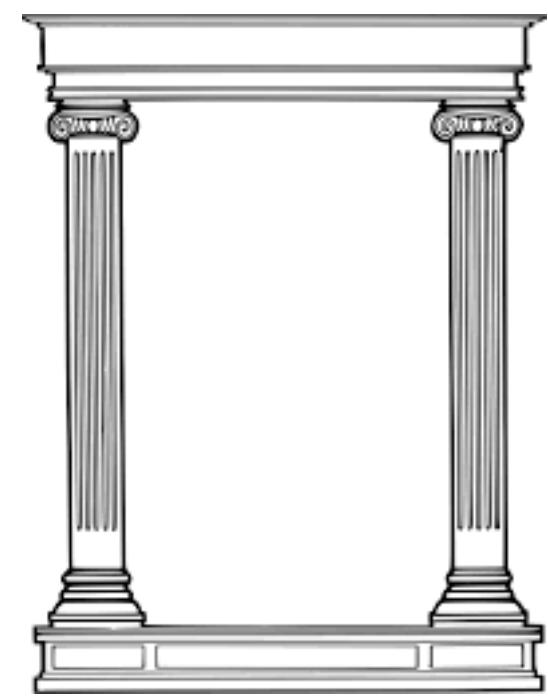
Dynamically add new attributes as rows
in this table



Columnar store

Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
1	Expiry	2345689070
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale
4	Expiry	2456123989

No wastage of space with empty cells!

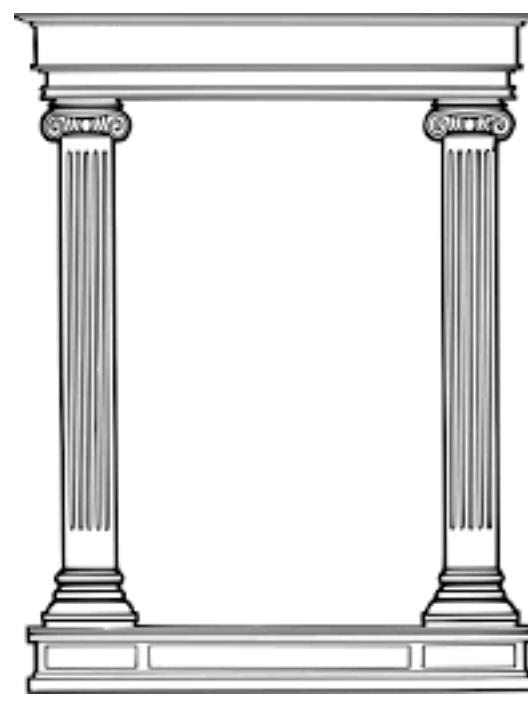


Columnar store

Note that this is not the exact layout
of how data is stored in HBase

It is a **general structure** of
how columnar stores are
constructed

Properties of HBase



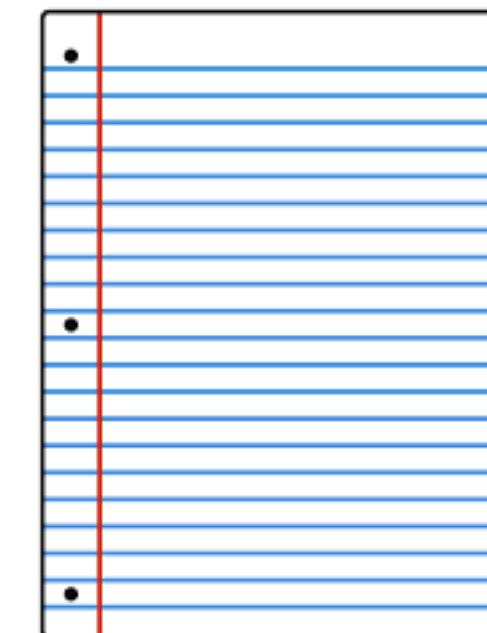
Columnar store



Denormalized storage



Only CRUD operations



ACID at the row level



Denormalized storage

Traditional databases use
normalized forms of database
design to minimize redundancy



Denormalized storage

Minimize Redundancy

Employee Details

Employee Subordinates

Employee Address



Denormalized storage

Employee Details

Id	Name	Function	Grade
1	Emily	Finance	6

Employee Subordinates

Id	Subordinate Id
1	2
1	3

Employee Address

Id	City	Zip Code
1	Palo Alto	94305
2	Seattle	98101



Denormalized storage

Employee Details

Id	Name	Function	Grade
1	Emily	Finance	6
2	John	Finance	3
3	Ben	Finance	4

All employee details in one table



Denormalized storage

Employee Subordinates

Id	Subordinate Id
1	2
1	3

Employees referenced only by ids
everywhere else



Denormalized storage

Employee Address

Id	City	Zip Code
1	Palo Alto	94305
2	Seattle	98101

Data is made more granular by splitting it across multiple tables



Denormalized storage

Id	Name	Function	Grade
1	Emily	Finance	6

Id	Subordinate Id
1	2
1	3

Id	City	Zip Code
1	Palo Alto	94305
2	Seattle	98101

Normalization



Denormalized storage

Normalization

Optimizes storage

But storage is cheap in a
distributed system!



Denormalized storage

But storage is cheap in a
distributed system!

Optimize number of
disk seeks

Denormalized Storage

Id	Name	Function	Grade
1	Emily	Finance	6
2	John	Finance	3
3	Ben	Finance	4

Id	Subordinate Id
1	2
1	3



Id	Name	Function	Grade	Subordinates
1	Emily	Finance	6	<ARRAY>
2	John	Finance	3	
3	Ben	Finance	4	

Denormalized Storage

Id	Name	Function	Grade
1	Emily	Finance	6
2	John	Finance	3
3	Ben	Finance	4

Id	City	Zip Code
1	Palo Alto	94305
2	Seattle	98101



Id	Name	Function	Grade	Subordinates	Address
1	Emily	Finance	6	<ARRAY>	<STRUCT>
2	John	Finance	3		
3	Ben	Finance	4		

Denormalized Storage

Id	Name	Function	Grade	Subordinates	Address
1	Emily	Finance	6	<ARRAY>	<STRUCT>
2	John	Finance	3		
3	Ben	Finance	4		

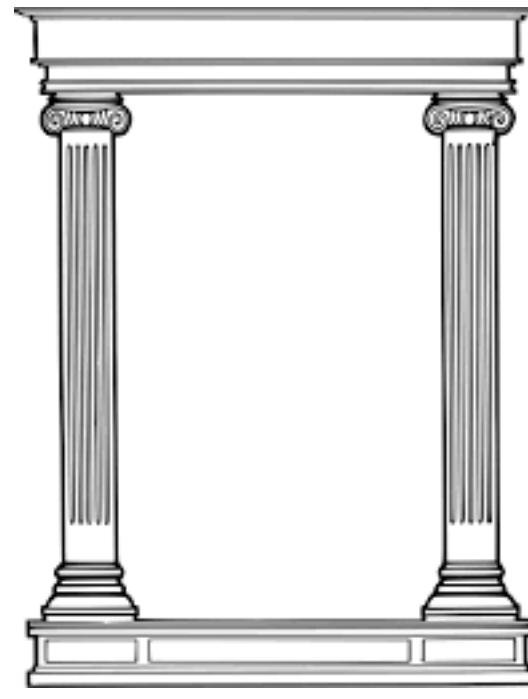
Store everything related to an employee in the same table

Denormalized Storage

Id	Name	Function	Grade	Subordinates	Address
1	Emily	Finance	6	<ARRAY>	<STRUCT>
2	John	Finance	3		
3	Ben	Finance	4		

Read a single record to get all details
about an employee in one read operation

Properties of HBase



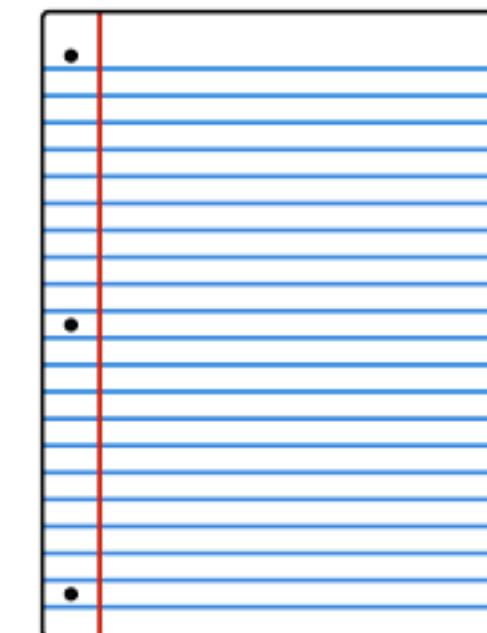
Columnar store



Denormalized storage



Only CRUD operations



ACID at the row level



Traditional Databases and SQL

Only CRUD operations

Joins: Combining information across tables using keys

Group By: Grouping and aggregating data for the groups

Order By: Sorting rows by a certain column



Only CRUD operations

HBase does not
support SQL

NoSQL



Only CRUD operations

Only a limited set of operations are
allowed in HBase

Create

Read

Update

Delete

CRUD



Only CRUD operations

No operations involving multiple tables

No indexes on tables

No constraints

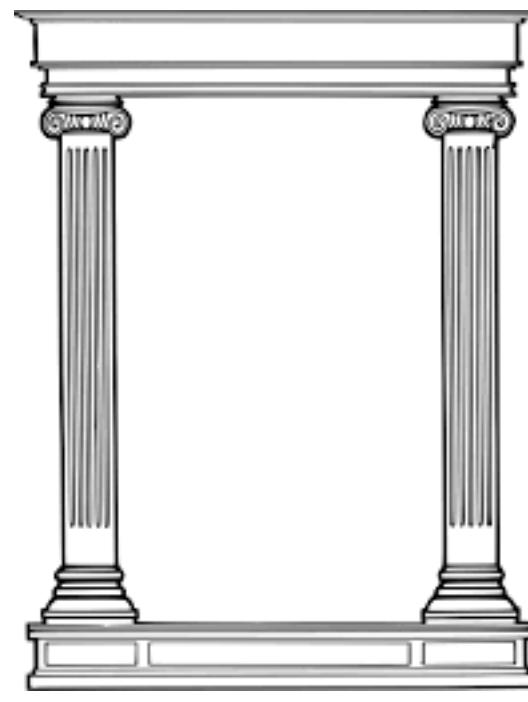


Only CRUD operations

Id	Name	Function	Grade	Subordinates	Address

This is why all details need to
be **self contained** in one row

Properties of HBase



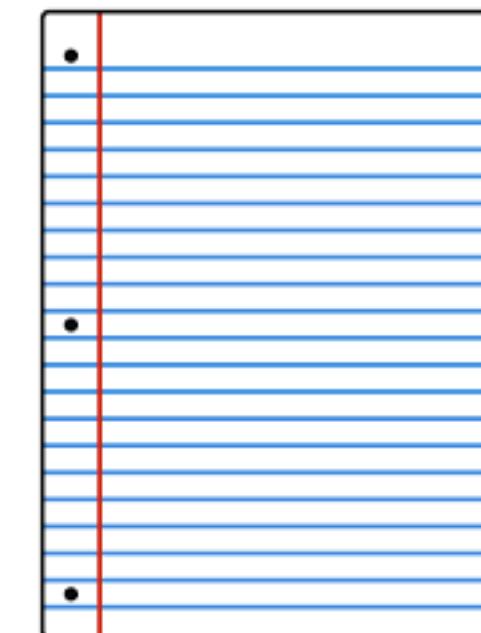
Columnar store



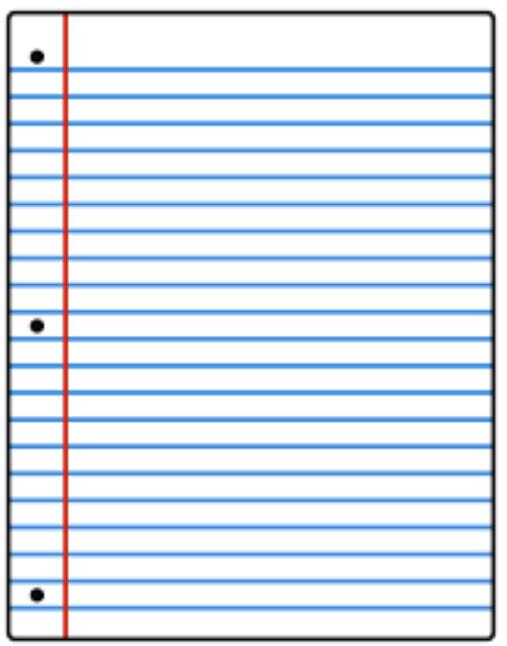
Denormalized storage



Only CRUD operations



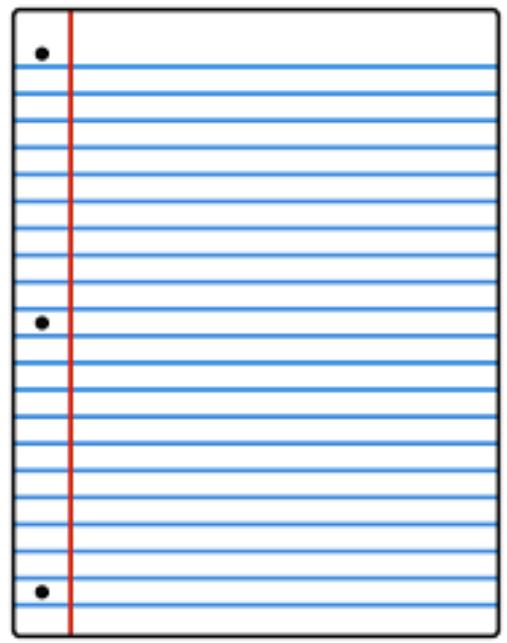
ACID at the row level



ACID at the row level

Updates to a single row are atomic

All columns in a row
are updated or none are



ACID at the row level

Updates to multiple rows are **not** atomic

Even if the update is on the same column in multiple rows

Traditional RDBMS vs. HBase

Traditional RDBMS

Data arranged in rows and columns

Supports SQL

Complex queries such as grouping, aggregates, joins etc

Normalized storage to minimize redundancy and optimize space

ACID compliant

HBase

Data arranged in a column-wise manner

NoSQL database

Only basic operations such as create, read, update and delete

Denormalized storage to minimize disk seeks

ACID compliant at the row level

How Is Data Laid out in HBase?

Notification Data in a Traditional Database

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

This is a 2-dimensional data model

Notification Data in a Traditional Database

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

unique row id

column name

4-dimensional Data Model

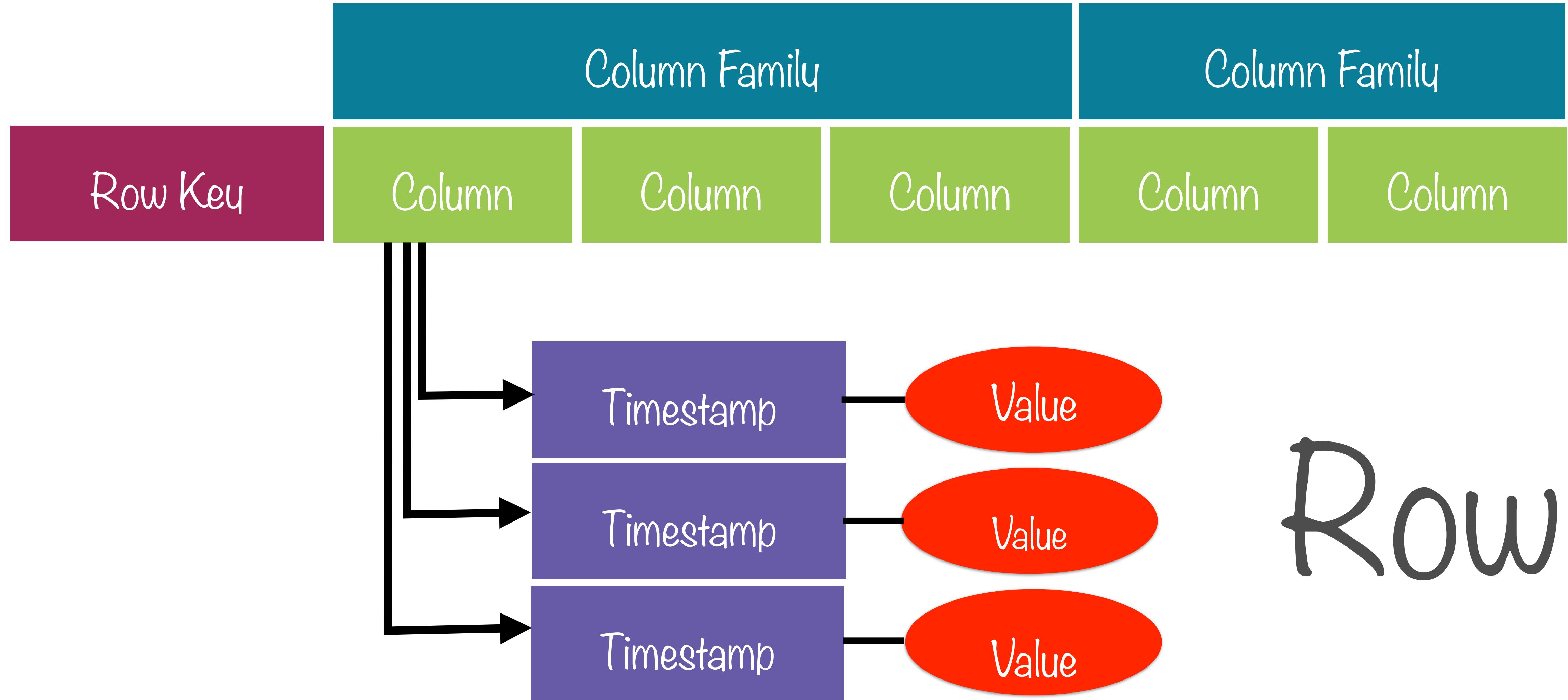
Row Key

Column Family

Column

Timestamp

4-dimensional Data Model



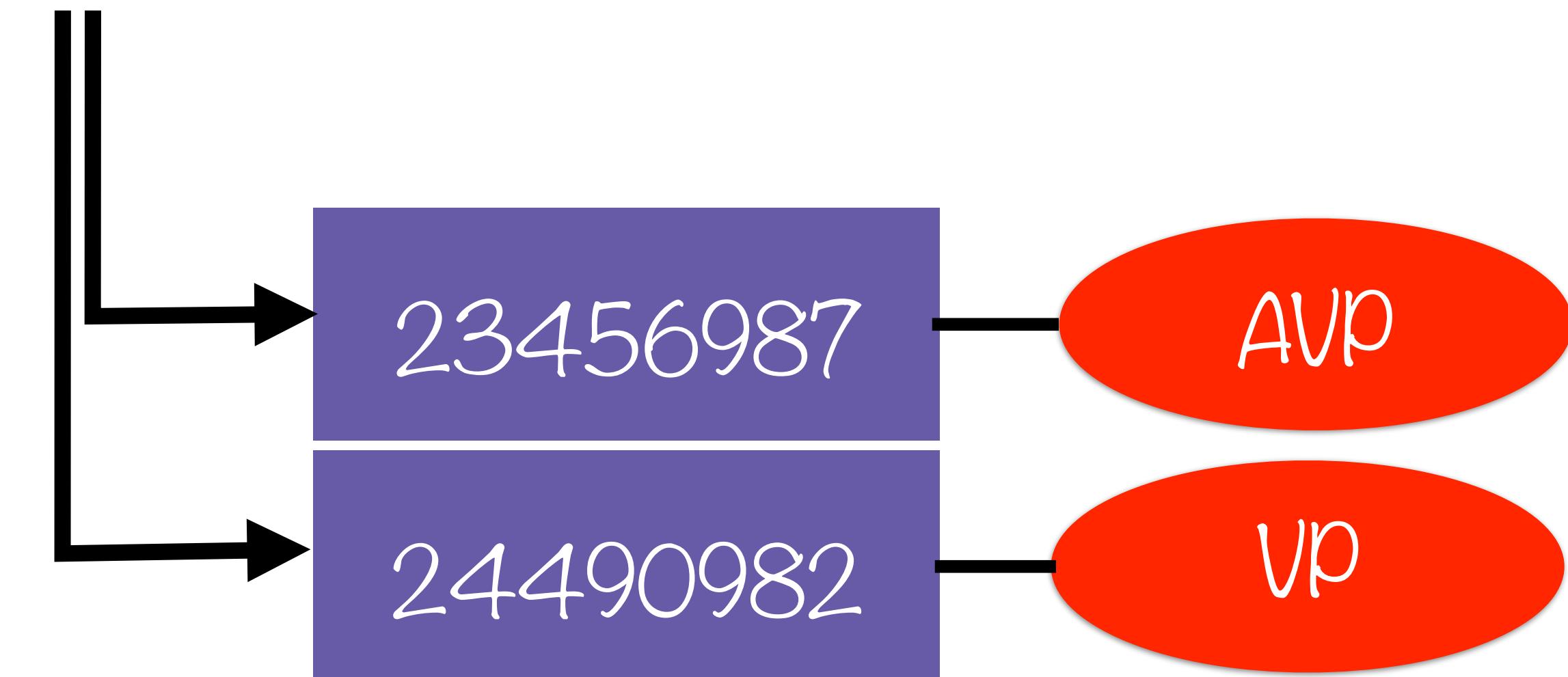
A Table for Employee Data

Work		Personal			
EmplID	Dept	Grade	Title	Name	SSN

A Table for Employee Data

Empld	Work			Personal		
	Dept	Grade	Title	Name	SSN	

A record for a
single employee



Notification Data

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Notification Data

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Row Key

Notification Data

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Column Family

Notification Data

Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale

Columns



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Notification Data

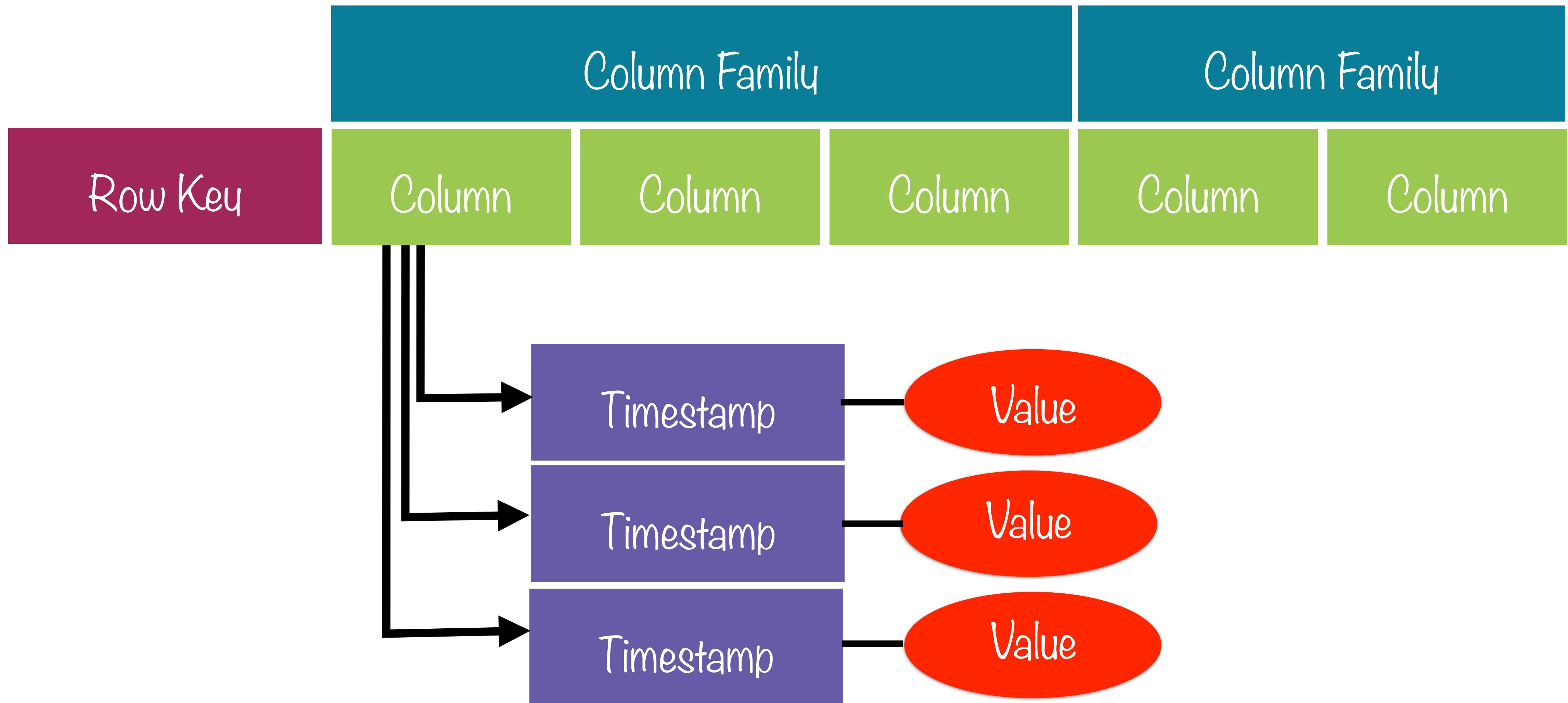
Id	To	Type	Content
1	mike	offer	Offer on mobiles
2	john	sale	Redmi sale
3	jill	order	Order delivered
4	megan	sale	Clothes sale



Id	Column	Value
1	To	mike
1	Type	offer
1	Content	Offer on mobiles
2	To	john
2	Type	sale
2	Content	Redmi sale
3	To	jill
3	Type	order
3	Content	Order delivered
4	To	megan
4	Type	sale
4	Content	Clothes sale

Value + Timestamp

Data Layout in HBase



Row Key

Uniquely identifies a row

Can be primitives, structures, arrays

Represented internally as a byte array

Sorted in ascending order

Column Family

- All rows have the same set of column families
- Each column family is stored in a separate data file
- Set up at schema definition time
- Can have different columns for each row

Column

Columns are units within a column family

New columns can be added on the fly

ColumnFamily:ColumnName = Work:Department

Timestamp

Used as the version number for the values stored in a column

The value for any version can be accessed

Census Data Layout in HBase

	Personal			Professional		
Some ID	name	gender	marital_status	employed	field	

Filtering Rows Based on Conditions

SQL vs. HBase Shell Commands

select * from census

SQL

HBase Shell

scan 'census'

SQL vs. HBase Shell Commands

select name from census

SQL

HBase Shell

```
scan 'census',  
{COLUMNS =>  
['personal:name']}
```

SQL vs. HBase Shell Commands

select * from census limit 1

SQL

HBase Shell

scan 'census', {LIMIT => 1}

SQL vs. HBase Shell Commands

select * from census where
rowkey = 1

SQL

HBase Shell

get 'census', 1



Filters allow you to
control what data is
returned from a scan
operation



Built-in Filters

Conditions on row keys

Conditions on columns

Multiple conditions on columns

Timestamp range

BigTable Performance

Avoid BigTable When

- Don't use if you need transaction support (OLTP) - use Cloud SQL or Cloud Spanner
- Don't use for data less than 1 TB (can't parallelize)
- Don't use if analytics/business intelligence/data warehousing - use BigQuery instead
- Don't use for documents or highly structured hierarchies - use DataStore instead
- Don't use for immutable blobs like movies each > 10 MB - use Cloud Storage instead

Use BigTable When

- Use for very fast scanning and high throughput
- Use for non-structured key/value data
- Where each data item < 10 MB and total data > 1 TB
- Use where writes infrequent/unimportant (no ACID) but fast scans crucial
- Use for Time Series data

Use BigTable for Time Series

- BigTable is a natural fit for Timestamp data (range queries)
- Say IOT sensor network emitting data at intervals
 - Use **Device ID # Time** as row key if common query = “All data for a device over period of time”
 - Use **Time # Device ID** as row key if common query = “All data for a period for all devices”

Hotspotting and Schema Design

- Like Cloud Spanner, data stored in sorted lex order of keys
- Data is distributed based on key values
- So, performance will be really poor if
 - Reads/writes are concentrated in some ranges
 - For instance if key values are sequential
 - Use hashing of key values, or non-sequential keys

Avoiding Hotspotting

- Field Promotion: Use in reverse URL order like Java package names
 - This way keys have similar prefixes, differing endings
- Salting
 - Hash the key value

“Warming the Cache”

- BigTable will improve performance over time
- Will observe read and write patterns and redistribute data so that shards are evenly hit
- Will try to store roughly same amount of data in different nodes
- This is why testing over hours is important to get true sense of performance

SSD or HDD Disks

- Use SSD unless skimping on cost
- SSD can be 20x faster on individual row reads
- More predictable throughput too (no disk seek variance)
- Don't even think about HDD unless storing > 10 TB and all batch queries
- The more random access, the stronger the case for SSD

SSD or HDD Disks

Storage Type	Reads	Writes	Scans
SSD	10,000 QPS ¹ @ 6 ms	10,000 QPS @ 6 ms	220 MB/s
HDD	500 QPS @ 200 ms	10,000 QPS @ 50 ms	180 MB/s

¹ Queries per second. A query is a read or write operation against a single row.

Reasons for Poor Performance

- Poor schema design (eg sequential keys)
- Inappropriate workload
 - too small (<300 GB)
 - used in short bursts (needs hours to tune performance internally)
- Cluster too small
- Cluster just fired up or scaled up
- HDD used instead of SSD
- Development v Production instance

Schema Design

- Each table has just one index - the row key. Choose it well
- Rows are sorted lexicographically by row key
- All operations are atomic at row level
- Related entities in adjacent rows

Size Limits

- Row keys: 4KB per key
- Column Families: ~100 per table
- Column Values: ~ 10 MB each
- Total Row Size: ~100 MB

Types of Row Keys

- Reverse domain names
- String identifiers
- Timestamps as suffix in key

Row Keys to Avoid

- Domain names
- Sequential numeric values
- Timestamps alone
- Timestamps as prefix of row-key
- Mutable or repeatedly updated values

Datastore

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

Use-Cases

When you need

Storage for Compute, Block Storage

Storing media, Blob Storage

SQL Interface atop file data

Document database, NoSQL

Fast scanning, NoSQL

Transaction Processing (OLTP)

Analytics/Data Warehouse (OLAP)

Use

Persistent (hard disks), SSD

Cloud Storage

BigQuery

DataStore

BigTable

Cloud SQL, Cloud Spanner

BigQuery

DataStore

- Document data - eg XML or HTML - has a characteristic pattern
- Key-value structure, i.e. structured data
- Typically not used either for OLTP or OLAP
- Fast lookup on keys is the most common use-case

DataStore

- Speciality of DataStore is that query execution time depends on size of returned result (not size of data set)
- So, a returning 10 rows will take the same length of time whether dataset is 10 rows, or 10 billion rows
- Ideal for “needle-in-a-haystack” type applications, i.e. lookups of non-sequential keys

Traditional RDBMS vs. DataStore

Traditional RDBMS

Atomic transactions

Indices for fast lookup

Some queries use indices - not all

Query time depend on both size of data set and size of result set

DataStore

Atomic transactions

Indices for fast lookup

All queries use indices!

Query time independent of data set, depends on result set alone

Traditional RDBMS vs. DataStore

Traditional RDBMS

Structured relational data

Rows stored in Tables

Rows consist of fields

Primary Keys for unique ID

DataStore

Structured hierarchical data (XML, HTML)

Entities of different kinds (think HTML tags)

Entities consist of Properties

Keys for unique ID

Traditional RDBMS vs. DataStore

Traditional RDBMS

Rows of table have same properties (Schema is strongly enforced)

Types of all values in a column are the same

DataStore

Entities of a kind can have different properties (think optional tags in HTML)

Types of different properties with same name in an entity can be different

Traditional RDBMS vs. DataStore

Traditional RDBMS

Lots of joins

Filtering on subqueries

Multiple inequality conditions

DataStore

No joins

No filtering on subqueries

Only one inequality filter OK per query

Avoid DataStore When

- Don't use if you need **very strong transaction support** (OLTP) - OK for basic ACID support though
- Don't use for non-hierarchical or unstructured data - BigTable is better
- Don't use if analytics/business intelligence/data warehousing - use BigQuery instead
- Don't use for immutable blobs like movies each > 10 MB - use Cloud Storage instead
- Don't use if application has lots of writes and updates on key columns

Use DataStore When

- Use for crazy scaling of read performance - to virtually any size
- Use for hierarchical documents with key/value data

Full Indexing

- “Built-in” Indices on each property (~field) of each entity kind (~table row)
- “Composite” Indices on multiple property values
- If you are certain a property will never be queried, can explicitly exclude it from indexing
- Each query is evaluated using its “perfect index”

Perfect Index

- Given a query, which is the index that most optimally returns query results?
- Depends on following (in order)
 - equality filter
 - inequality filter (only 1 allowed)
 - sort conditions if any specified

Implications of Full Indexing

- Updates are really slow
- No joins possible
- Can't filter results based on subquery results
- Can't include more than one inequality filter (one is OK)

Multi- tenancy

- Separate data partitions for each client organization
- Can use the same schema for all clients, but vary the values
- Specified via a namespace (inside which kinds and entities can exist)

Transaction Support

- Can optionally use transactions - not required
- Not as strong as Cloud Spanner (which is ACID++), but stronger than BigQuery or BigTable

Consistency

- Two consistency levels possible for query results
 - Strongly consistent: return up-to-date result, however long it takes
 - Eventually consistent: faster, but might return stale

Transfer Service

Importing Data

- The transfer service helps get data **into Cloud Storage**
- From where?
 - From AWS, i.e. an S3 bucket
 - From HTTP/HTTPS location
 - From local files
 - From another Cloud Storage Bucket

gsutil or Transfer Service?

- Recall that gsutil can be used to get data into cloud storage buckets
- Prefer the transfer service when transferring from AWS etc
- If copying files over from on-premise, use gsutil

Transfer Service Bells & Whistles

- One-time vs recurring transfers
- Delete from destination if they don't exist in source
- Delete from source after copying over
- Periodic synchronisation of source and destination based on file filters

Summary

Block storage for compute VMs - persistent disks or SSDs

Immutable blobs like video/images - Cloud Storage

OLTP - Cloud SQL or Cloud Spanner

NoSQL Documents like HTML/XML - Datastore

NoSQL Key-values - BigTable (~HBase)

Getting data into Cloud Storage - Transfer service