Cloud Computing
Container & Kubernetes

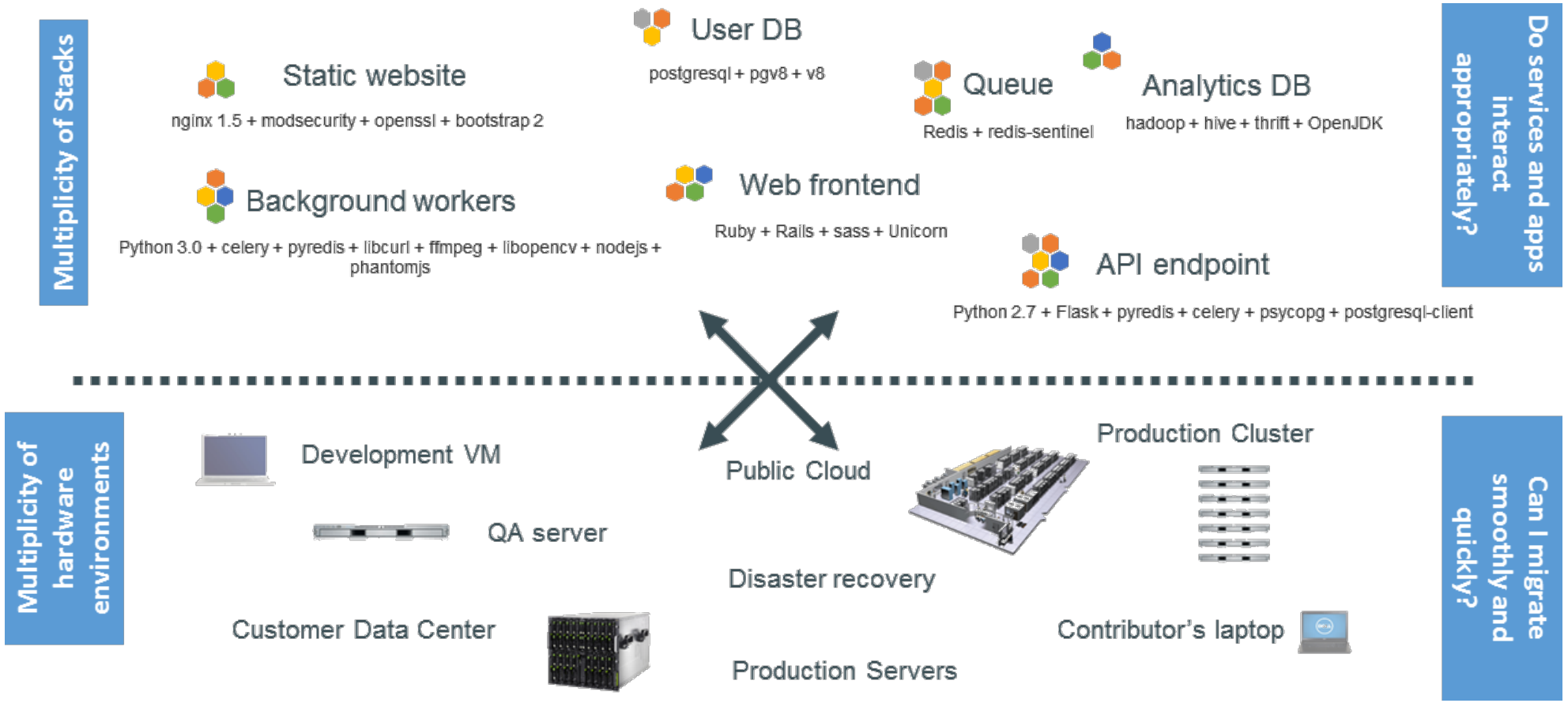
Minchen Yu
SDS@CUHK-SZ
Fall 2024

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

Why containers?

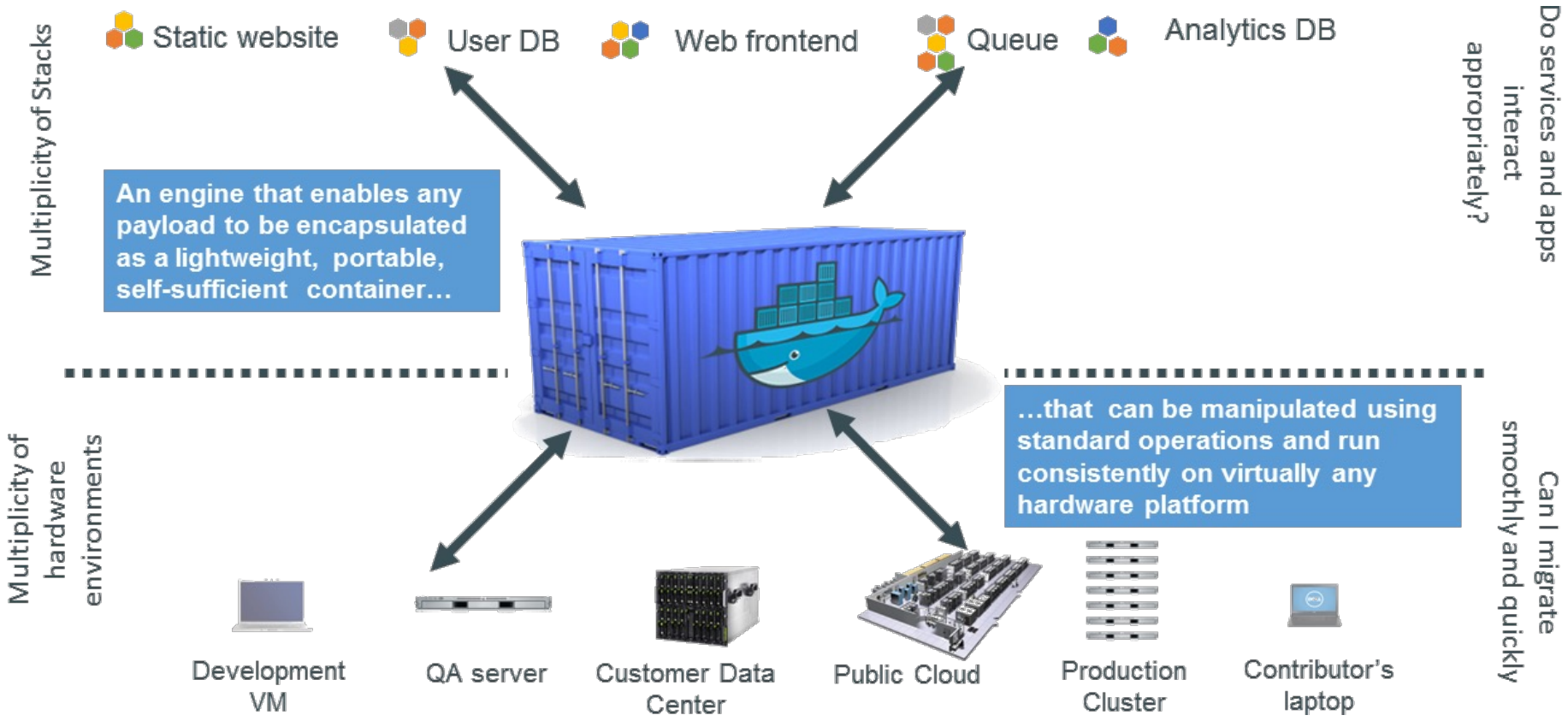
The challenge



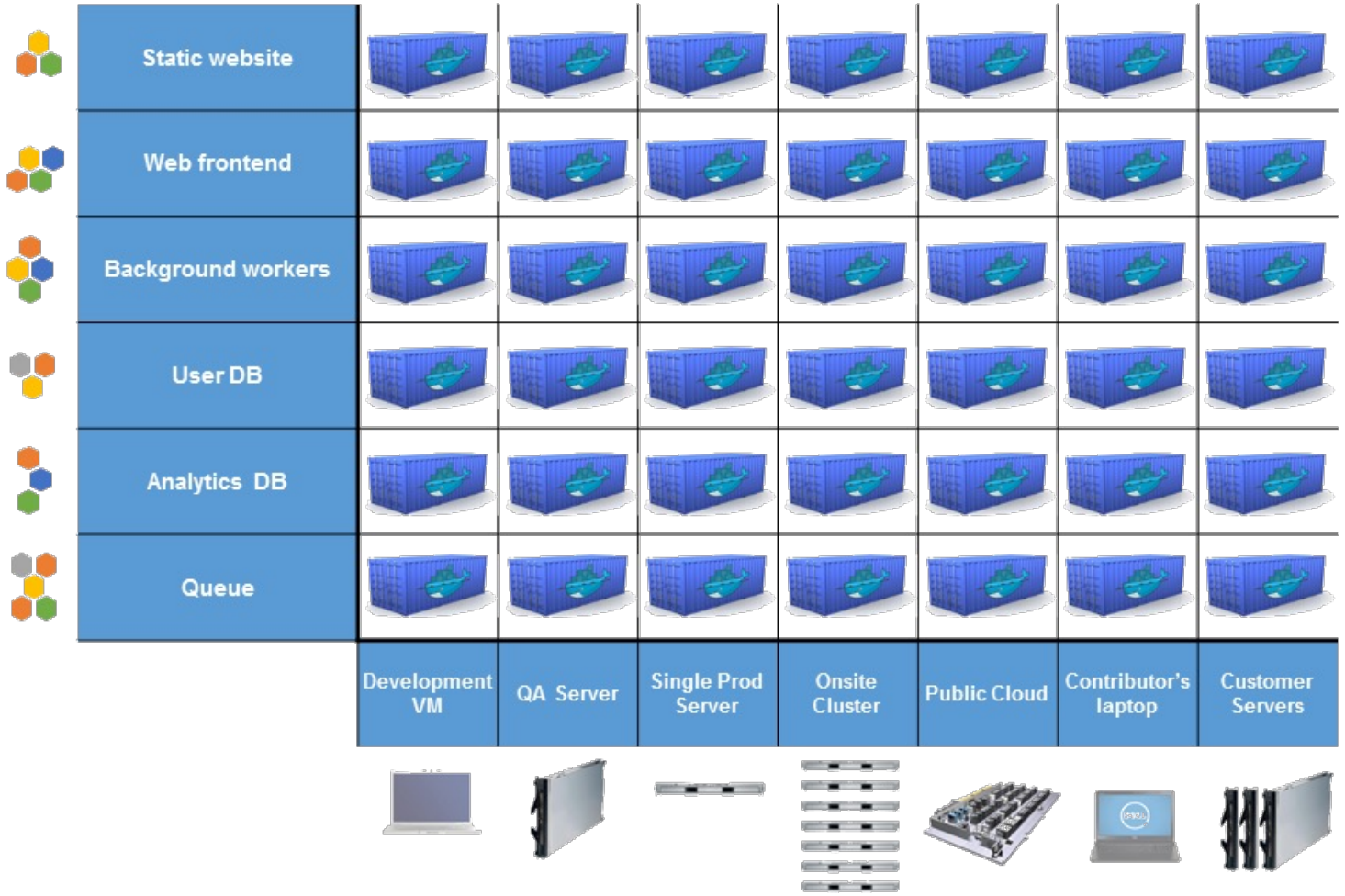
The Matrix from hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

A container system for code

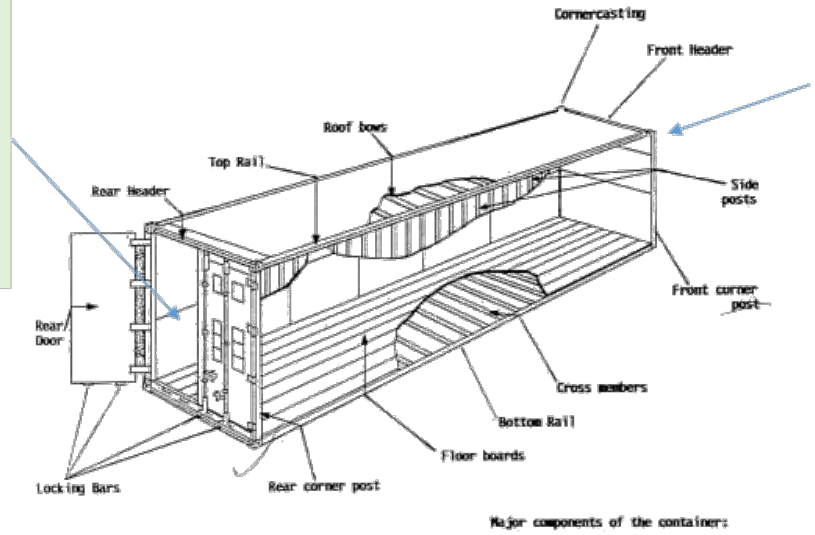


Eliminate the matrix from hell



Separation of concerns

- **Dan the Developer**
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same

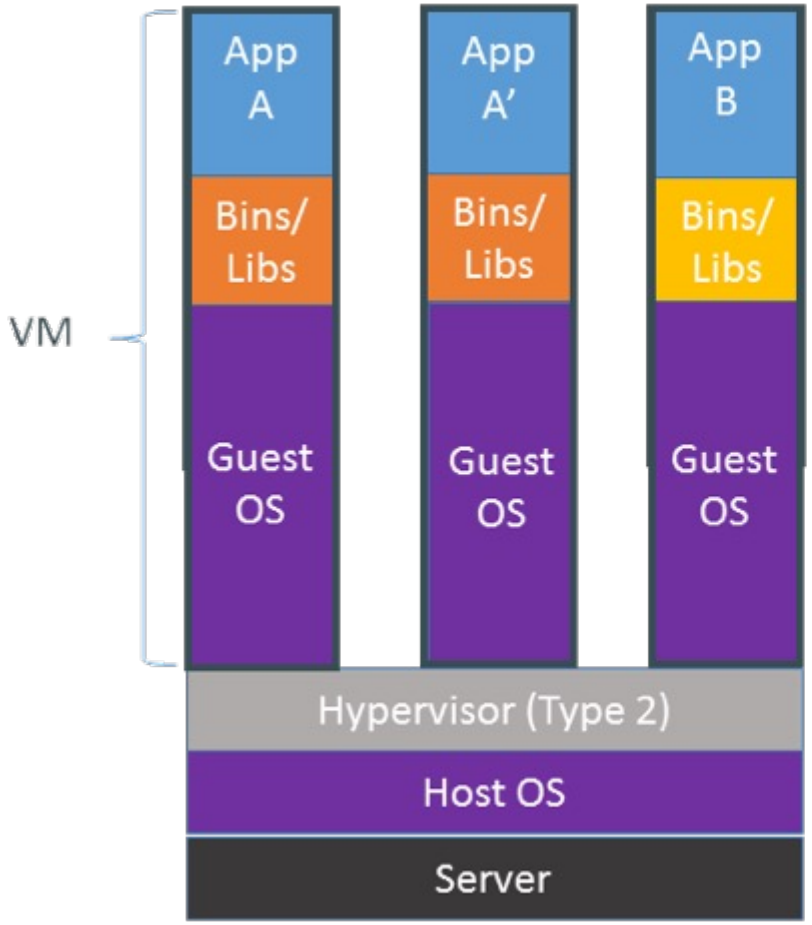


Major components of the container

- **Oscar the Ops Guy**
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

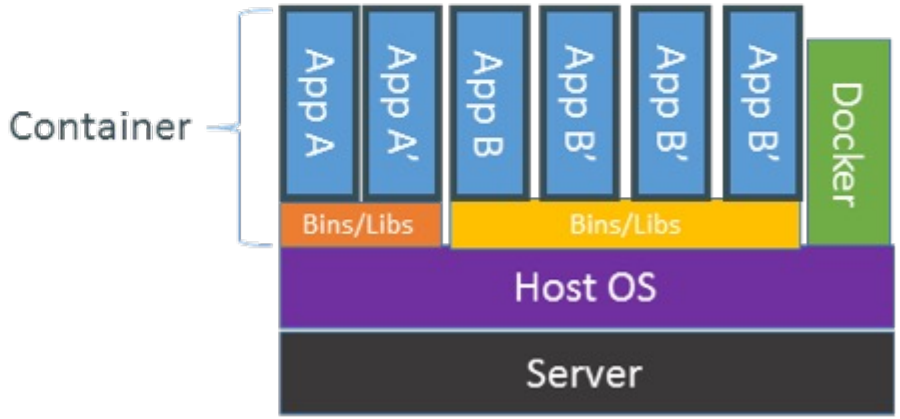
Configure once, run anything anywhere

VM vs. Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Container implementation

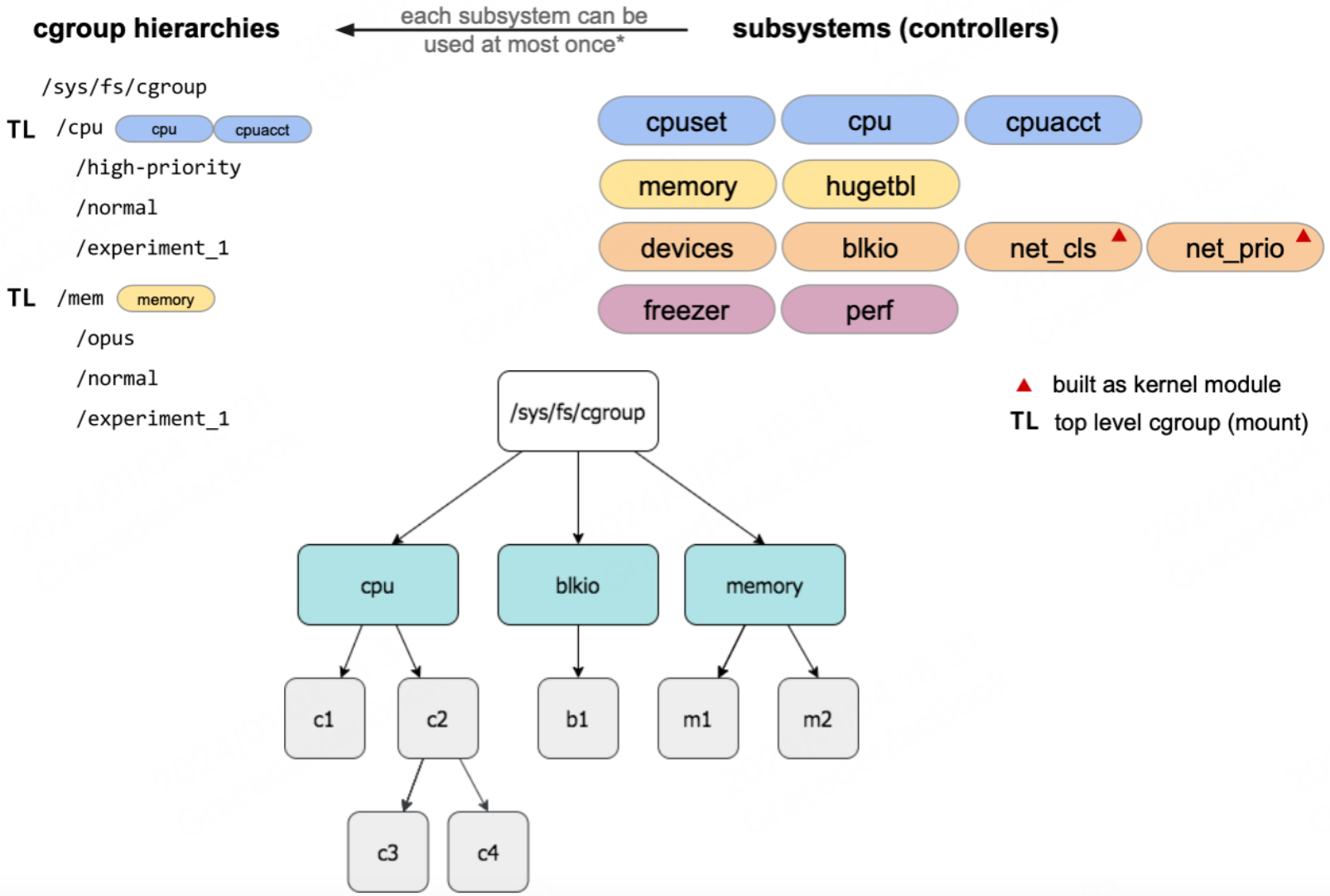
Leveraging Linux kernel mechanisms

- **cgroups:** manage resources for groups of processes
- **namespaces:** per process resource isolation
- seccomp: limit available system calls
- capabilities: limit available privileges
- CRIU: checkpoint/restore (w/ kernel support)

cgroups: Linux control groups

- control group subsystem offering a resource management solution for a group of processes
- Each subsystem has a ***hierarchy*** **(a tree)**
    - separate hierarchies for CPU, memory, block I/O
    - each process is in a node in each hierarchy
    - each node = a group of processes sharing the same resources

cgroup hierarchies



namespaces

Limit the scope of kernel-side names and data structures at process granularity

mnt	(mount points, filesystems)	*CLONE_NEWNS*
pid	(processes)	*CLONE_NEWPID*
net	(network stack)	*CLONE_NEWNET*
ipc	(System V IPC)	*CLONE_NEWIPC*
uts	(unix timesharing - domain name, etc)	*CLONE_NEWUTS*
user	(UIDs)	*CLONE_NEWUSER*

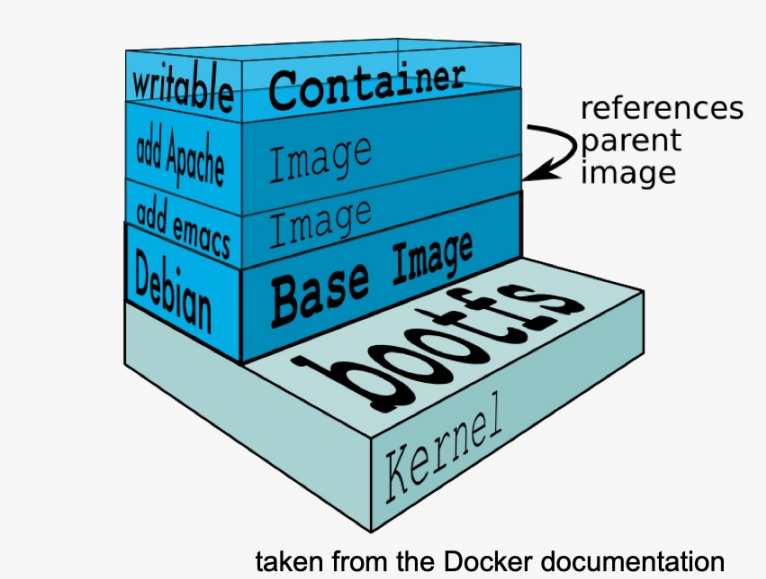
Three system calls for management

`clone()`	new process, new namespace, attach process to ns
`unshare()`	new namespace, attach current process to it
`setns(int fd, int nstype)`	join an existing namespace

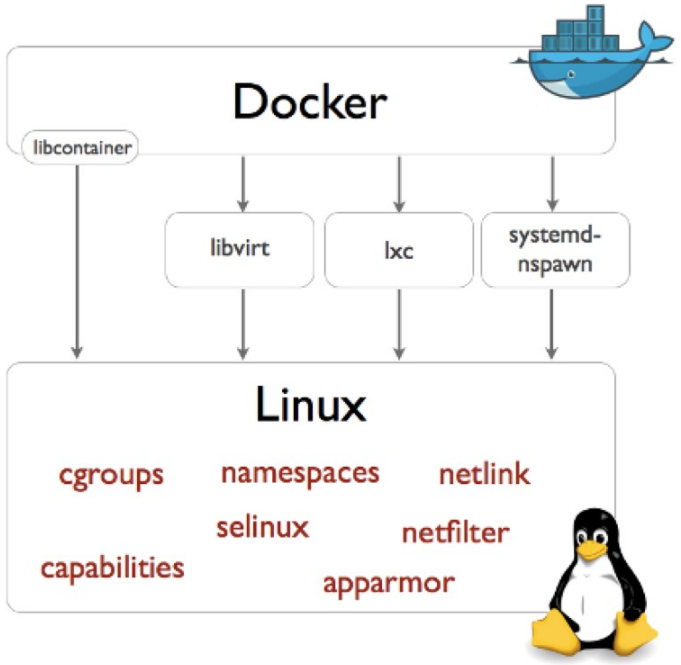
Containers

A light form of resource virtualization based on kernel mechanisms like cgroups and namespaces

Multiple containers run on the same kernel with the illusion that they are the only one using resources



taken from the Docker documentation



Docker

Provides container runtime and the isolation among containers

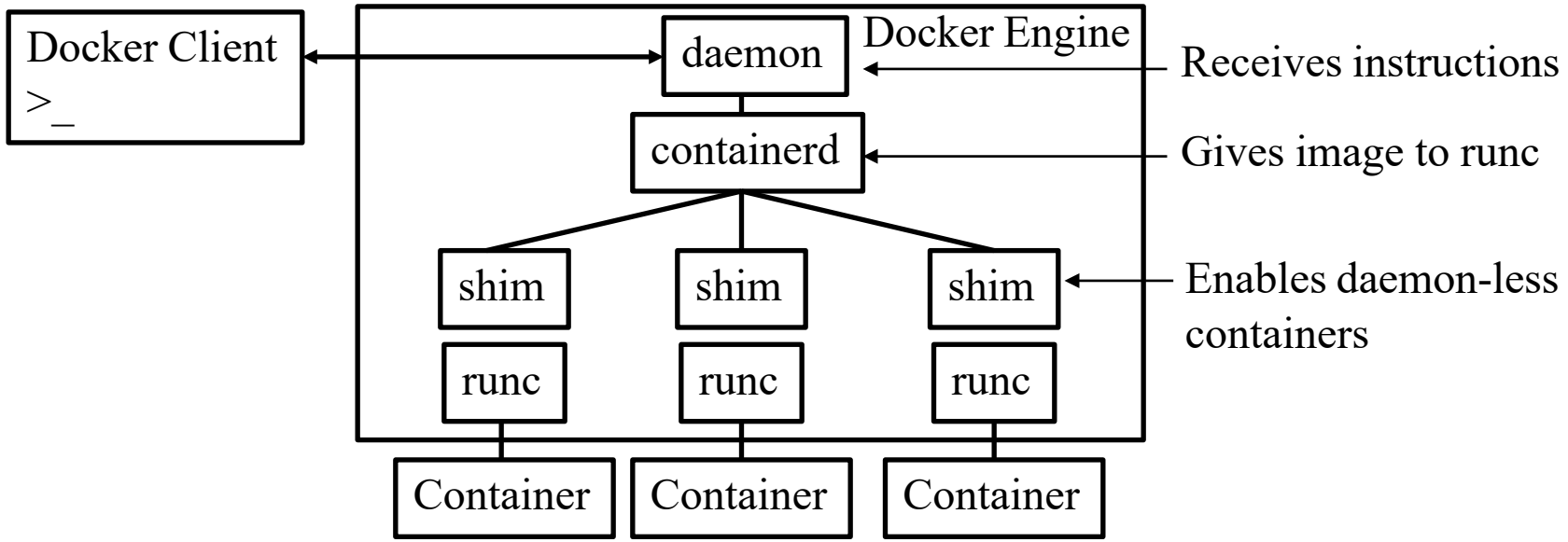
Helps them share the OS

Developed initially by Docker.com

Downloadable for Linux, Windows, and Mac

Docker

- **daemon**: API and other features
- **containerd**: Execution logic. Responsible for container lifecycle. Start, stop, pause, unpause, delete containers
- **runc**: A lightweight runtime CLI
- **shim**: runc exists after creating the container. shim keeps the container running. Keep stdin/stdout open

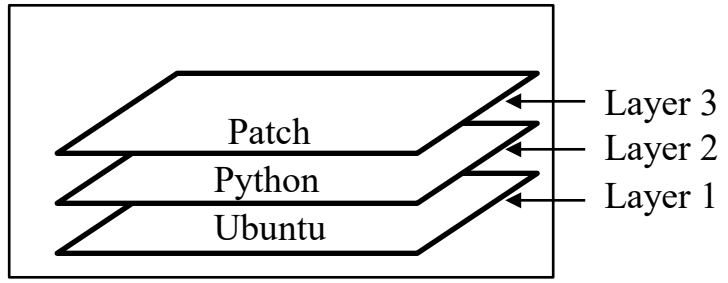


Docker images

- Containers are built from images and can be saves as images
- Images are stored in registries
  - Local registry on the same host
  - Docker Hub Registry: Globally shared
  - Private registry on Docker.com
- Any component not found in the local registry is downloaded from specified location
- Each image has several tags, e.g., v2, latest, …
- Each image is identified by its 256-bit hash

Layers

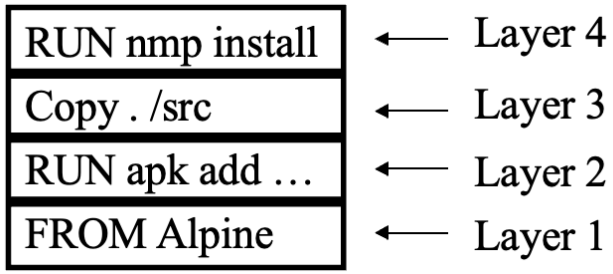
- Each image has many layers
- Image is built layer by layer
- Layers in an image can be inspected by Docker commands
- Each layer has its own 256-bit hash
- For example:
  - Ubuntu OS is installed, then
  - Python package is installed, then
  - a security patch to the Python is installed
- Layers can be shared among many containers



Build docker images

Create a Dockerfile that describes the application, its dependencies, and how to run it

```
FROM Alpine                                  ← Start with Alpine Linux
LABEL maintainer="xx@gmail.com"              ← Who wrote this container
RUN apk add –update nodejs nodejs –npm       ← Use apk package to install nodejs
COPY . /src                                  ← Copy the app files from build context
WORKDIR /src                                 ← Set working directory
RUN nmp install                              ← Install application dependencies
EXPOSE 8080                                  ← Open TCP Port 8080
ENTRYPOINT ["node", "./app.js"]              ← Main application to run
```

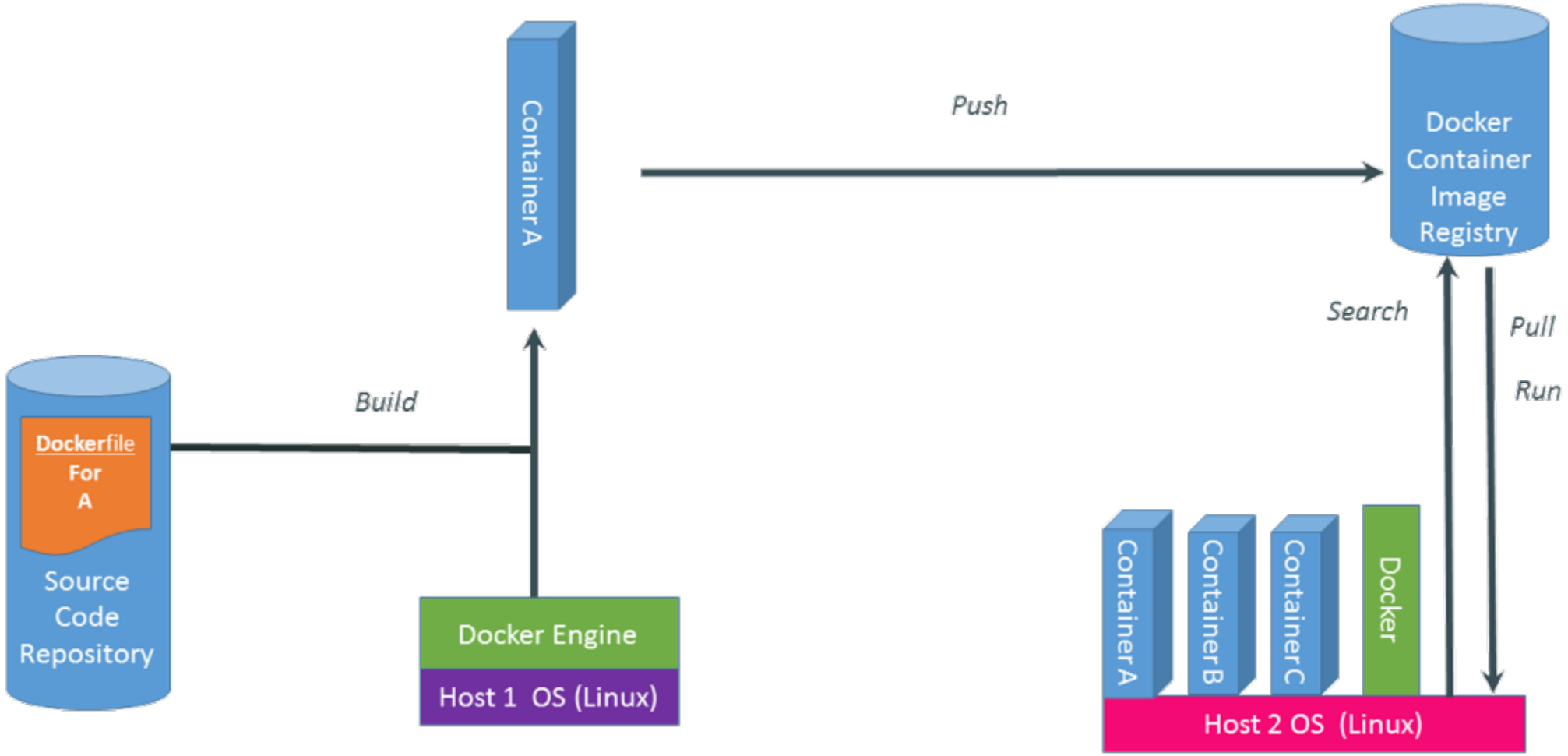


Note: WORKDIR, EXPOSE, ENTRYPOINT result in tags. Others in Layers

Docker commands

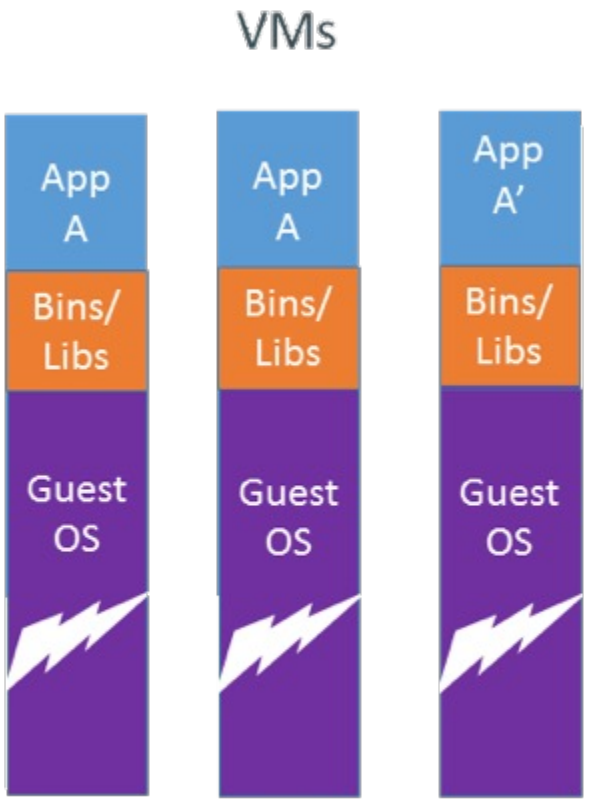
- **docker container run**: Run the specified image
- **docker container ls**: list running containers
- **docker container exec**: run a new process inside a container
- **docker container stop**: Stop a container
- **docker container start**: Start a stopped container
- **docker container rm**: Delete a container
- **docker container inspect**: Show information about a container

Overview of a Docker system



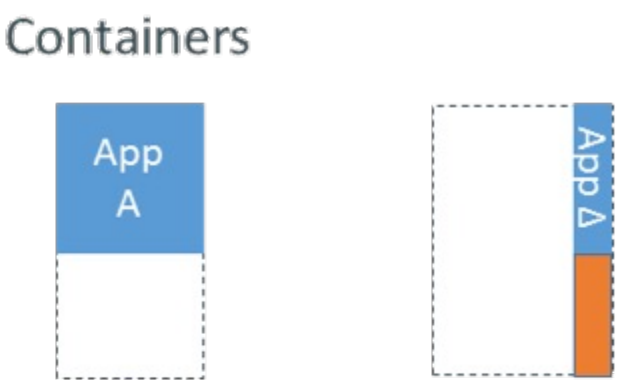
Configure once, run anything anywhere

Changes and updates



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server



Original App

(No OS to take up space, resources, or require restart)

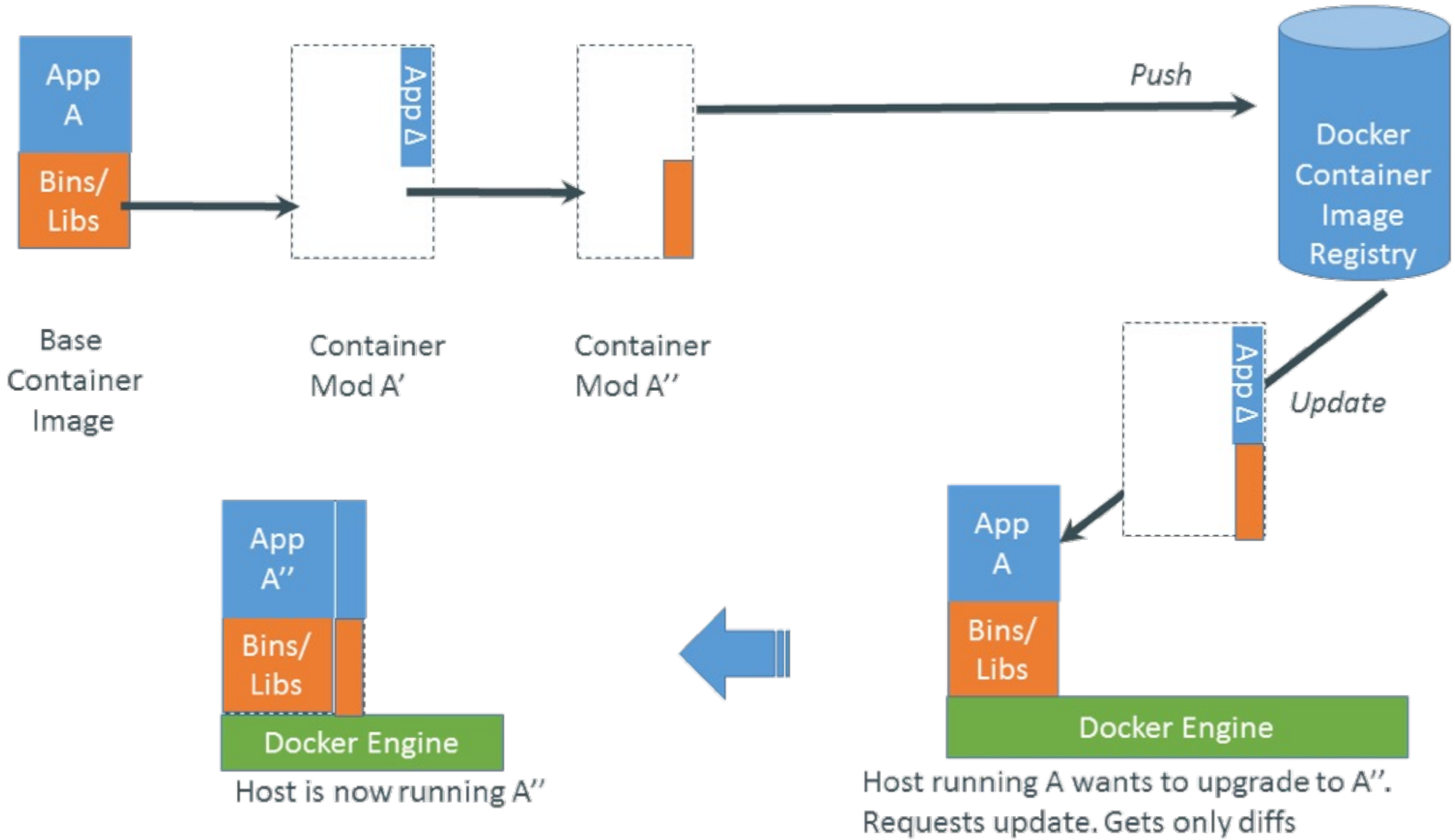
Copy of App

No OS. Can Share bins/libs

Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

Changes and updates



How do I deploy containers (at scale)?

Deployment options

Understanding **how** containers work is not the same as using them in production

Different organizations will have very different needs

- Not everyone needs to operate “at scale”

There are many ways to run and manage containers

Humans

Never underestimate the value of manual solutions

SSH into machines and run docker

- Pro: simple, available everywhere, no special tools needed, easily understood
- Con: not automated, not reproducible (human make mistakes), doesn't scale, doesn't self-heal

Scripts

A very common first step into containers

Puppet, Chef, Ansible, Salt, or just bespoke scripts

- Pro: integrates with existing environments, easily understood results, reproducible
- Con: manual scheduling, doesn't self-heal, doesn't scale, generally non-portable

Container orchestration

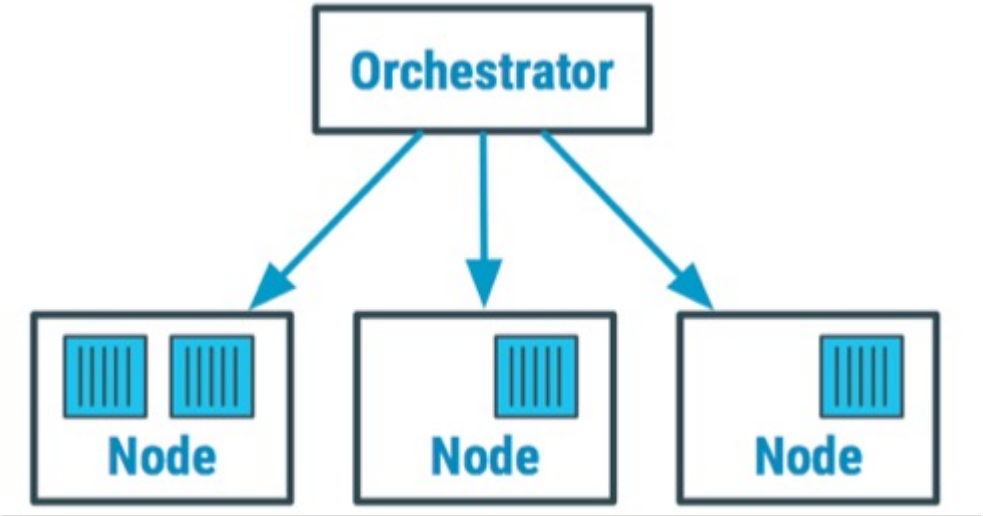
Need for something more?

- Docker started out with a CLI tool on top of Linux containers, that built, created, started, stopped and exec'd containers
- Does management at a node level, upon specific requests
- Easy to manually manage with up to 100s of containers and 10s of nodes, but what's next?

Orchestrator

Manage and organize both hosts and docker containers running on a cluster

Main Issue - **resource allocation** - where can a container be scheduled, to fulfill its requirements (CPU/RAM/disk) + how to keep track of nodes and scale



Some orchestrator tasks

Manage networking and access

Track state of containers

Scale services

Do load balancing

Relocation in case of unresponsive host

Service discovery

Attribute storage to containers

…

Kubernetes

What is Kubernetes?

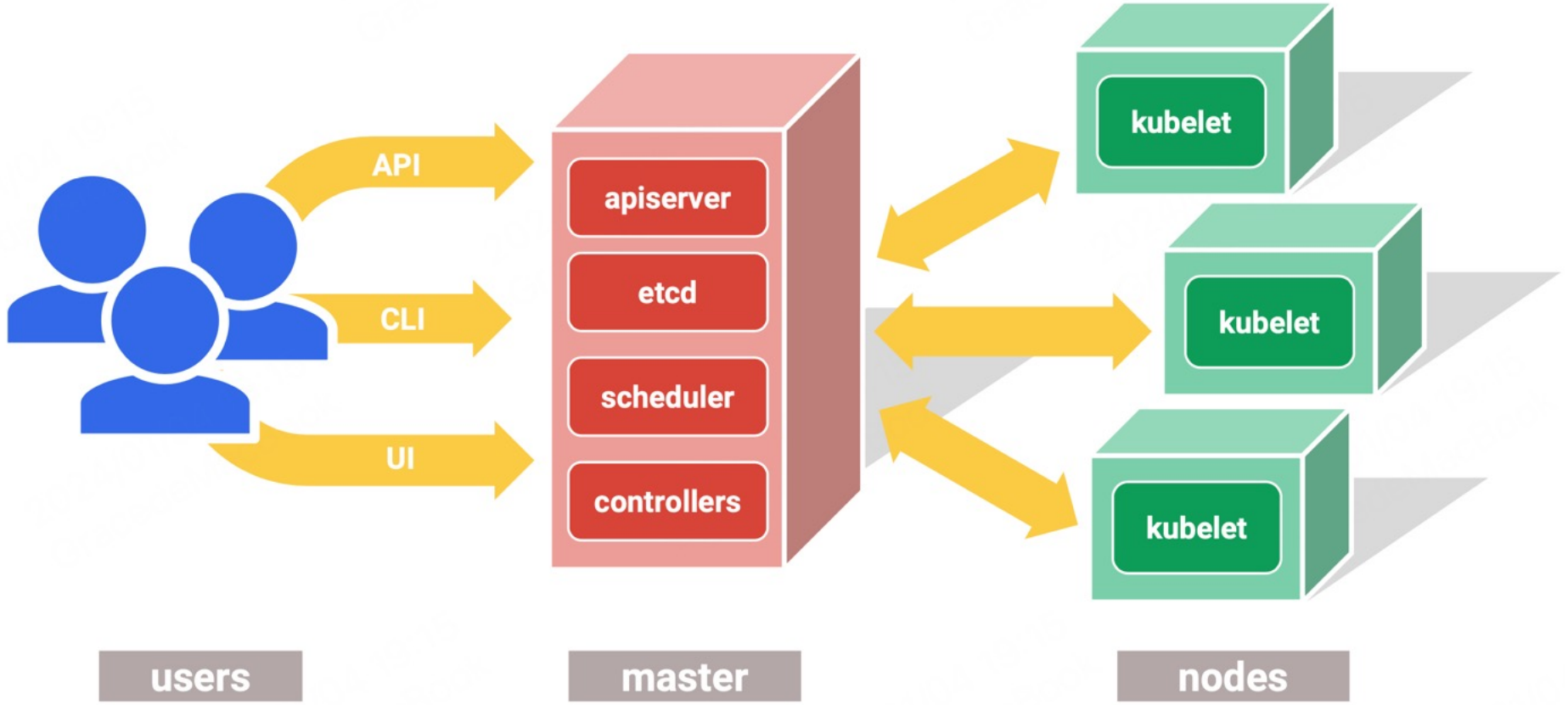
= A Production-Grade Container Orchestration System

Google-grown, based on **Borg** and **Omega**, systems that run inside of Google right now and are proven to work at Google for over 10 years.

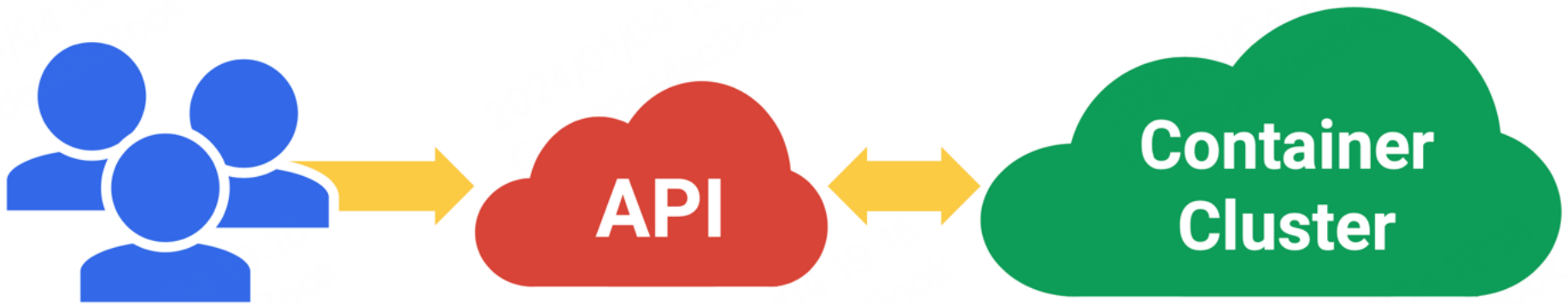
Google spawns billions of containers per week with these systems.

Created by three Google employees initially during the summer of 2014; grew exponentially and became the first project to get denoted to CNCF.

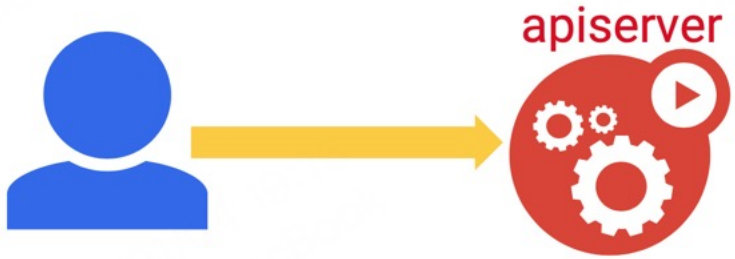
The 10000 foot view



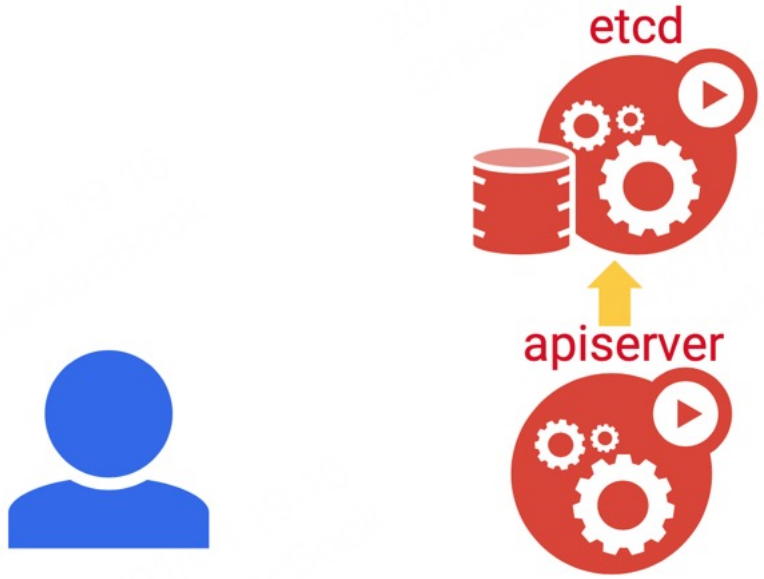
All you really care about



Running a container

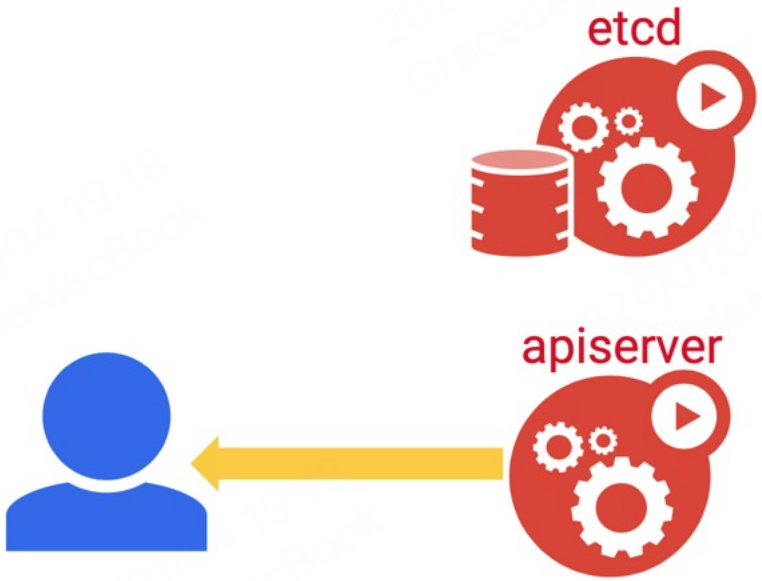


Running a container

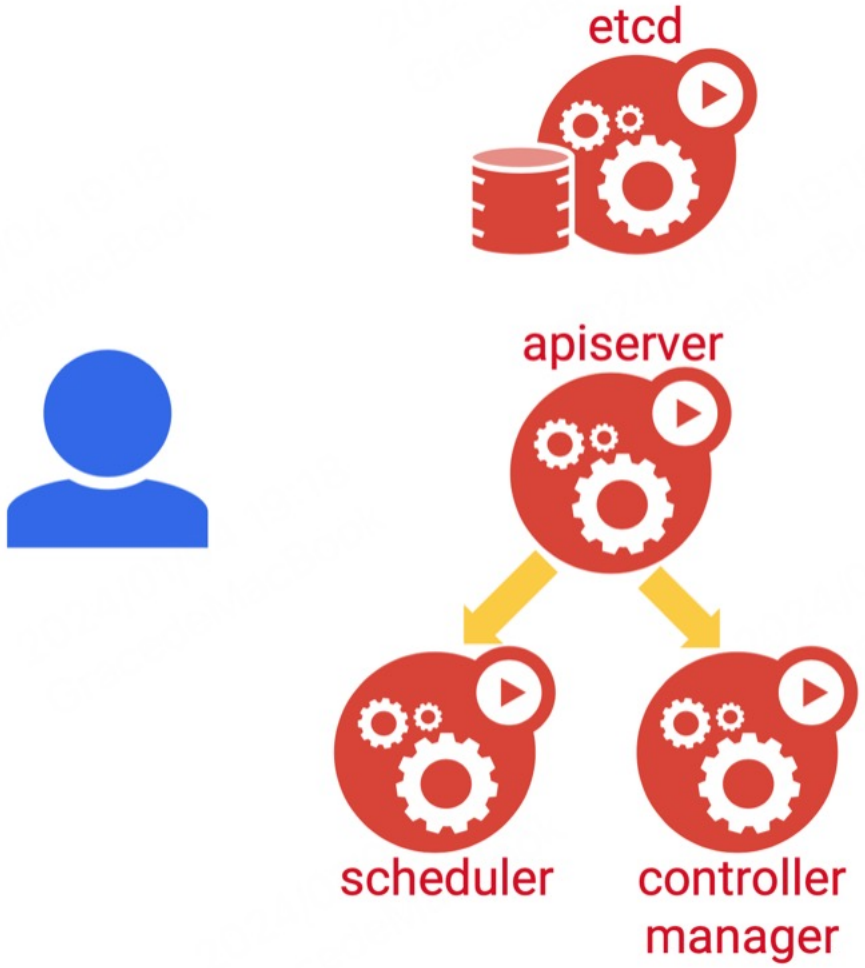


etcd: a consistent and highly-available key value store used as K8s' backing store for all cluster data

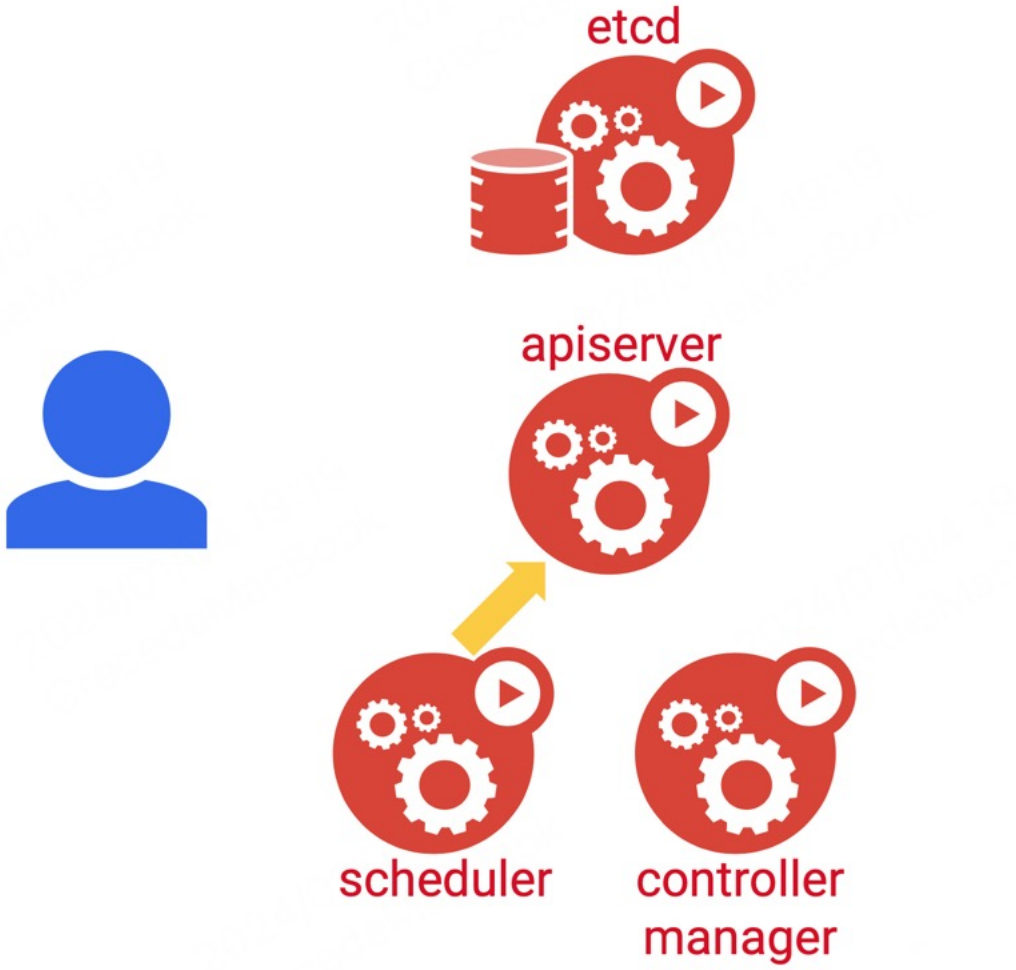
Running a container



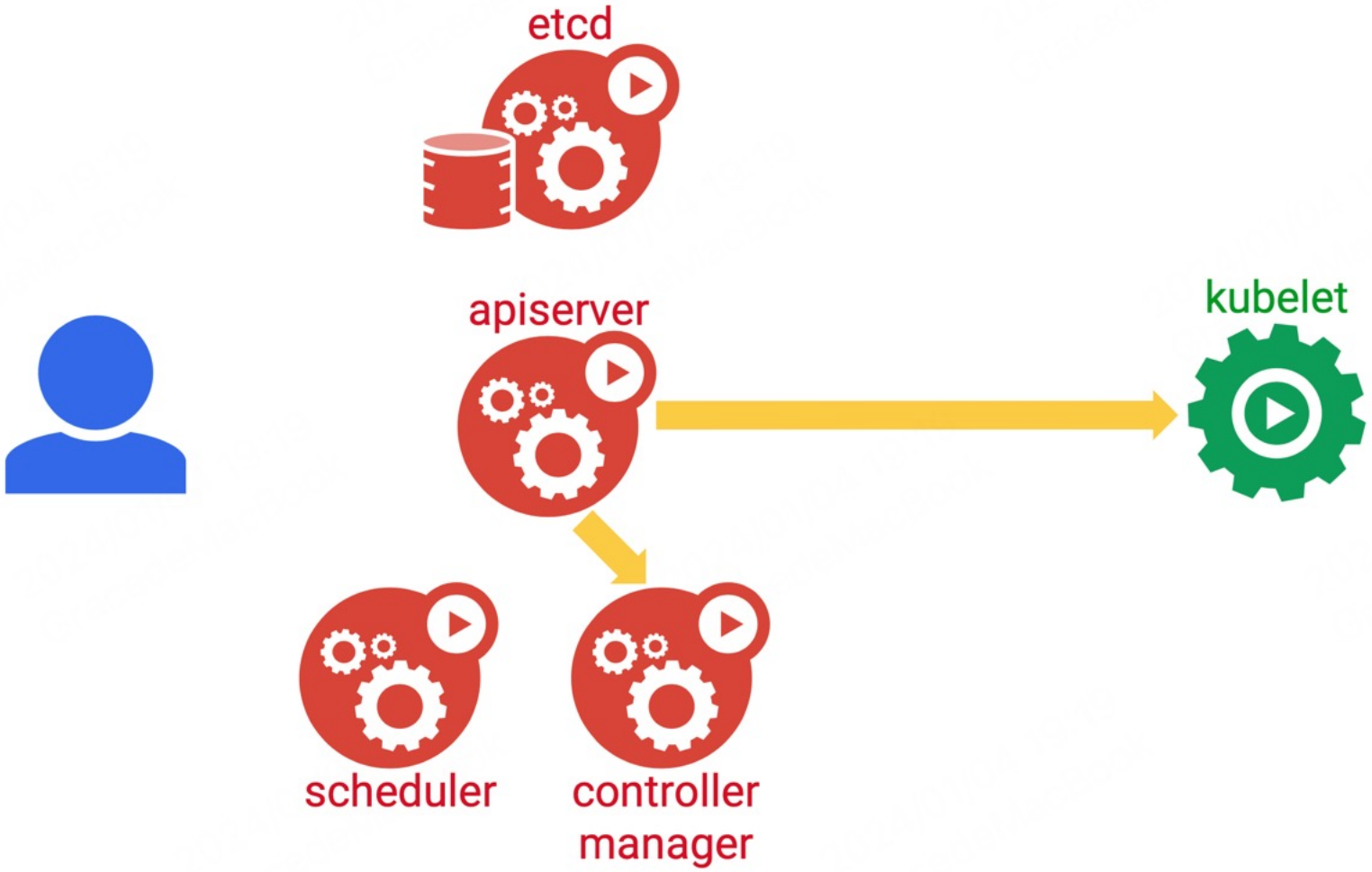
Running a container



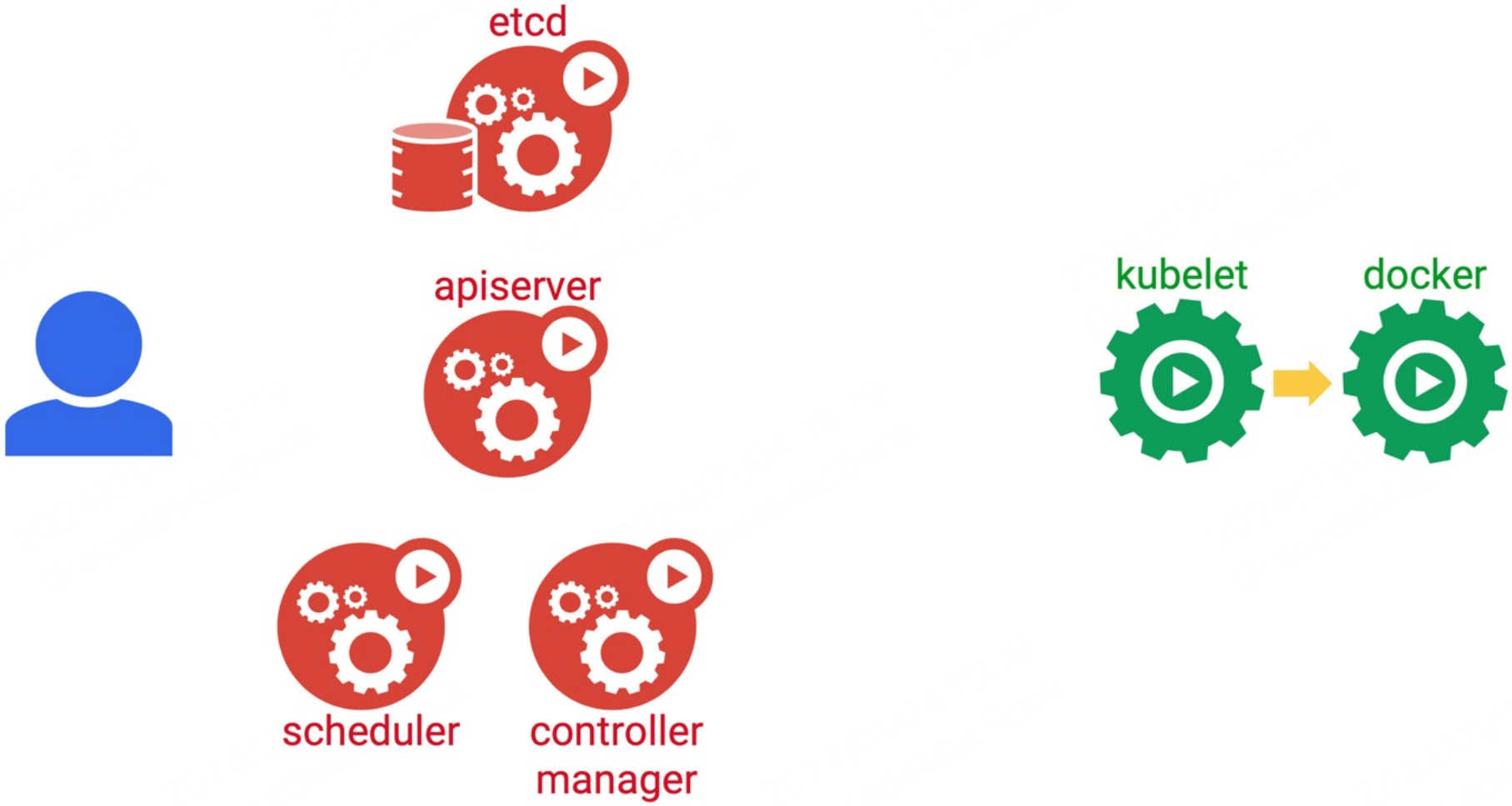
Running a container



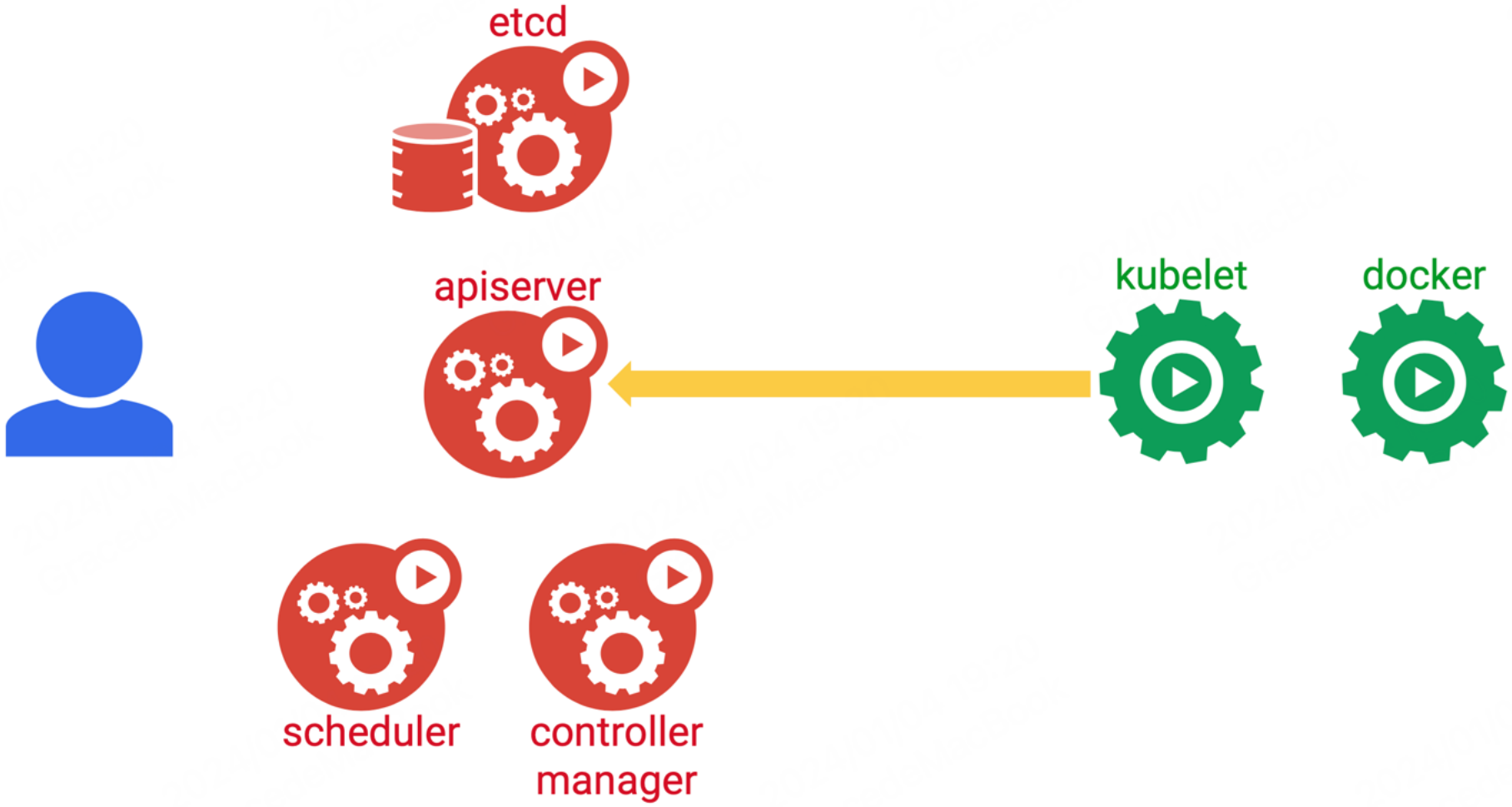
Running a container



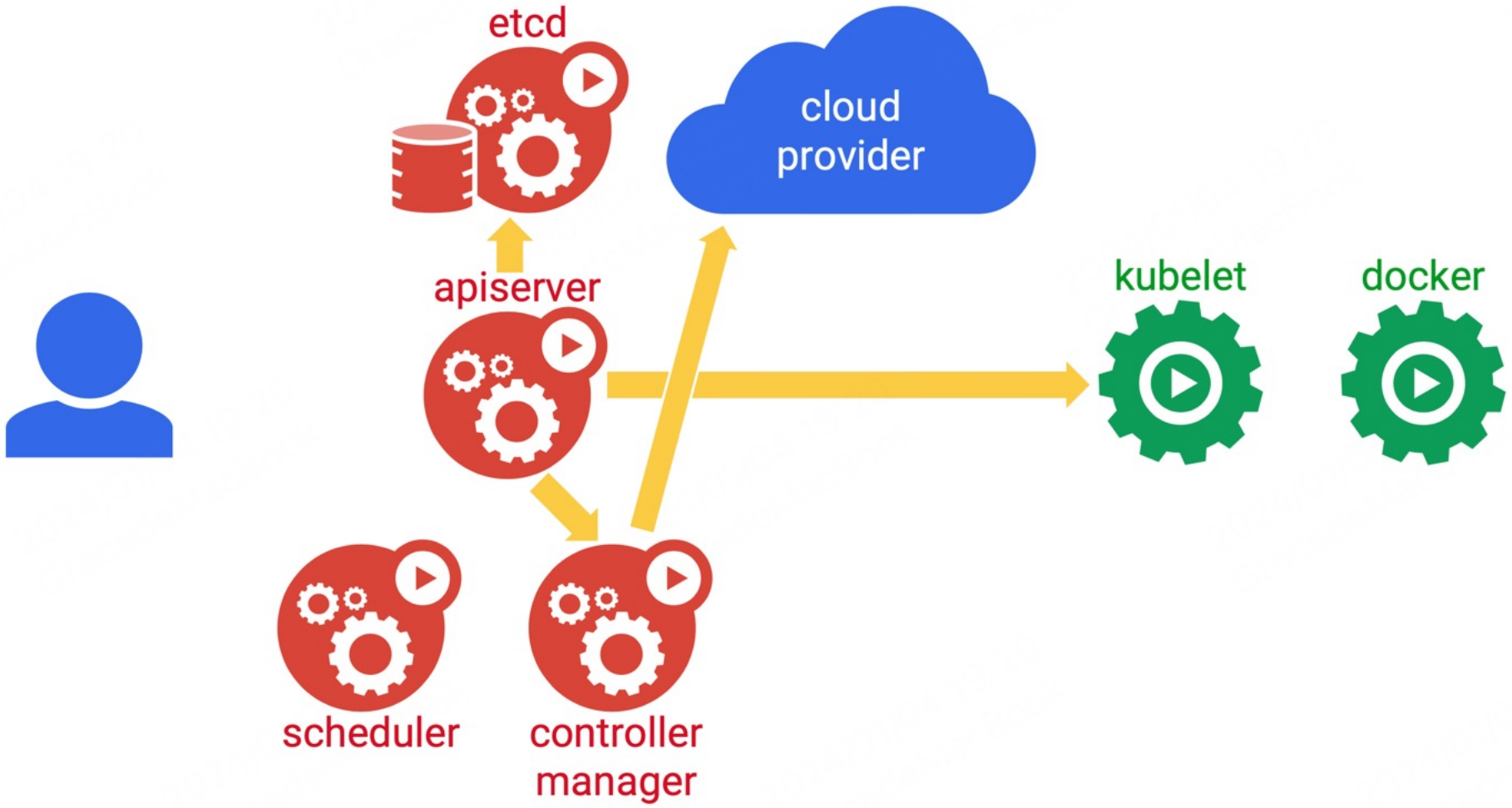
Running a container



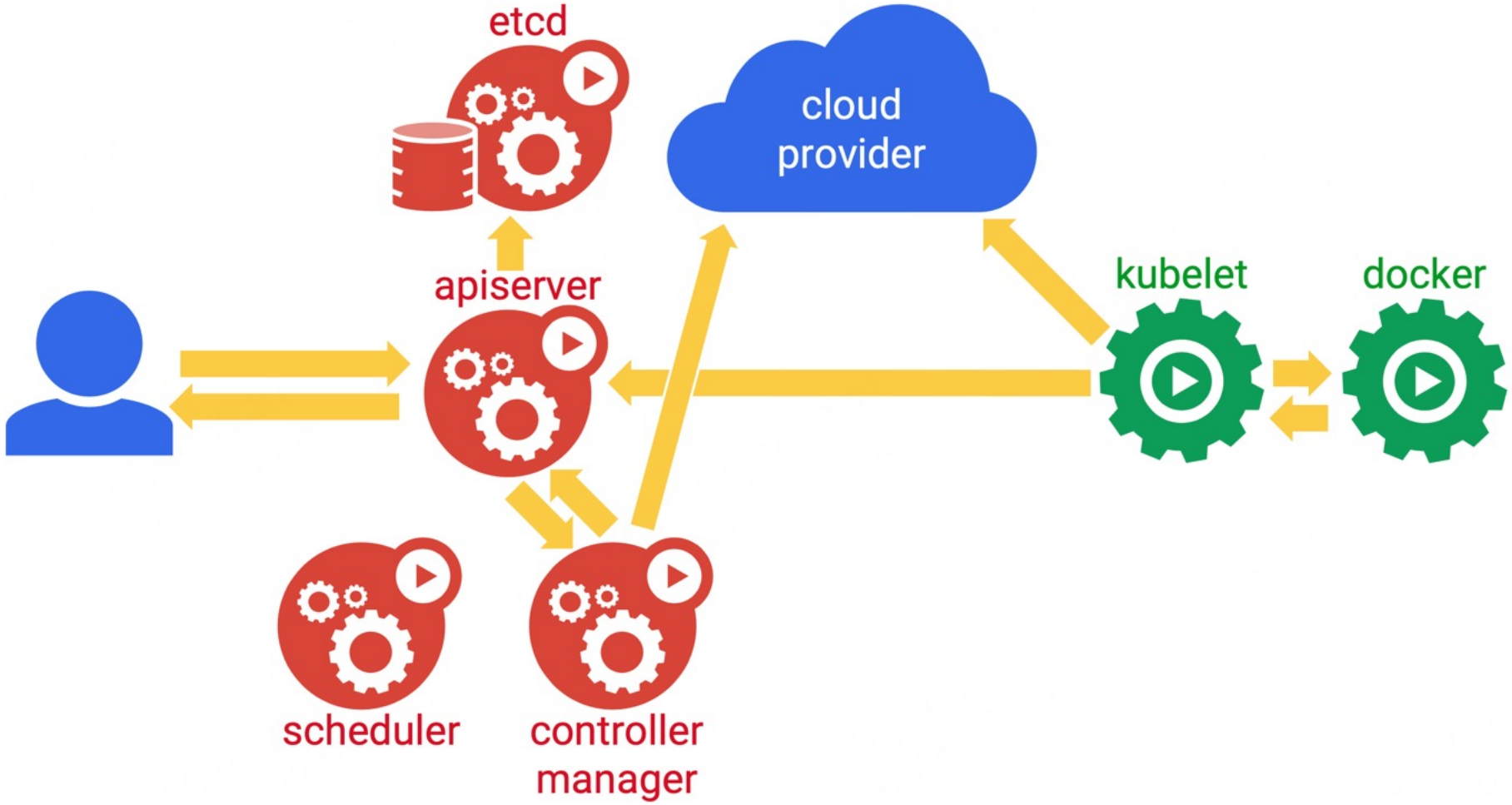
Running a container



Running a container



Running a container



Kubernetes and containers

Can we deploy a container in Kubernetes?

Kubernetes and containers

Can we deploy a container in Kubernetes?

- NO (not directly)

Kubernetes and containers

Can we deploy a container in Kubernetes?

- NO (not directly)

Why not?

Kubernetes and containers

Can we deploy a container in Kubernetes?

- NO (not directly)

Why not?

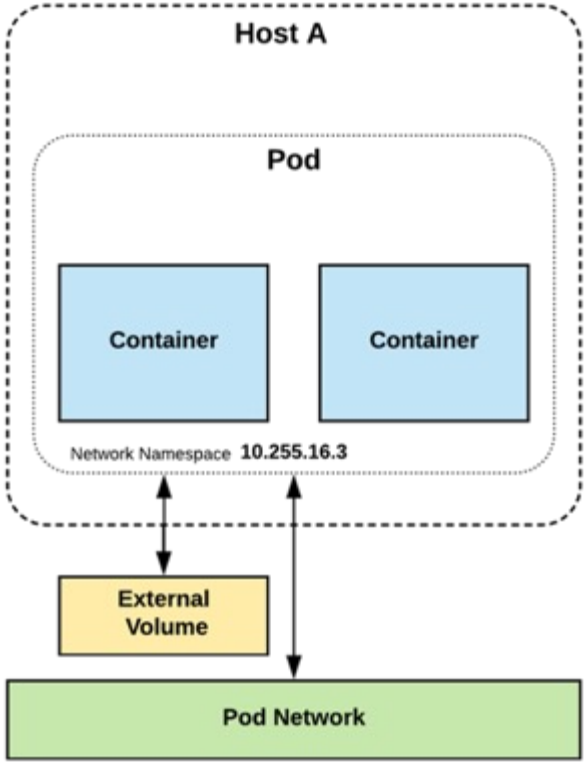
- Because the smallest deployable unit of computing is not a container, but a **pod**

Pods

Atomic unit or smallest “unit of work” of Kubernetes

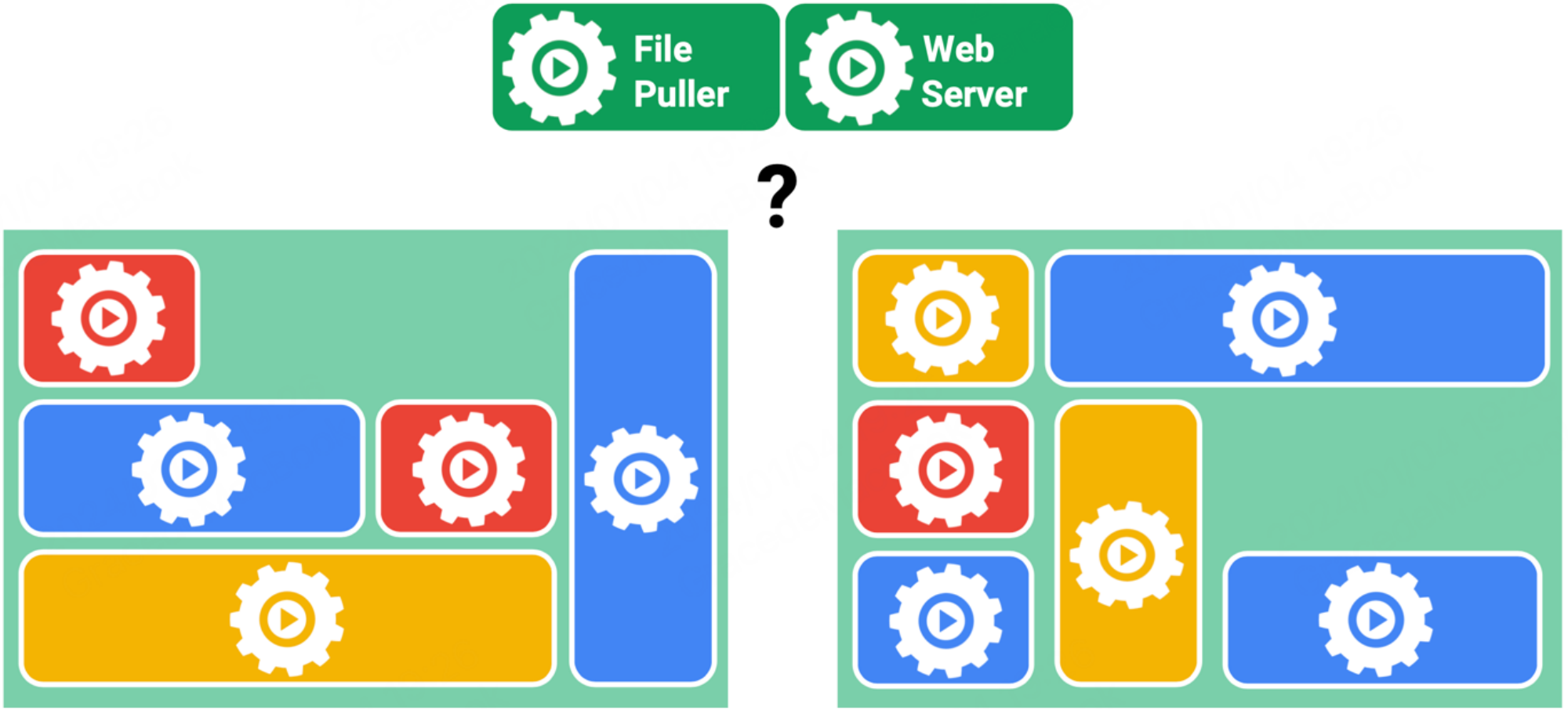
Pods are **one or MORE containers** that share volumes and namespace

They are also ephemeral!

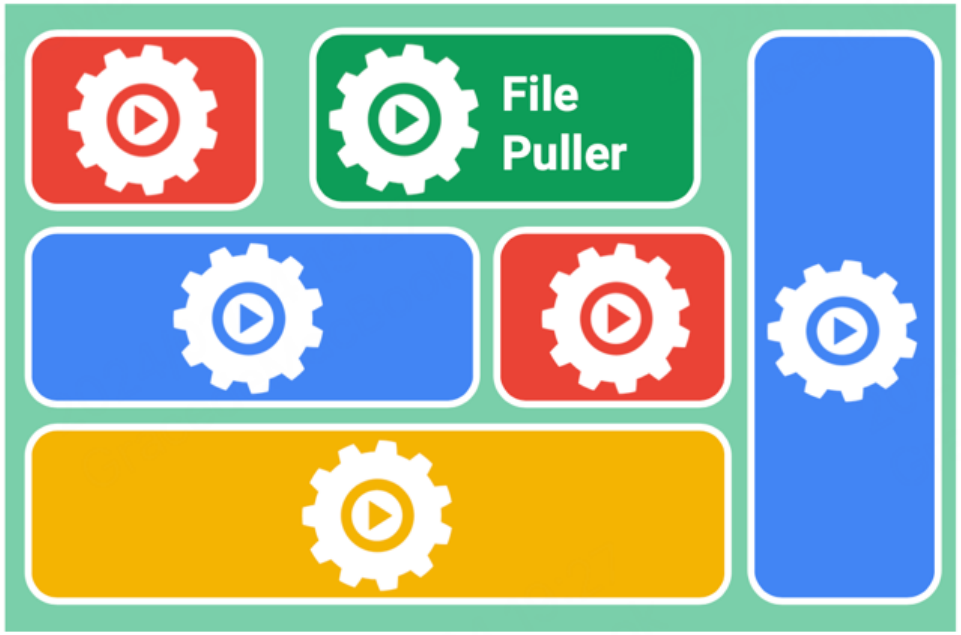


So, why a pod and not container directly?

High-coupled containers



High-coupled containers



So, why a pod and not container directly?

All or nothing approach for a group of symbiotic containers, that need to be kept together at all times

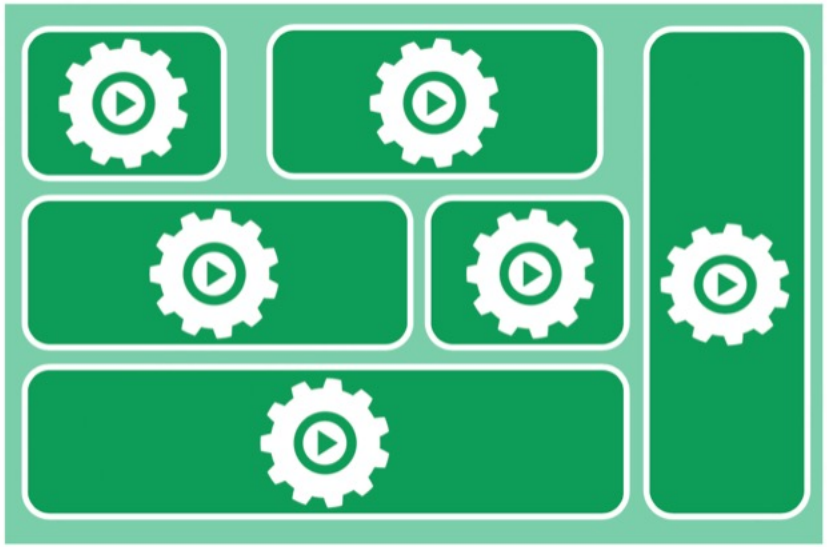
Pod considered running if all containers are scheduled and running

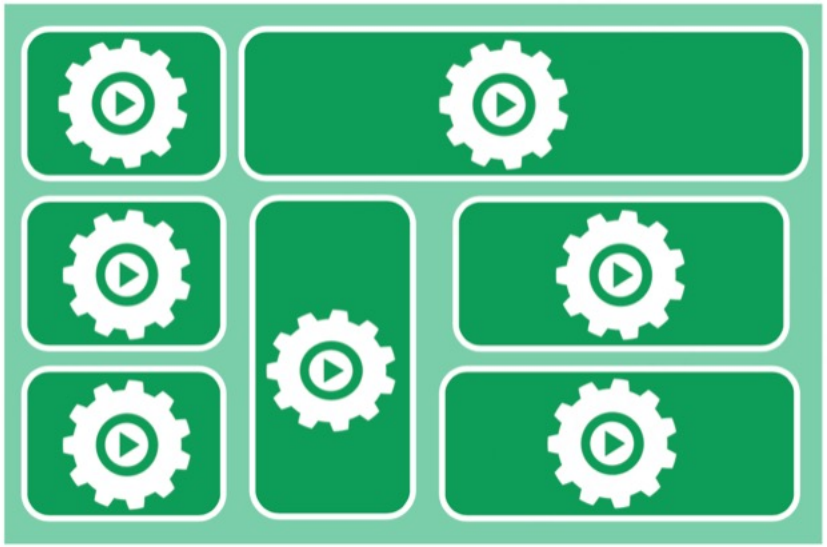
Can you deploy a container in Kubernetes?

- Yes, inside a pod!

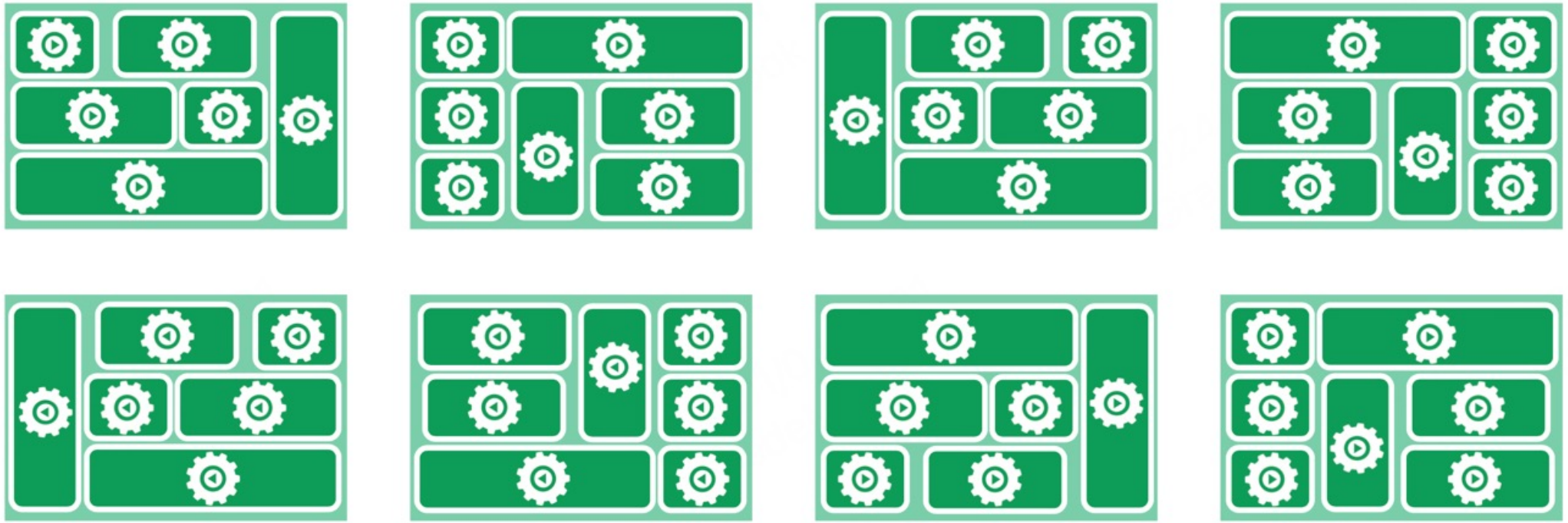
Finding things

Physical view

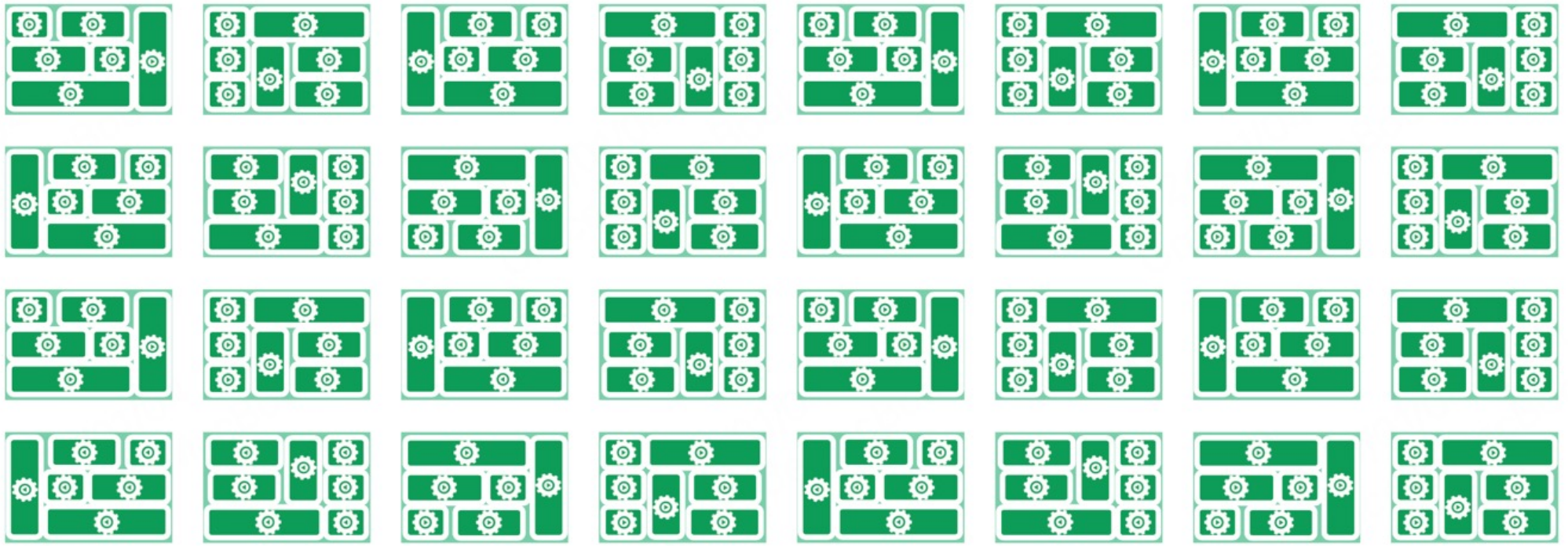




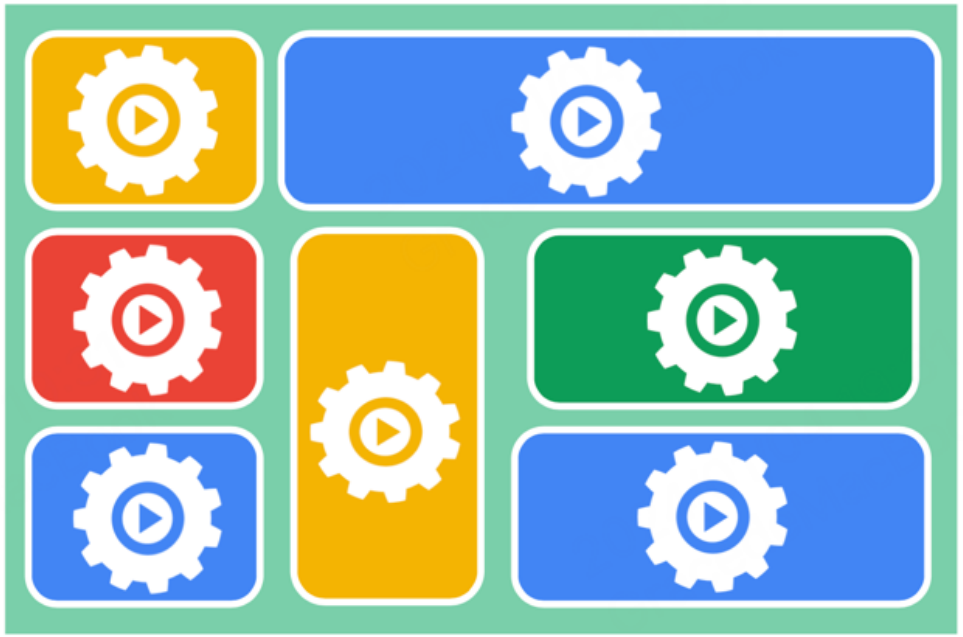
Physical view



Physical view



Logical view



Labels and selectors

Arbitrary metadata

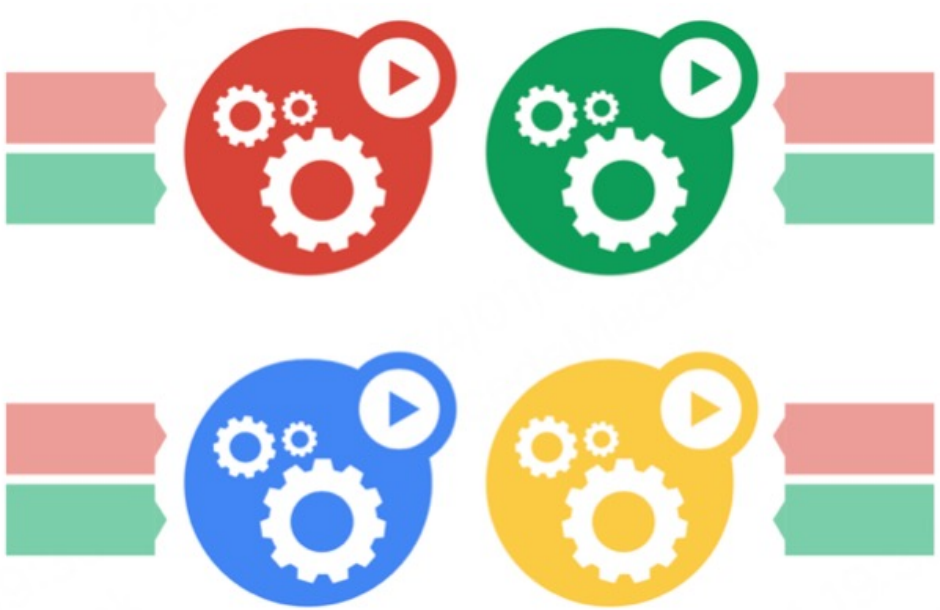
Attached to any API object

Generally represent identity

Queryable by selectors

- Think SQL ‘select … where …’

The only group mechanism



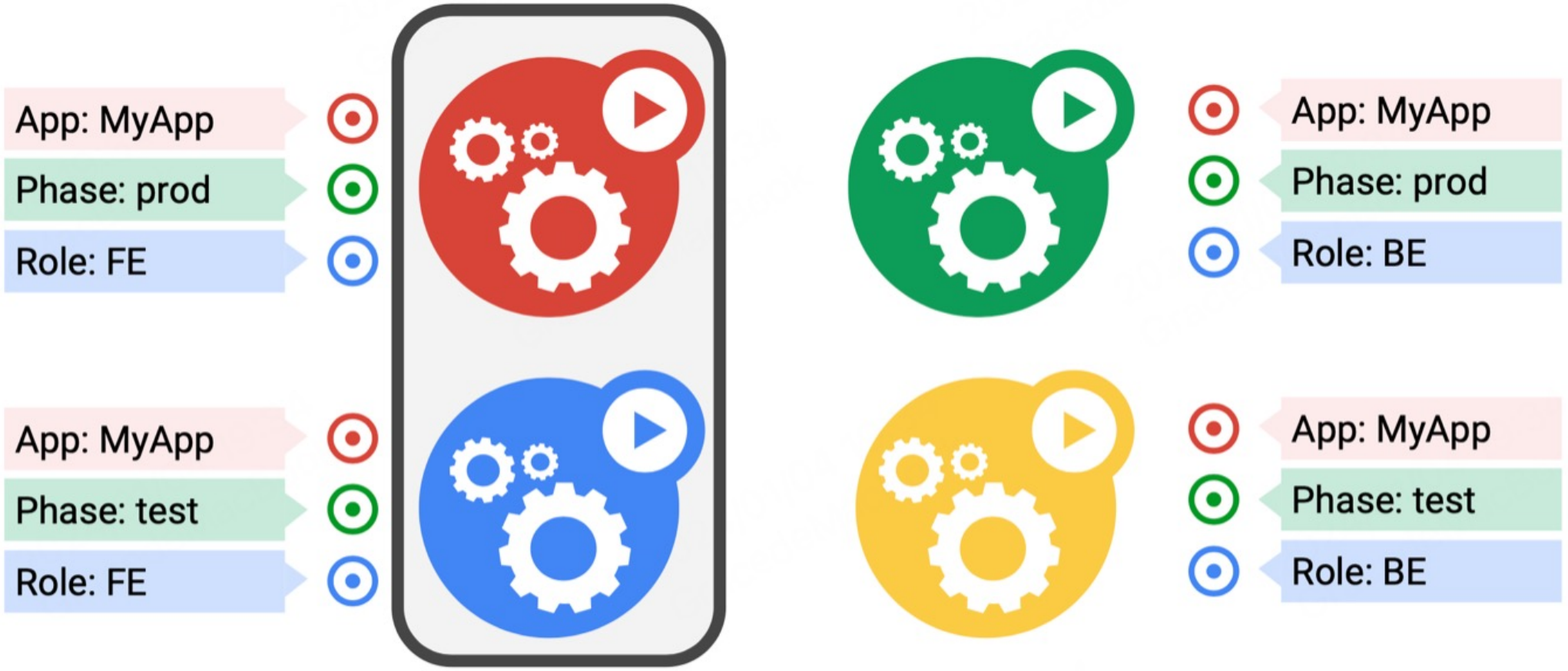
Selectors


App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE

Selectors


App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: BE
App = MyApp

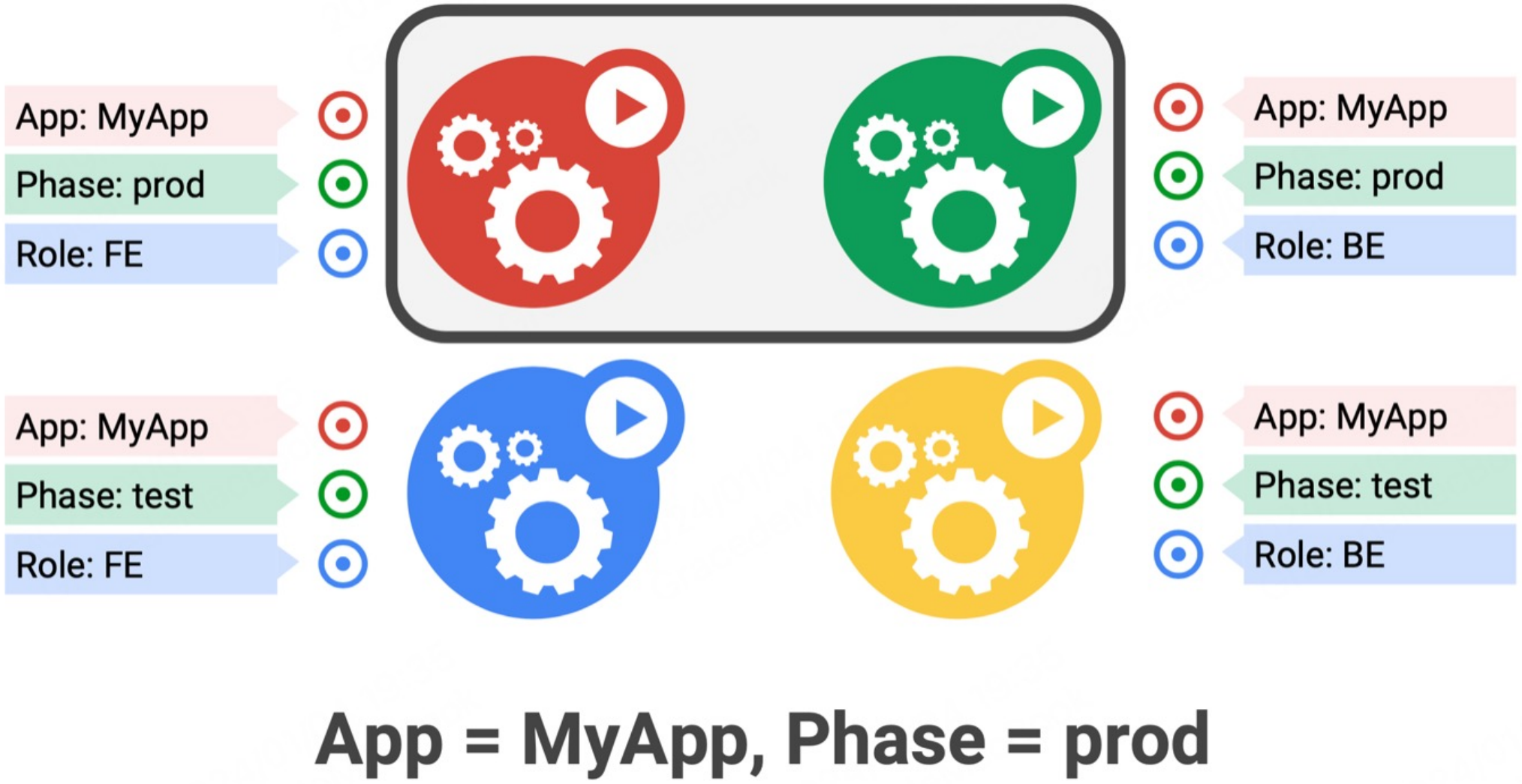
Selectors


App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: BE
App = MyApp, Role = FE

Selectors


App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: BE
App = MyApp, Role = BE

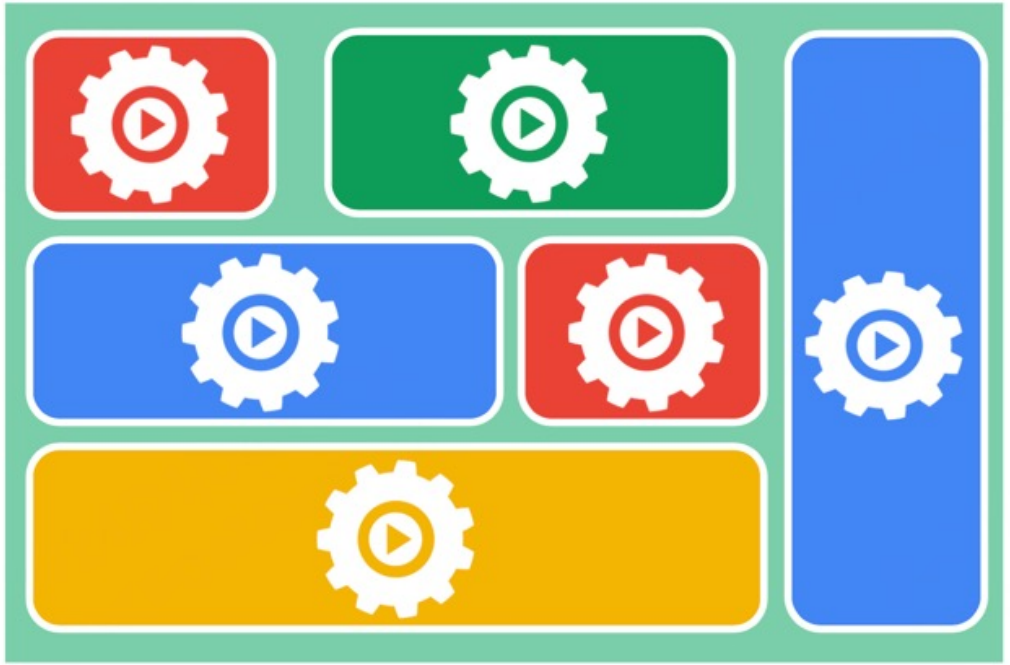
Selectors

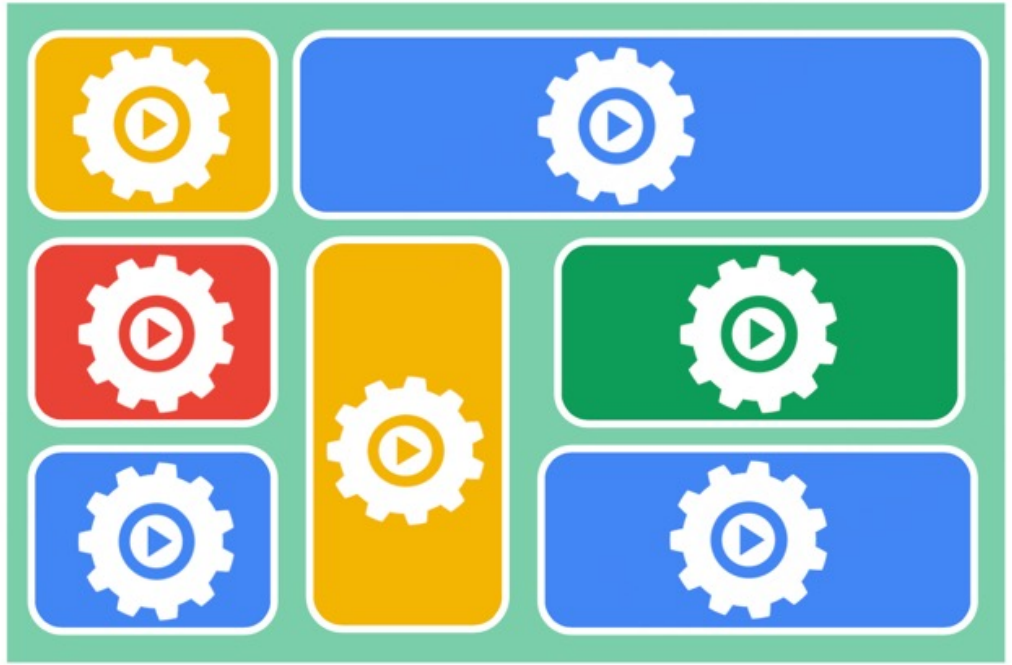

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE
App = MyApp, Phase = prod

Selectors

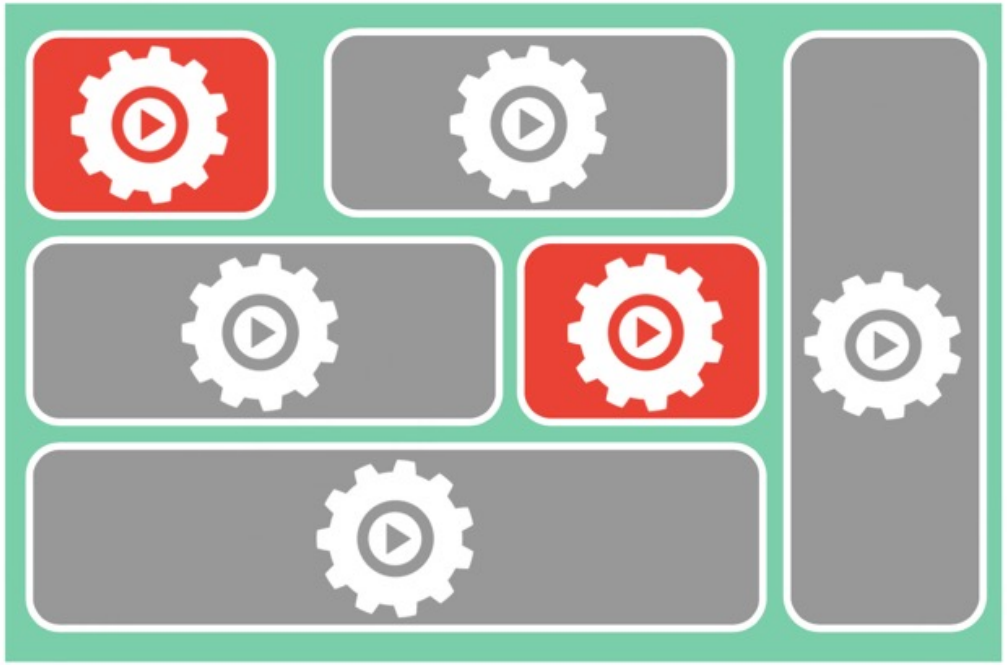

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE
App = MyApp, Phase = test

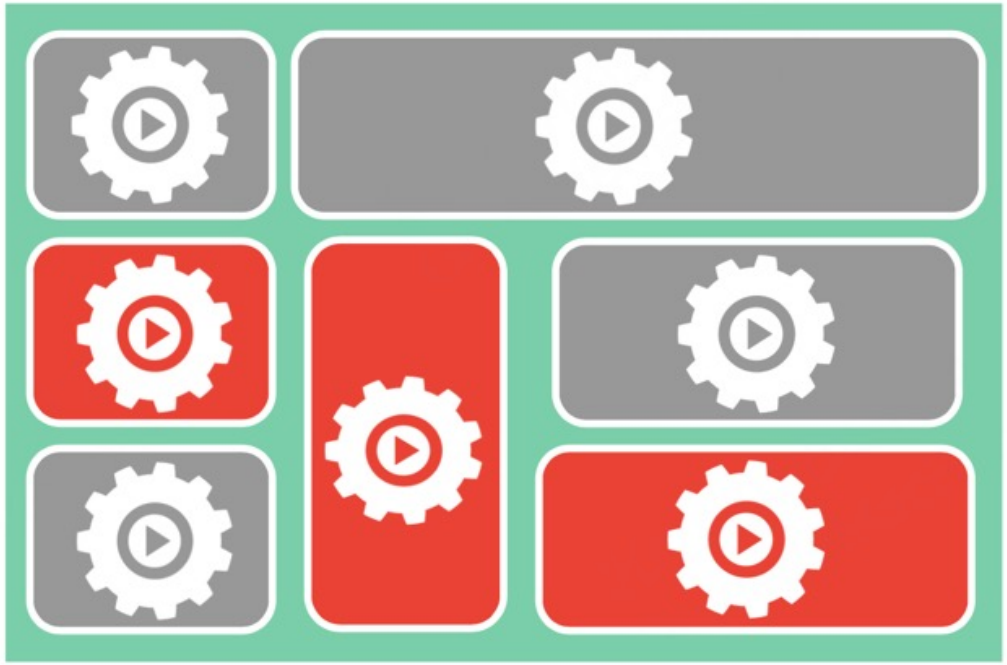
Logical view



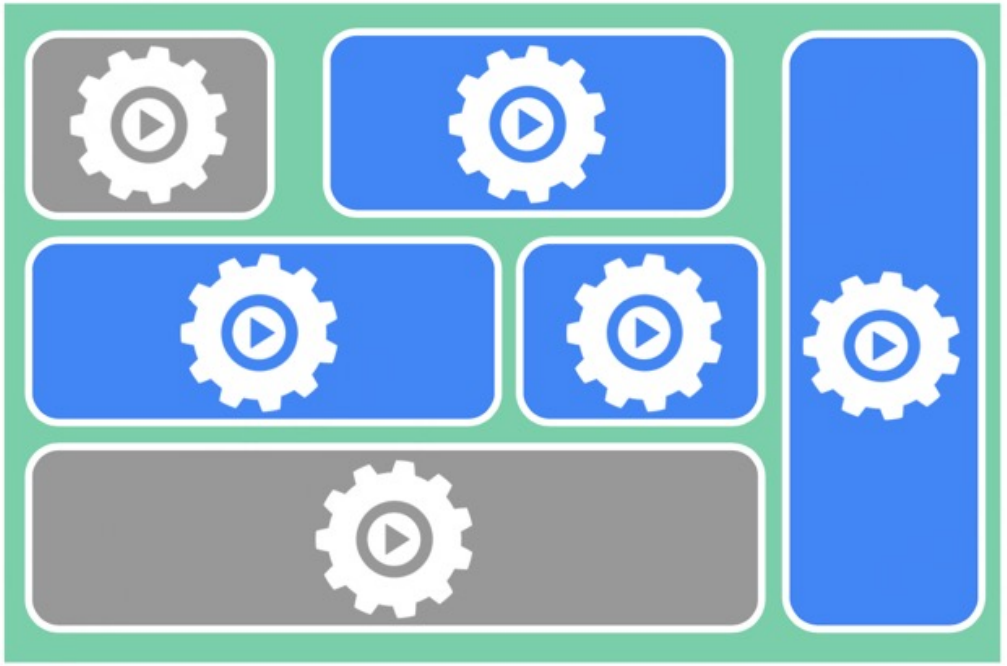


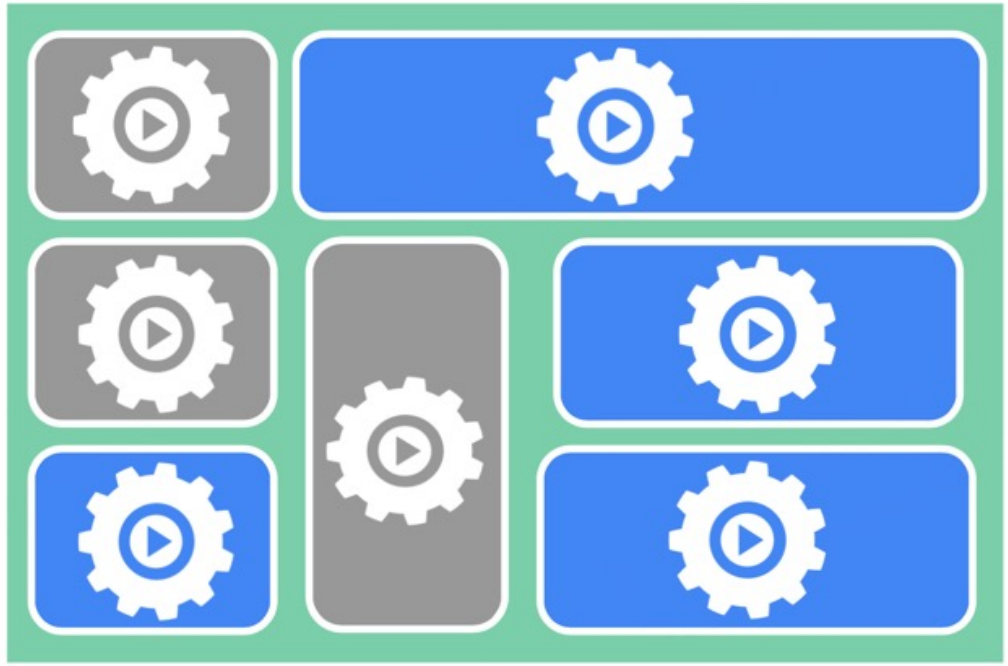
Logical view



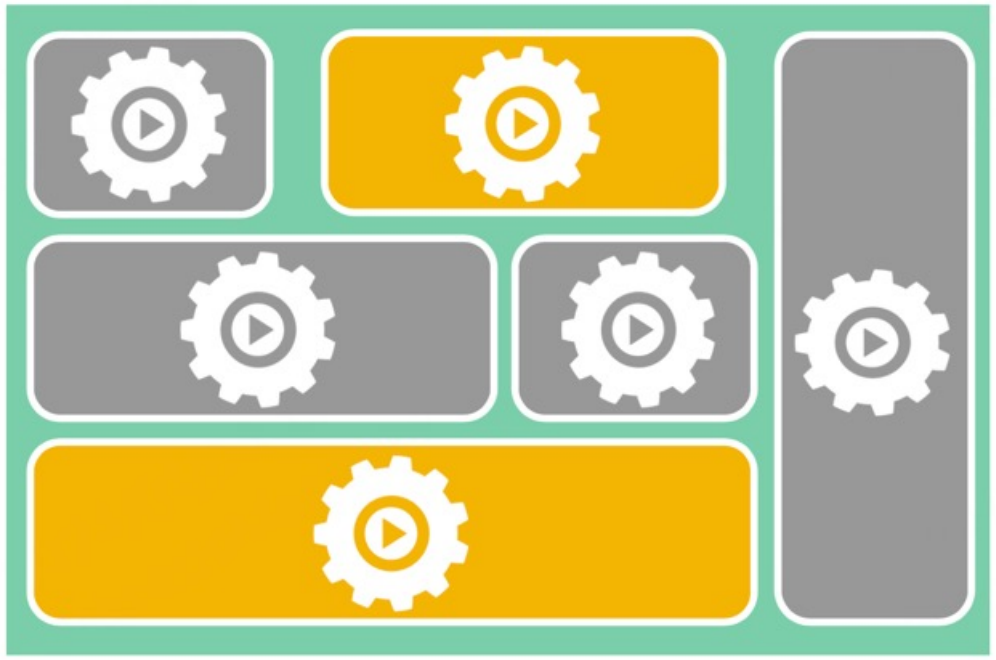


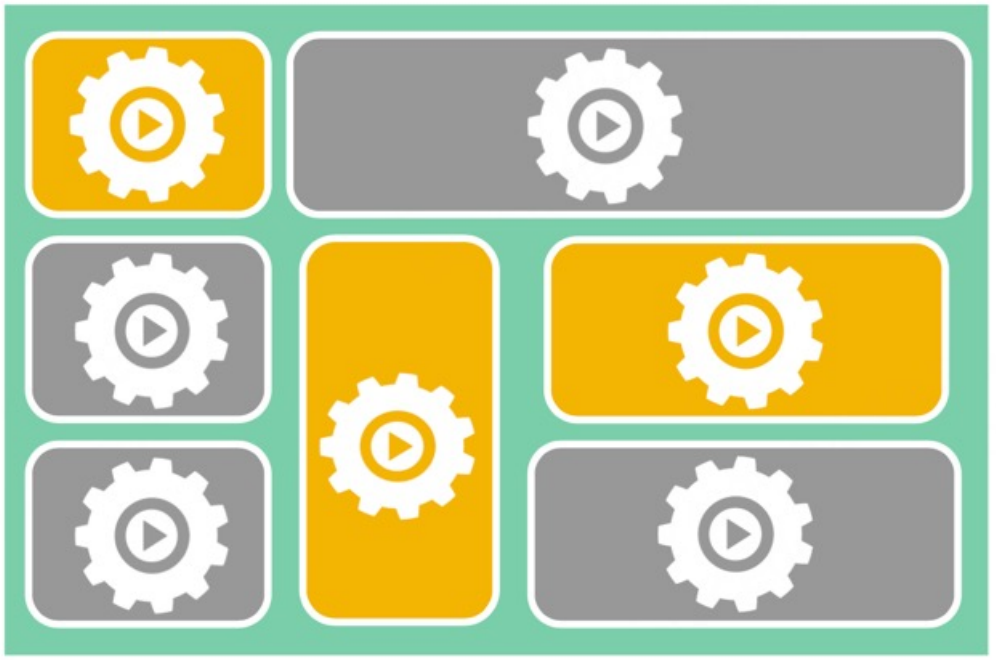
Logical view





Logical view





Discovery

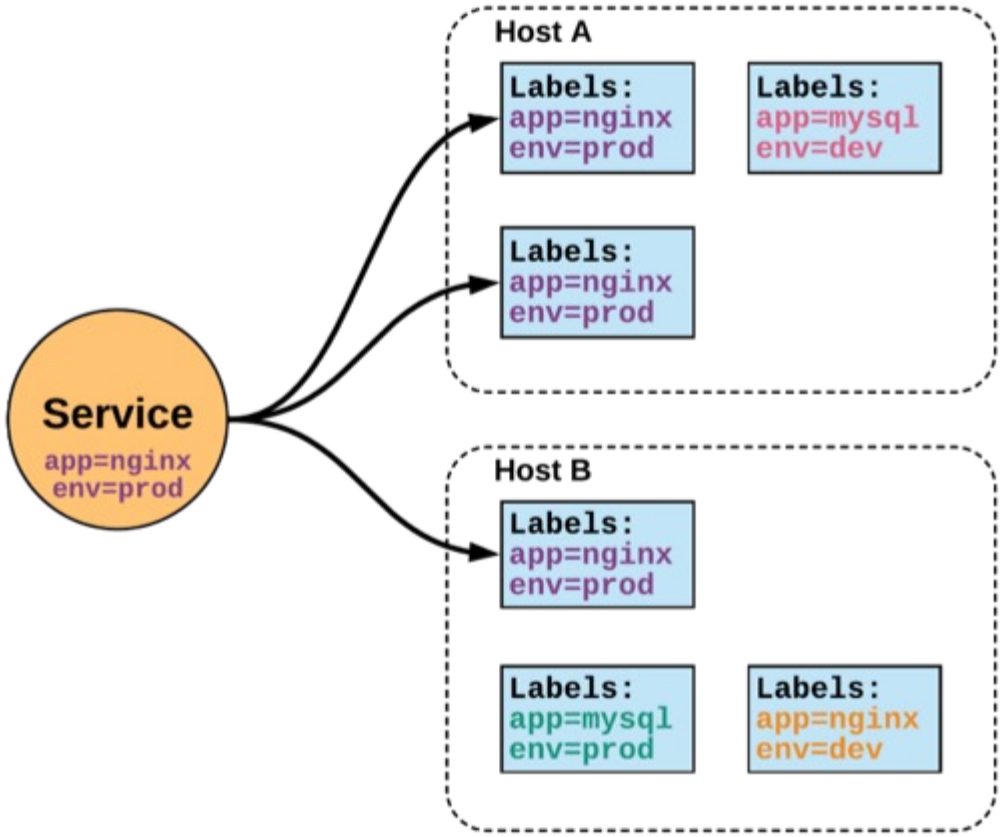
Services

Unified method of accessing the exposed workloads of Pods

Durable resource

- Static cluster IP
- Static namespaced DNS name

NOT Ephemeral!



Services

ClusterIP (default)

NodePort

LoadBalancer

ExternalName

Workloads

ReplicaSet

Deployment

DaemonSet

StatefulSet

Job

CronJob

…

ReplicaSet

Primary method of managing pod replicas and their lifecycle

Includes their scheduling, scaling, and deletion

Their job is simple: **Always ensure the desired number of pods are running**

ReplicaSet

replicas: The desired number of instances of the Pod.

selector: The label selector for the **ReplicaSet** will manage **ALL** Pod instances that it targets; whether it's desired or not.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  template:
    <pod template>
```

Deployment

Way of managing Pods via **ReplicaSets**.

Provide rollback functionality and update control.

Updates are managed through the **pod-template-hash** label.

Each iteration creates a unique label that is assigned to both the **ReplicaSet** and subsequent Pods.

Deployment

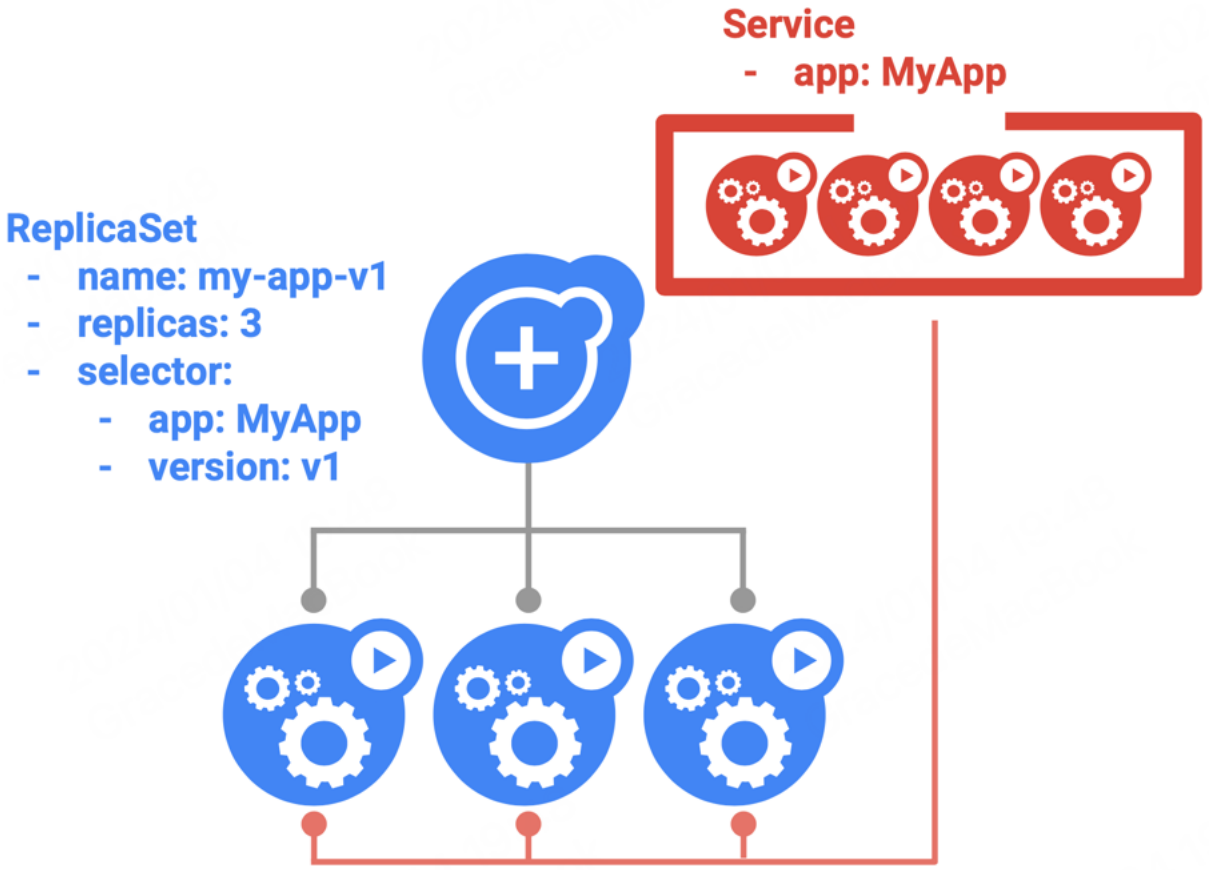
revisionHistoryLimit: The number of previous iterations of the Deployment to retain.

strategy: Describe the method of updating the Pods based on the type.

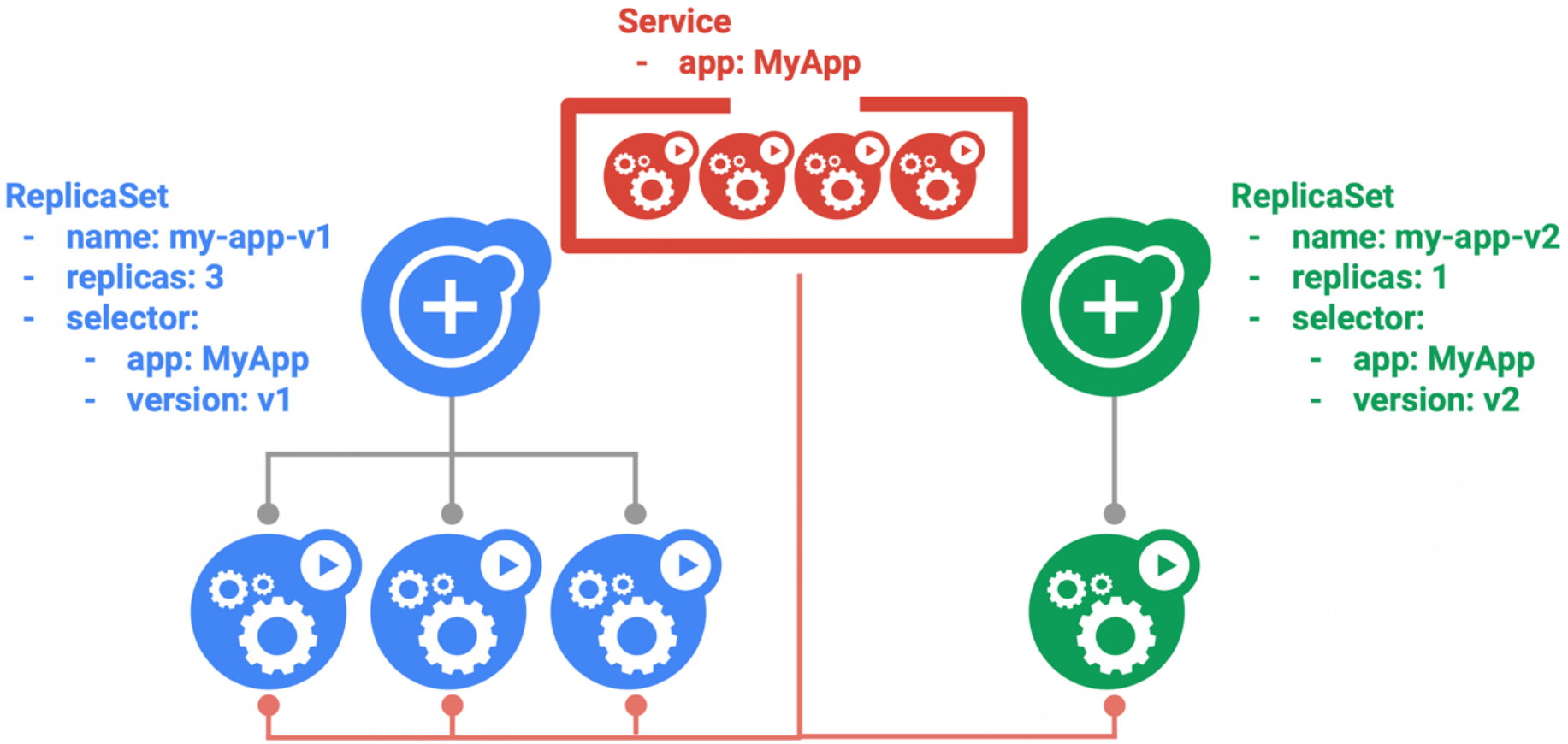
- Recreate
- RollingUpdate

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```

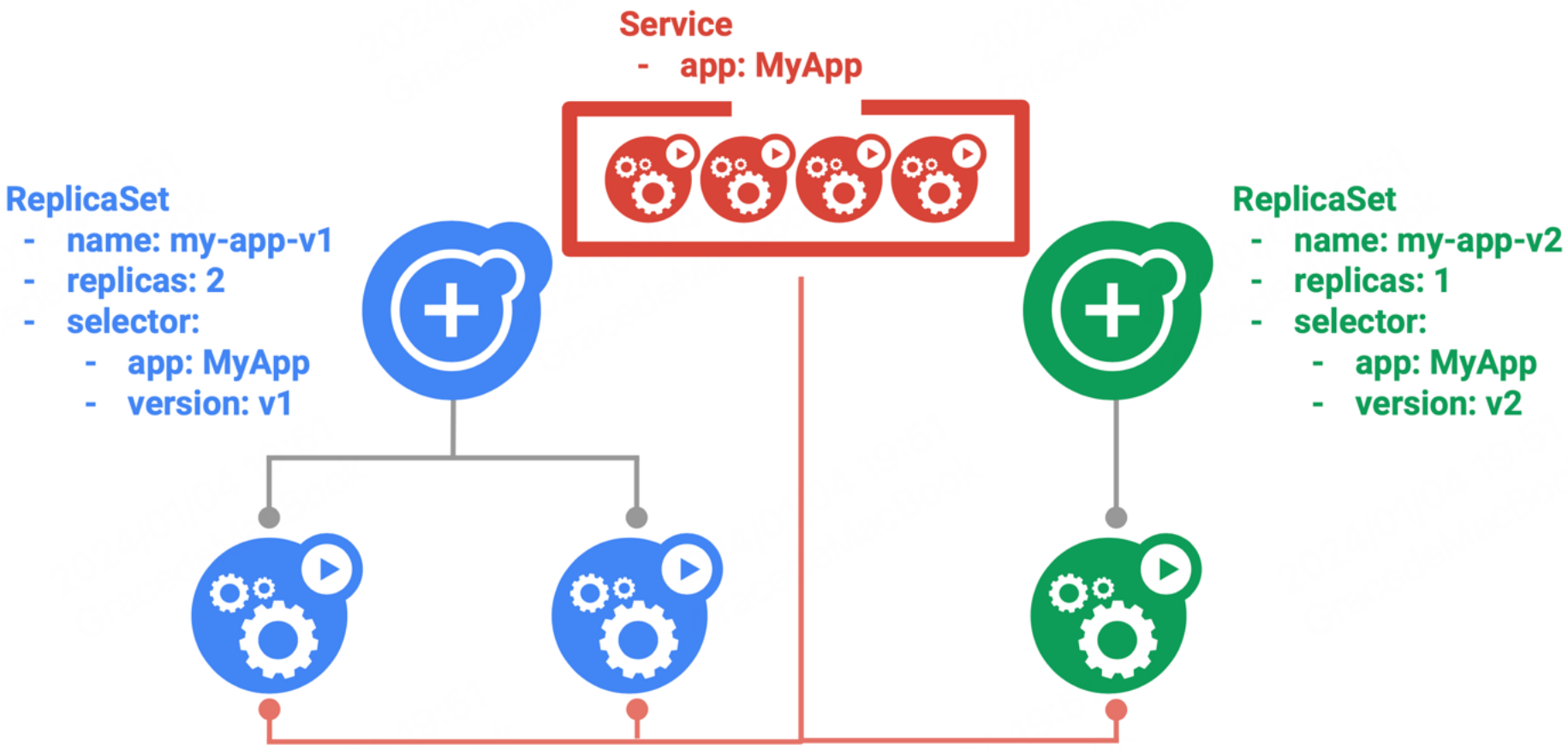
Rolling Update



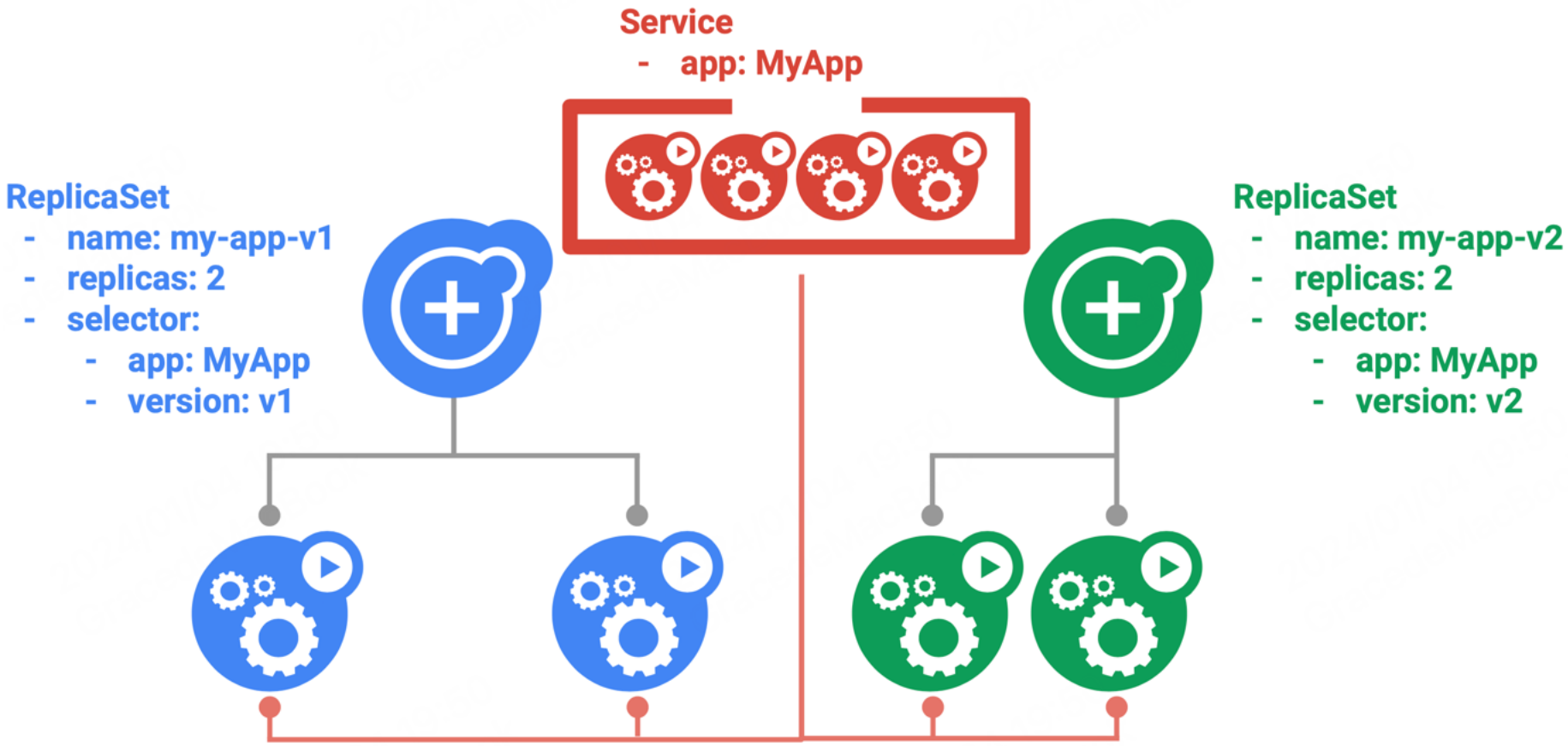
Rolling Update



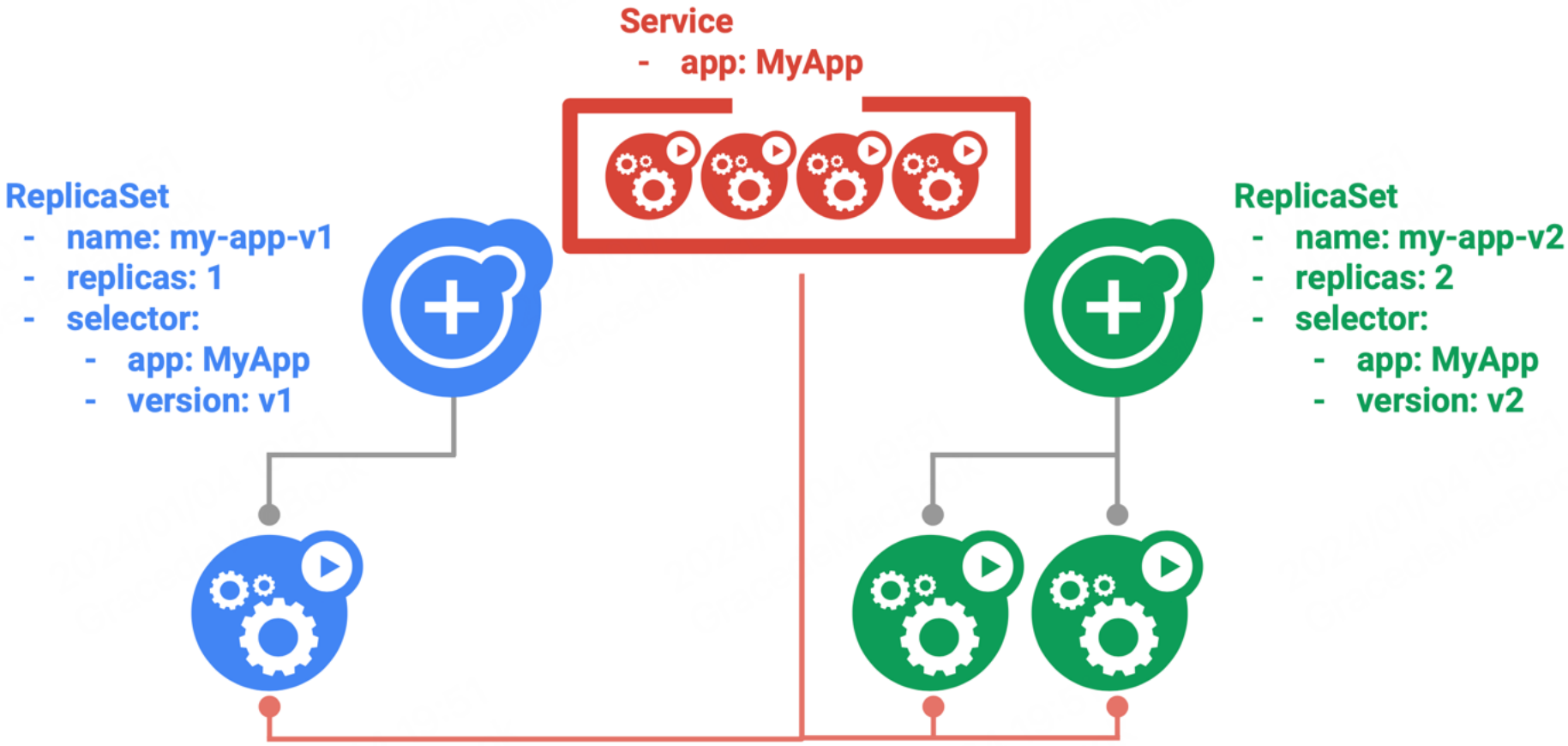
Rolling Update



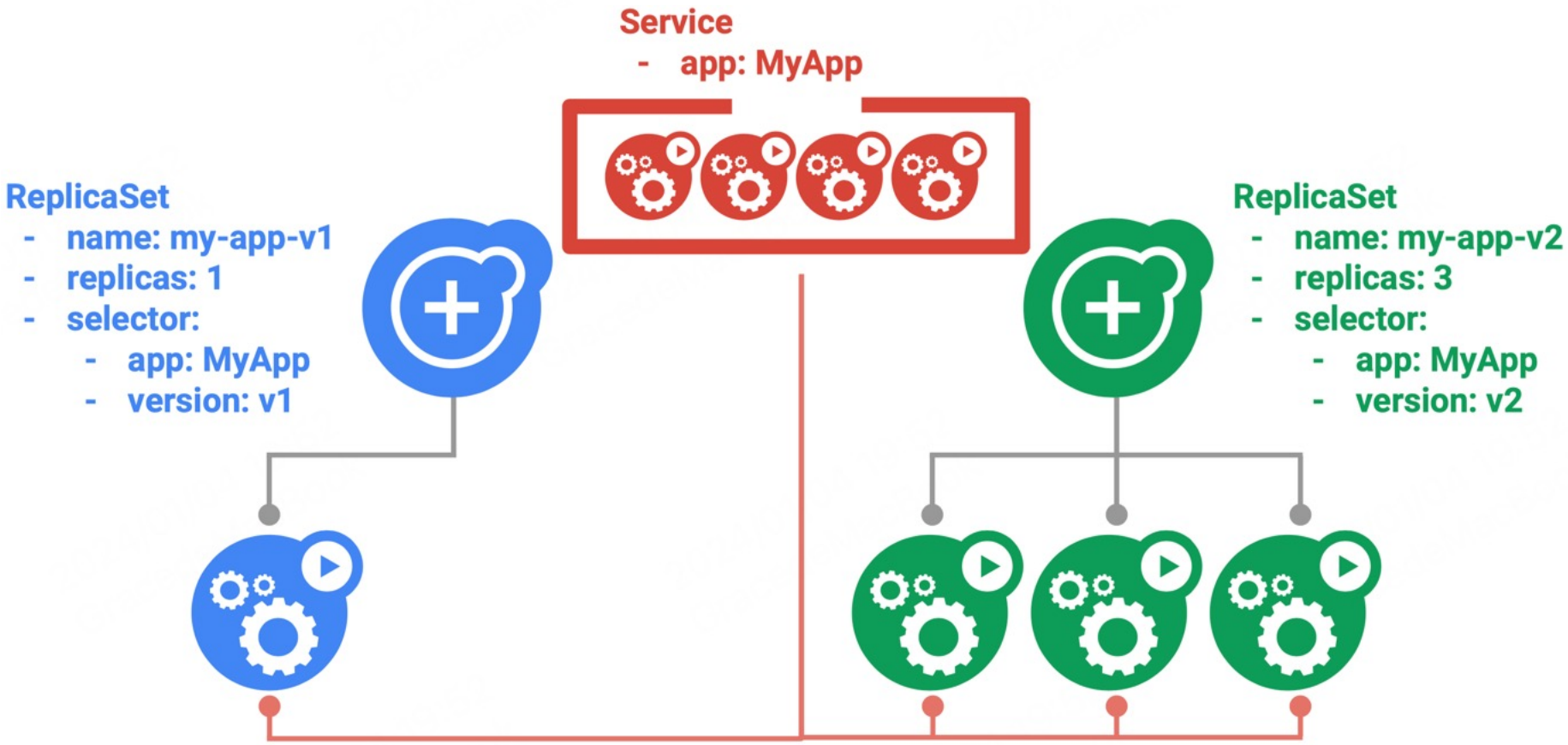
Rolling Update



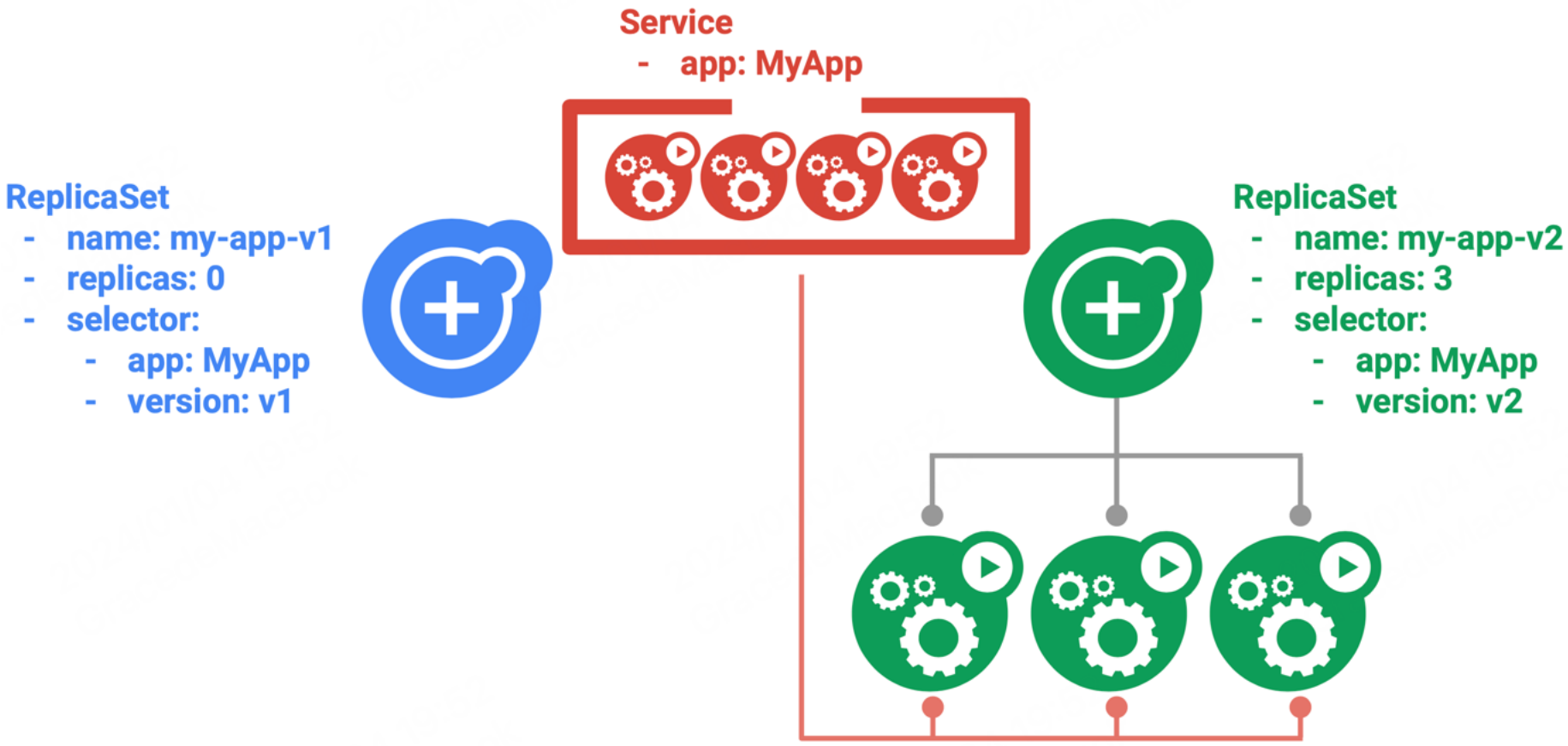
Rolling Update



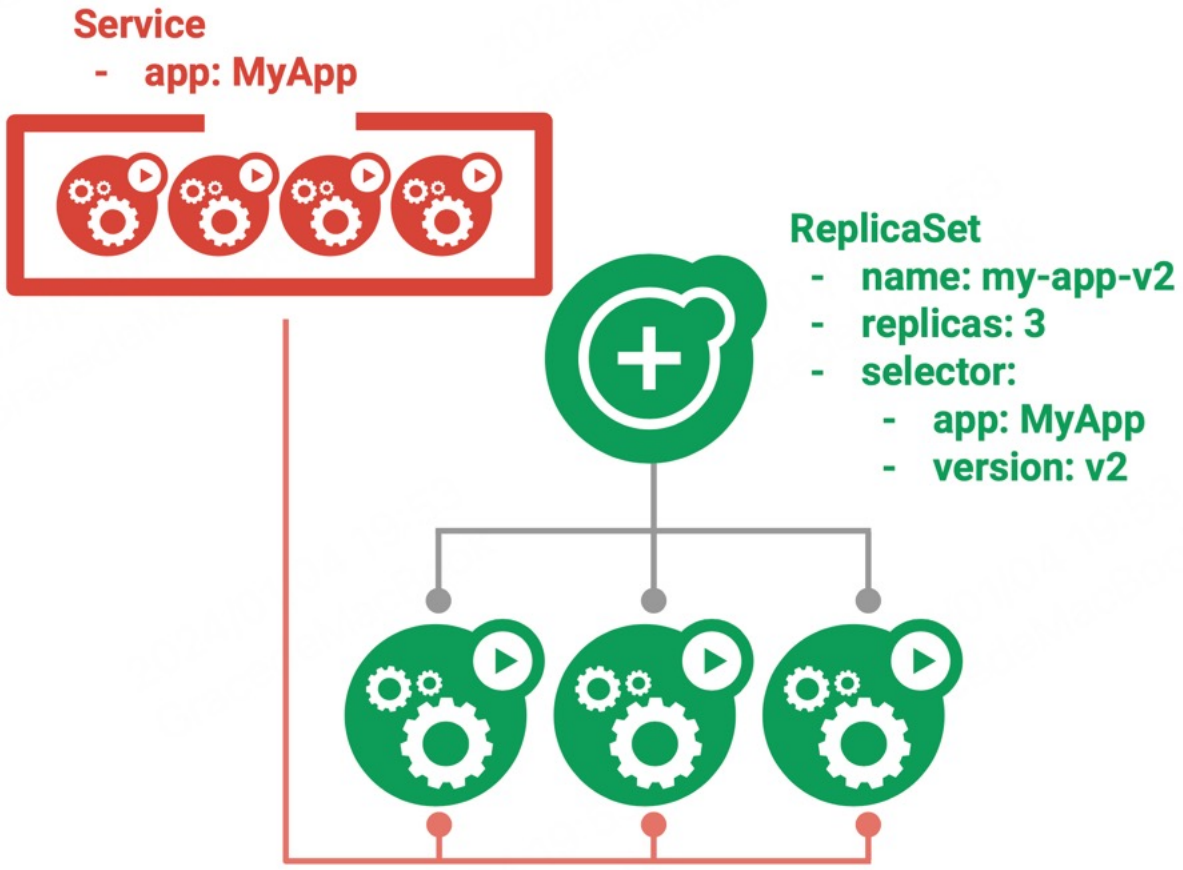
Rolling Update



Rolling Update



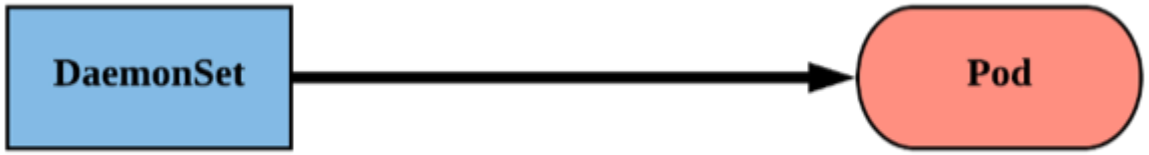
Rolling Update



DaemonSet

Ensure that all nodes matching certain criteria will run an instance of the supplied Pod.

Are ideal for cluster wide services such as log forwarding or monitoring.

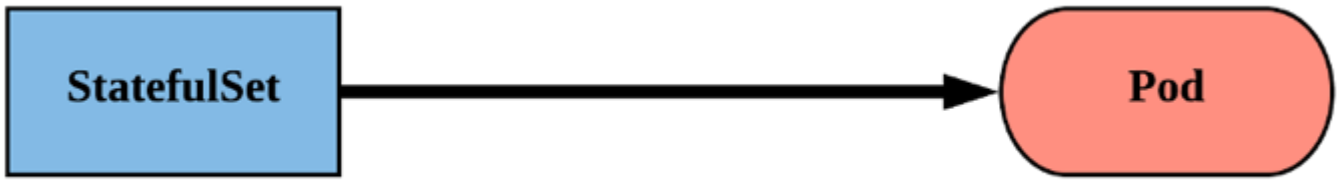


StatefulSet

Tailored to managing Pods that must persist or maintain state.

Pod lifecycle will be ordered and follow consistent patterns.

Assigned a unique ordinal name following the convention of ‘<statefulset name>-<ordinal index>’.

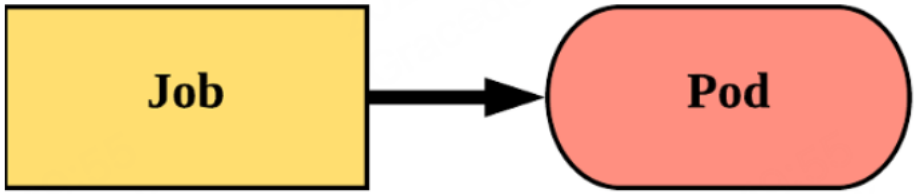


Job

Create one or more pods for carrying out batch processes

Continue to retry execution of the Pods until a specified number of them successfully terminate

Can run multiple Pods in parallel



CronJob

An extension of the Job Controller

Provide a method of executing jobs on cron-like schedule, i.e. running at specified times

Metrics and Monitoring

HPA

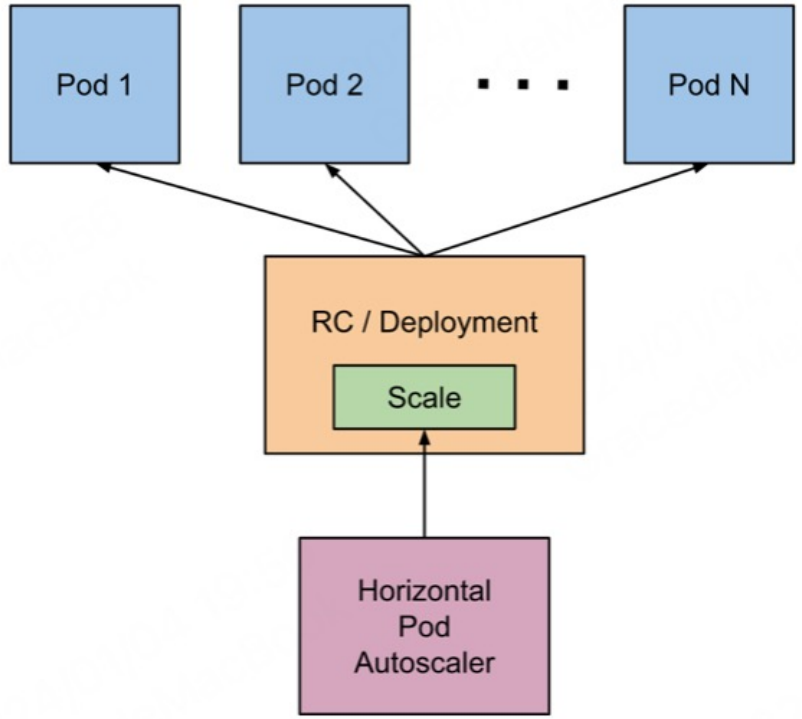
Horizontal Pod AutoScalers

Automatically scale number of pods as needed

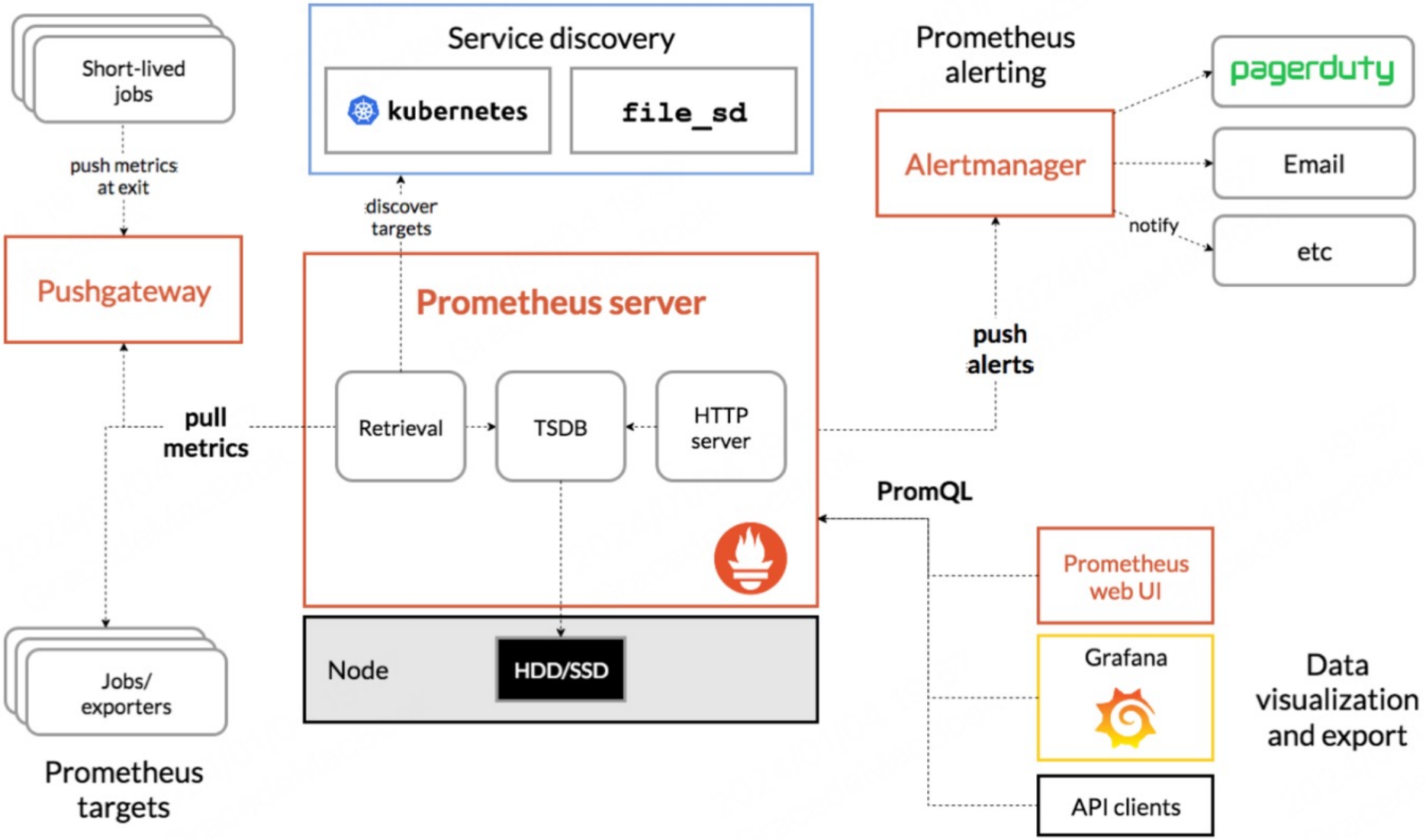
- Based on CPU utilization
- Custom metrics coming

Operate within user-defined

min/max bounds



Monitoring


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

Extending Kubernetes

Extension

Kubernetes - highly configurable and extensible

- Adapt the Kubernetes cluster to the needs of the specific work environment
- Support new types and new kinds of hardware

Extension

Kubectl plugins

API server

Custom Resources

Scheduler

Controllers

…

Credits

- Some slides are adapted from course slides of COMP 4651 in HKUST