Cloud Computing
Container & Kubernetes

容器及其集群管理系统K8S

Minchen Yu

SDS@CUHK-SZ

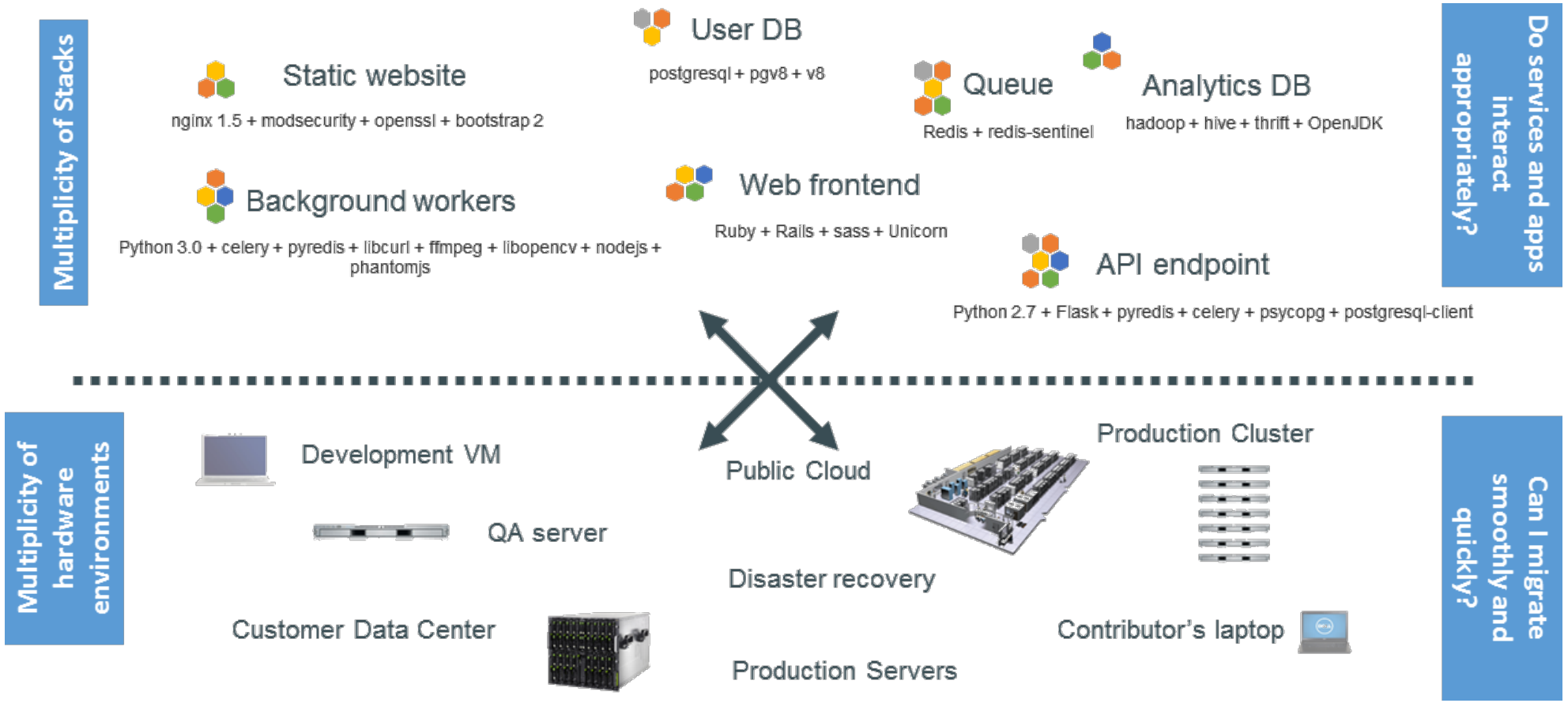
Fall 2024

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?

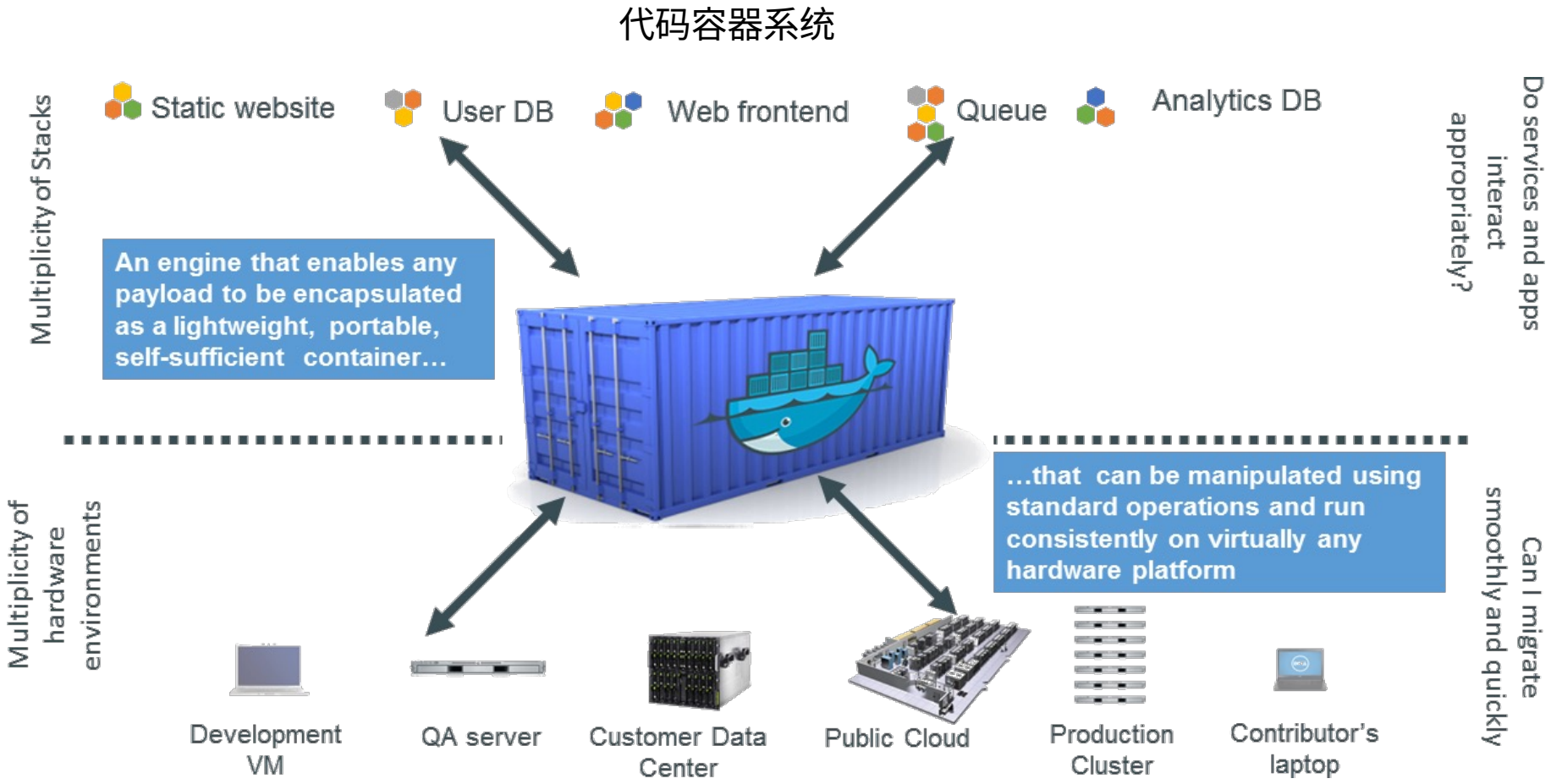
The challenge



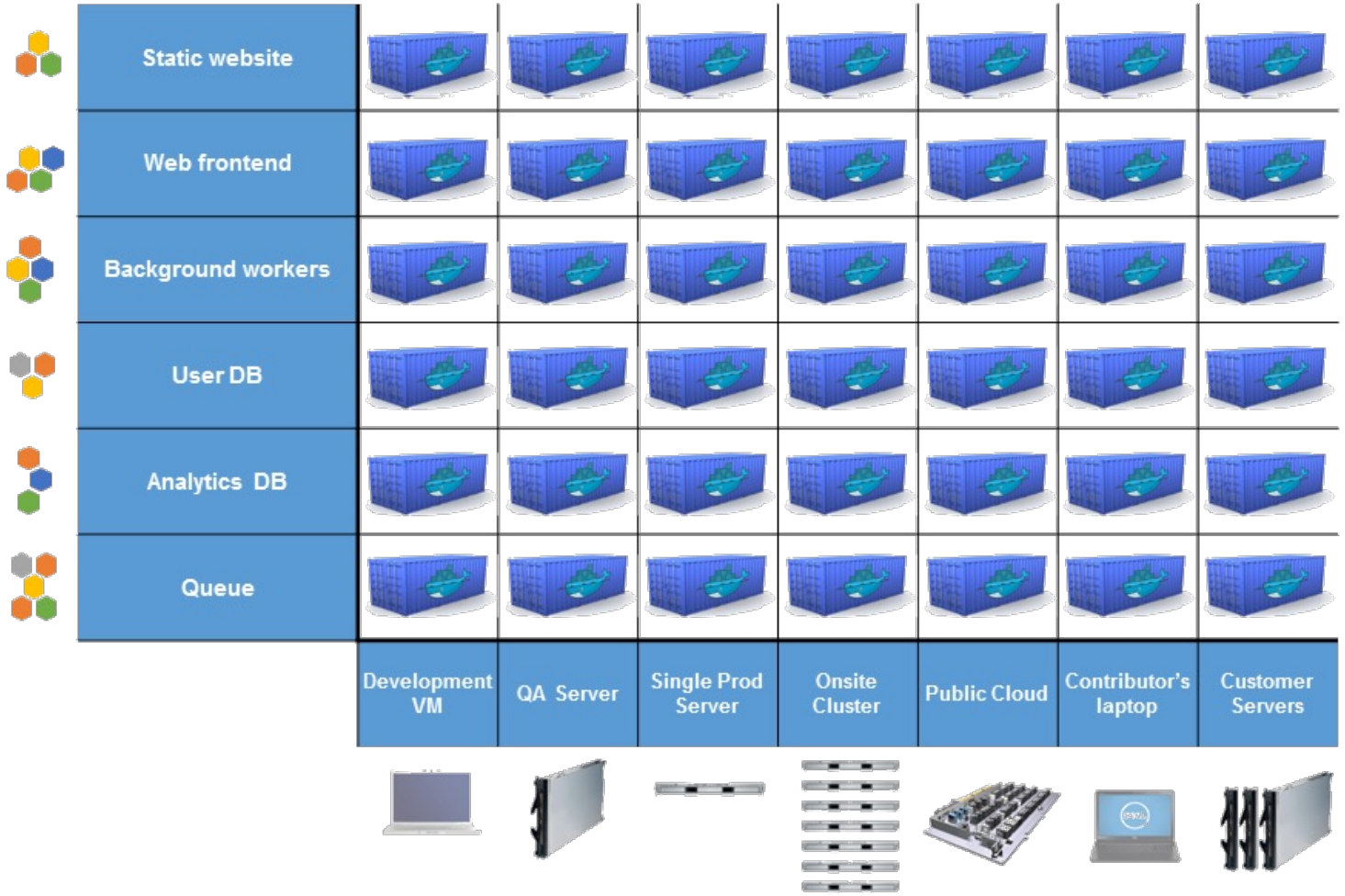
The Matrix from hell

	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

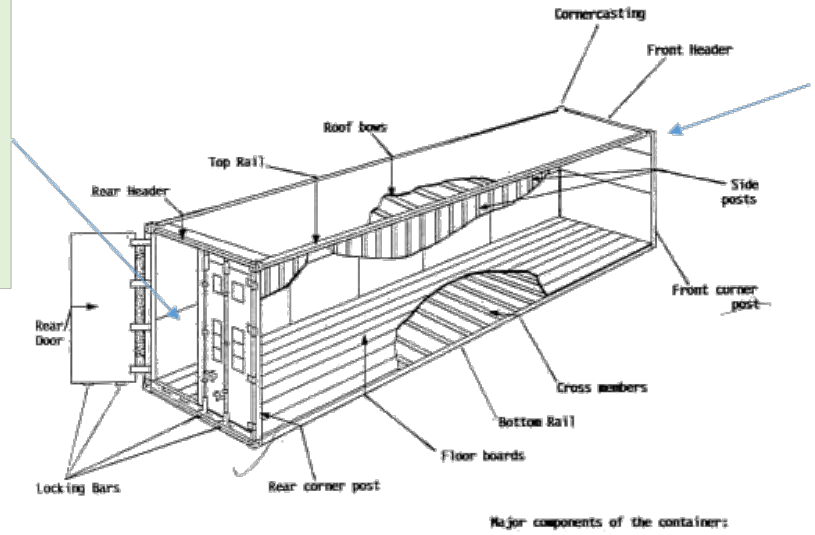


Eliminate the matrix from hell



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



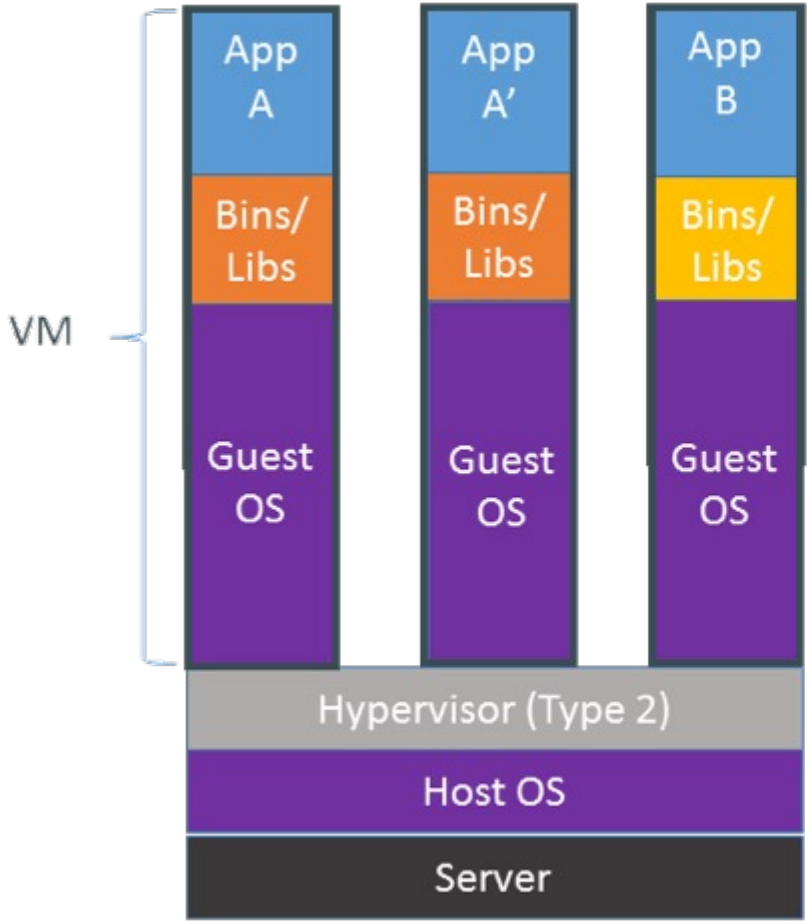
Major components of the container

- **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

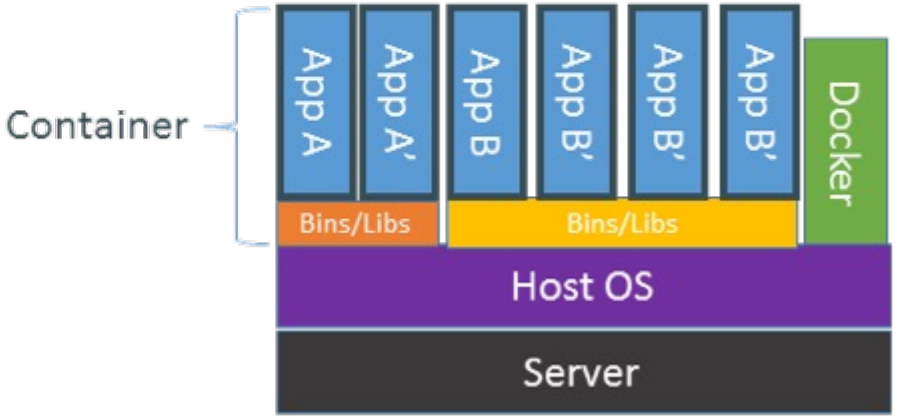
VM vs. Containers

虚拟机VS容器



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

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



容器是孤立的，但共享操作系统，并酌情共享仓库/库

结果是部署速度显著加快，开销大大减少，迁移更容易，重新启动速度更快

Container implementation

容器的实现

Leveraging Linux kernel mechanisms

利用 Linux 内核机制

- **cgroups:** manage resources for groups of processes

cgroups:管理进程组的资源

- **namespaces:** per process resource isolation

命名空间:每个进程的资源隔离

- seccomp: limit available system calls

seccomp:限制可用的系统调用

- capabilities: limit available privileges

功能:限制可用权限

- CRIU: checkpoint/restore (w/ kernel support)

CRIU:检查点/恢复(带内核支持)

cgroups: Linux control groups

cgroups:Linux控制组

- control group subsystem offering a resource management solution for a group of processes

控制组子系统为进程组提供资源管理解决方案

- Each subsystem has a ***hierarchy*** **(a tree)**

每个子系统都有一个层次结构(树)

  - separate hierarchies for CPU, memory, block I/O

CPU、内存、块 I/O 的单独层次结构

  - each process is in a node in each hierarchy

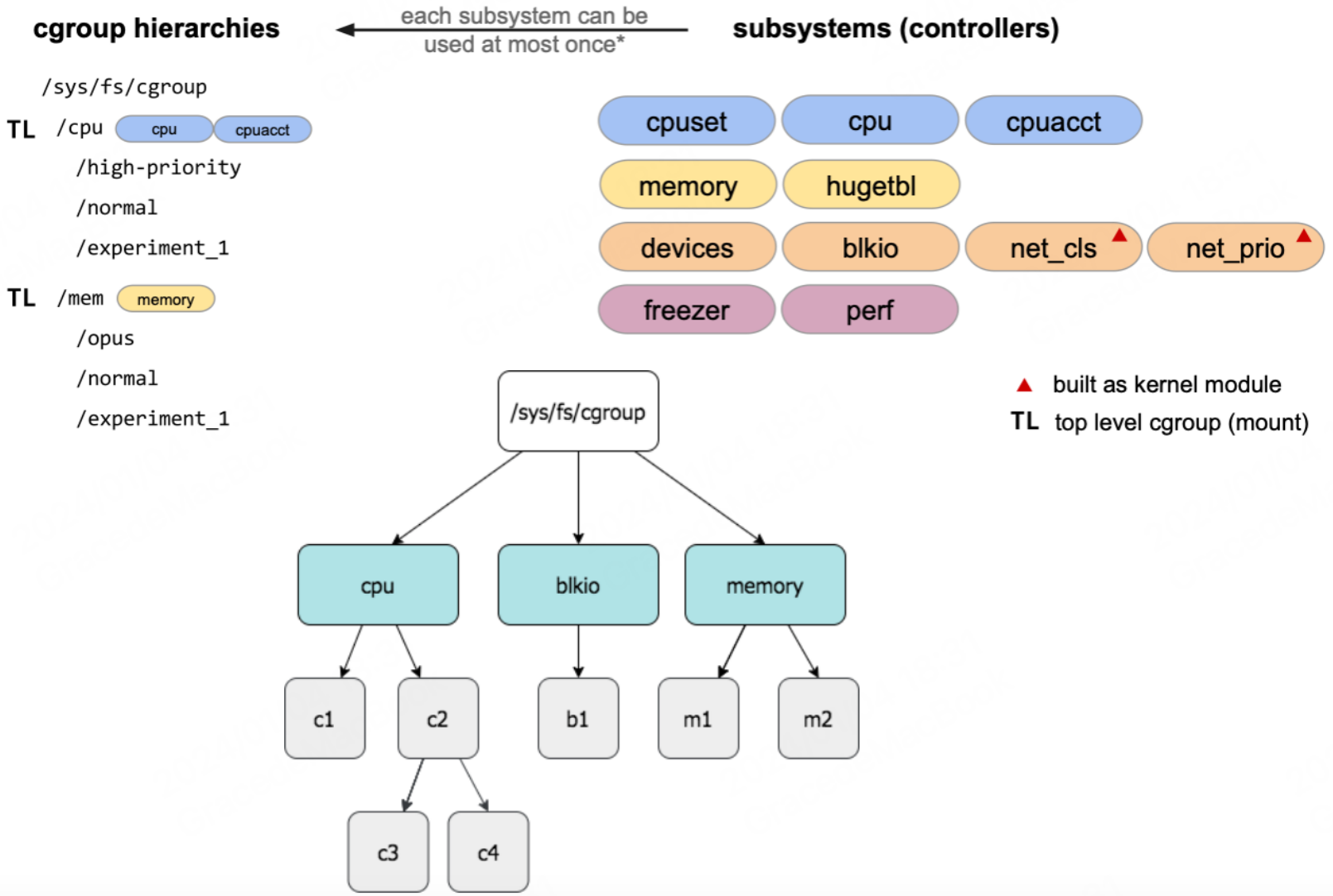
每个流程位于每个层次结构中的一个节点中

  - each node = a group of processes sharing the same resources

每个节点=一组共享相同资源的进程

cgroup hierarchies

层次结构



namespaces

命名空间

在进程粒度上限制内核端名称和数据结构的范围

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

管理的三个系统调用

`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

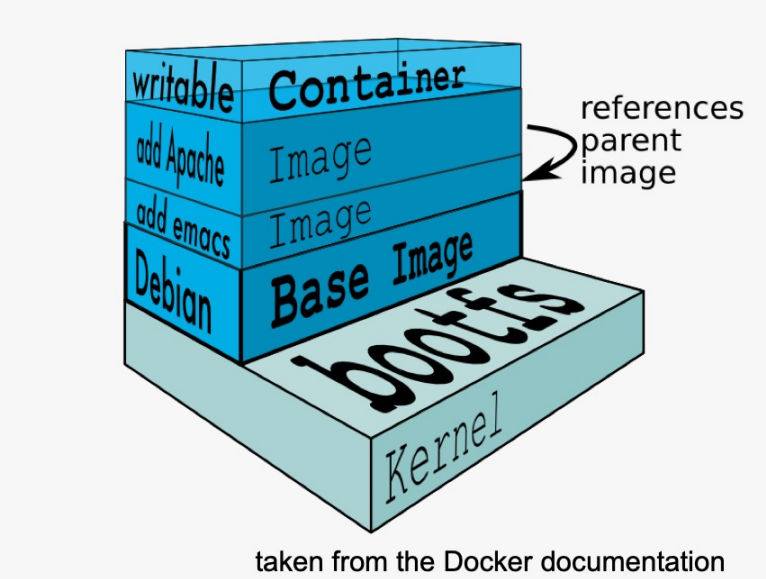
容器

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

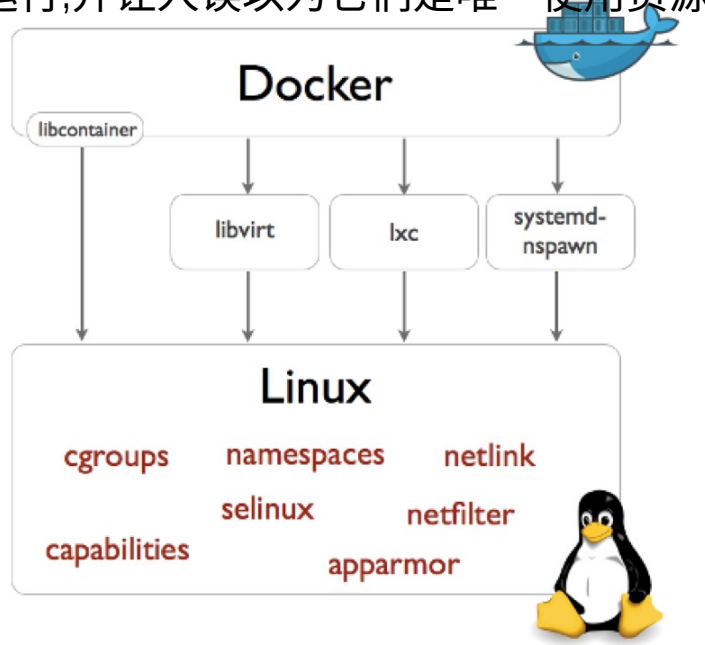
基于 cgroups 和 namespaces 等内核机制的轻量级资源虚拟化

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

多个容器在同一个内核上运行,并让人误以为它们是唯一使用资源的容器



taken from the Docker documentation



Docker

Provides container runtime and the isolation among containers

提供容器运行时以及容器间的隔离

Helps them share the OS

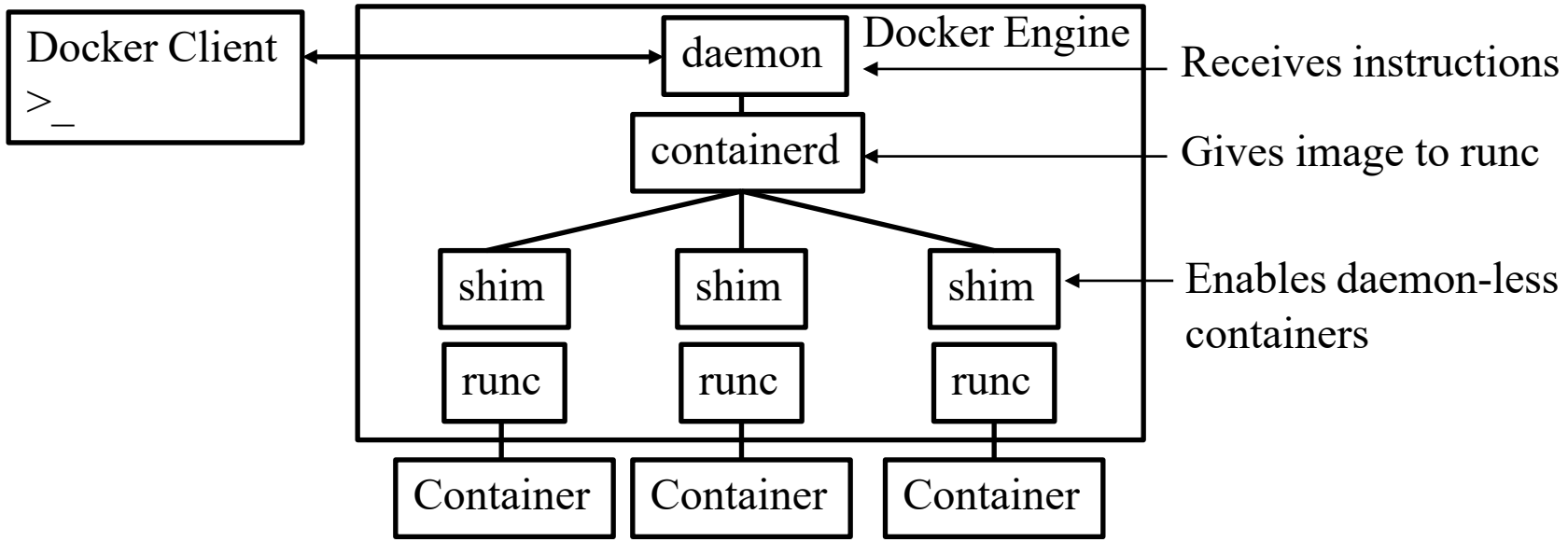
帮助他们共享操作系统

Developed initially by Docker.com

Downloadable for Linux, Windows, and Mac

Docker

- **daemon**: API and other features daemon守护进程: API和其他功能
- **containerd**: Execution logic. Responsible for container lifecycle. Start, stop, pause, unpause, delete containers containerd :执行逻辑。负责容器生命周期。启动、停止、暂停、取消暂停、删除容器
- **runc**: A lightweight runtime CLI runc:轻量级运行时 CLI
- **shim**: runc exists after creating the container. shim keeps the container running. Keep stdin/stdout open shim: runc 在创建容器后存在。shim 使容器保持运行。保持 stdin/stdout 开放



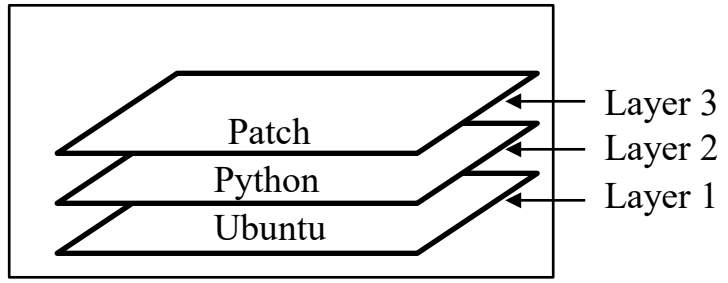
Docker images

Docker 镜像

- Containers are built from images and can be saves as images
  容器由图像构建,可以保存为图像
- Images are stored in registries
  图像存储在注册表中
  - Local registry on the same host
    同一主机上的本地注册表
  - Docker Hub Registry: Globally shared
    包括：
    Docker Hub Registry:全球共享
  - Private registry on Docker.com
    Docker.com 上的私有注册表
- Any component not found in the local registry is downloaded from specified location
  任何在本地注册表中找不到的组件都将从指定地点加载
- Each image has several tags, e.g., v2, latest, …
  每个图像都有多个标签,例如 v2、latest
- Each image is identified by its 256-bit hash
  每幅图像都通过其 256 位哈希值进行识别

Layers

- Each image has many layers 每幅图像都有许多层
- Image is built layer by layer 图像是逐层构建的
- Layers in an image can be inspected by Docker commands 可以通过 Docker 命令检查镜像中的各层
- Each layer has its own 256-bit hash 每层都有自己的 256 位哈希值
- For example: 例如:
  - Ubuntu OS is installed, then Ubuntu 操作系统安装完成后
  - Python package is installed, then Python 包已安装,然后
  - a security patch to the Python is installed 安装了 Python 的安全补丁
- Layers can be shared among many containers 多个容器之间可以共享图层



Build docker images

Create a Dockerfile that describes the application, its dependencies, and how to run it
创建一个Dockerfile ,描述应用程序、其依赖项以及如何运行它

Dockerfile		Description
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

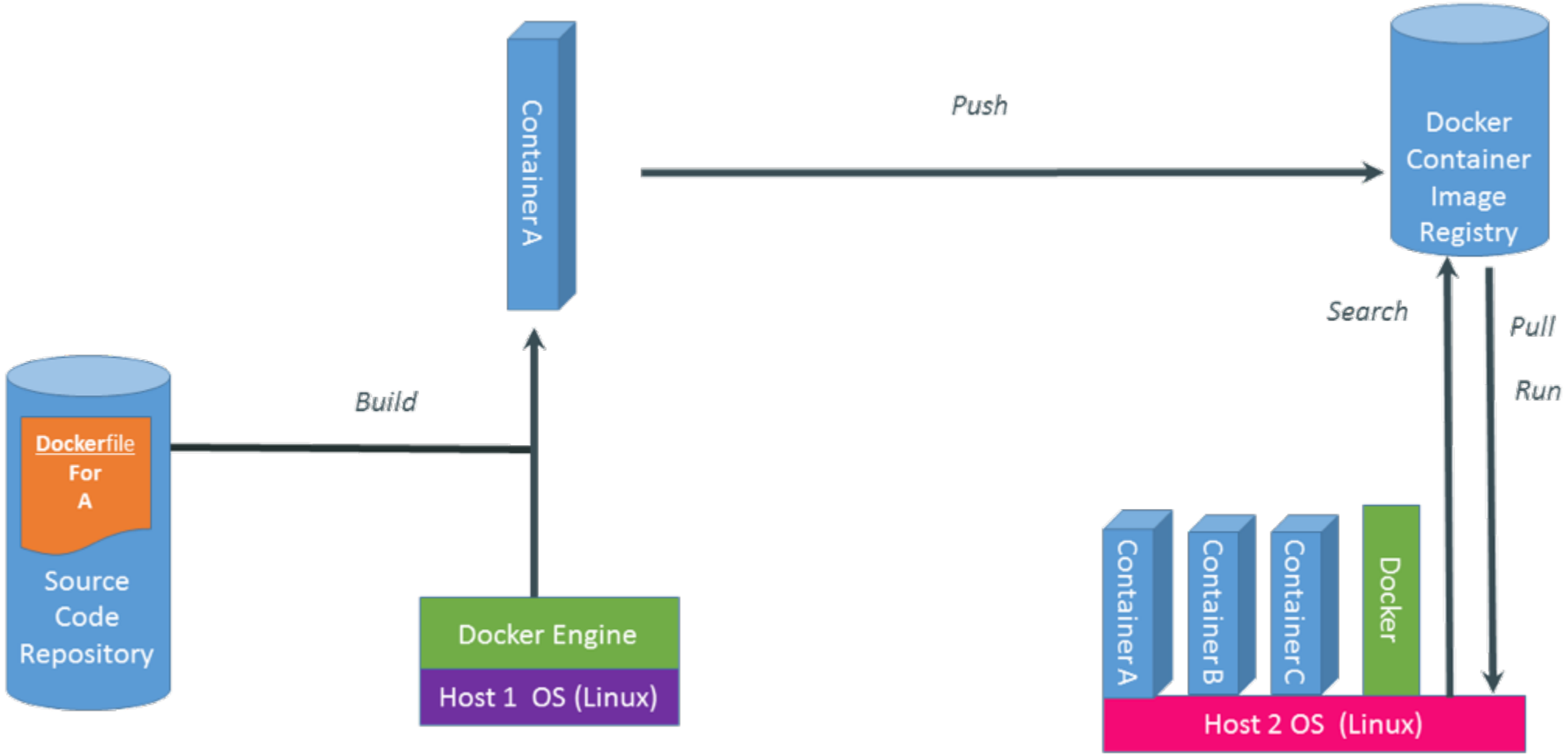
RUN nmp install	←	Layer 4
Copy . /src	←	Layer 3
RUN apk add …	←	Layer 2
FROM Alpine	←	Layer 1

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

Docker commands

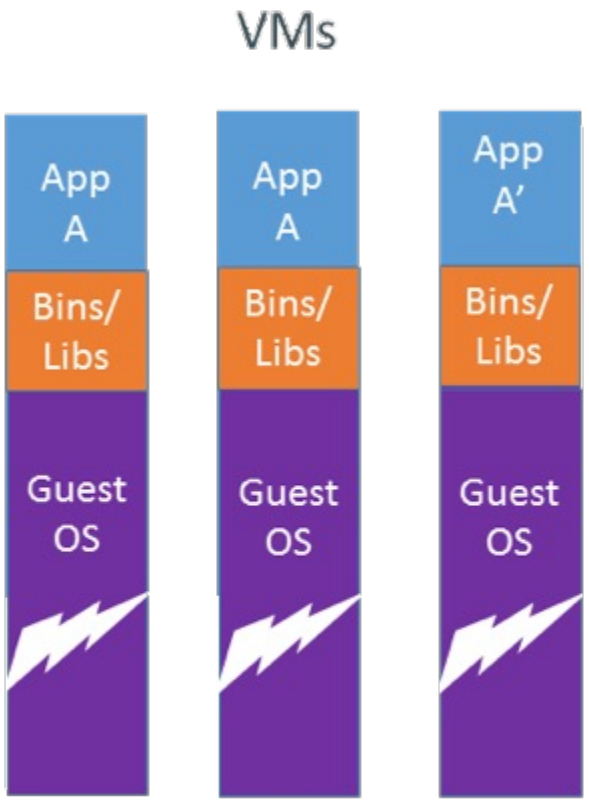
- **docker container run**: Run the specified image
  run:运行指定镜像
- **docker container ls**: list running containers
  ls:列出正在运行的容器
- **docker container exec**: run a new process inside a container
  exec:在容器内运行新进程
- **docker container stop**: Stop a container
  stop:停止容器
- **docker container start**: Start a stopped container
  start:启动已停止的容器
- **docker container rm**: Delete a container
  rm :删除容器
- **docker container inspect**: Show information about a container
  inspect:显示有关容器的信息

Overview of a Docker system 系统概述



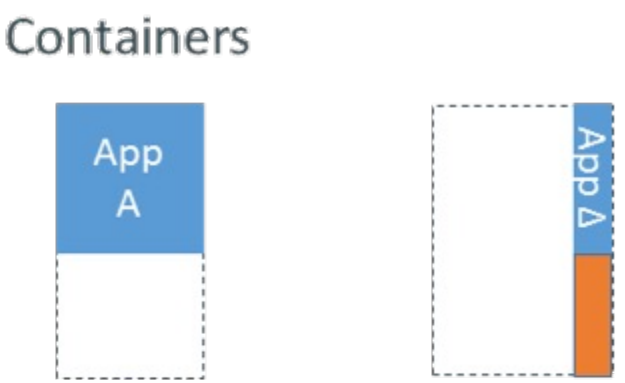
Configure once, run anything anywhere 一次配置，随处运行

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)

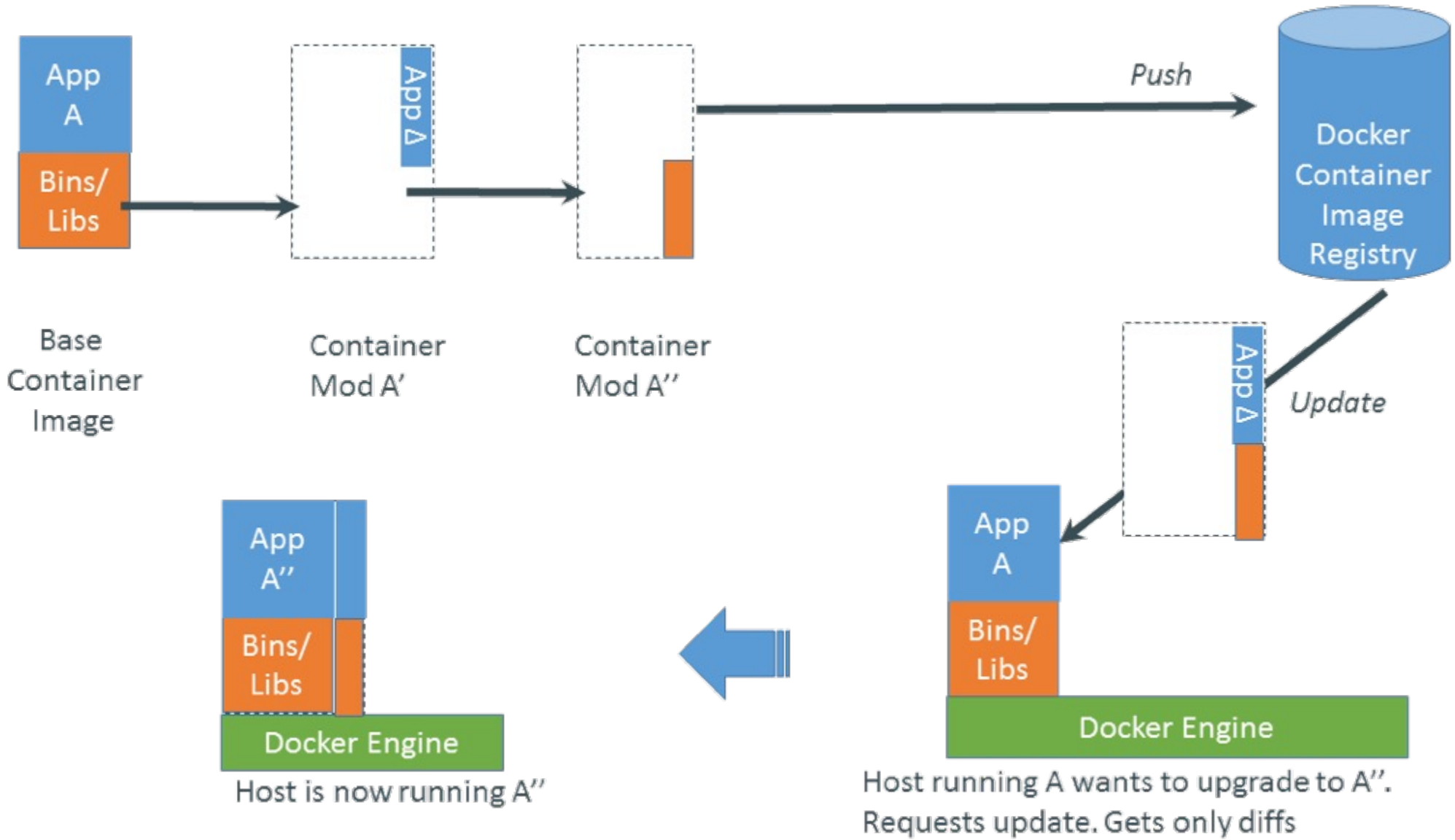
Copy of App

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)?

如何大规模部署容器

Deployment options

部署选项

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

了解容器的工作原理与在生产中使用它们不同

Different organizations will have very different needs

不同的组织会有截然不同的需求

- Not everyone needs to operate “at scale”

“ 大规模 ” 运营并非每个人都需要

There are many ways to run and manage containers

运行和管理容器的方法有很多种

Humans

Never underestimate the value of manual solutions

永远不要低估手动解决方案的价值

SSH into machines and run docker

通过 SSH 进入机器并运行 docker

- Pro: simple, available everywhere, no special tools needed, easily understood

  优点:简单、随处可用、无需特殊工具、轻松理解

- Con: not automated, not reproducible (human make mistakes), doesn't scale, doesn't self-heal

  缺点:非自动化、不可重复(人为失误)、不规模,不能自我修复

Scripts

脚本

A very common first step into containers

进入容器的常见第一步

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

包括Puppet、Chef、Ansible、Salt 或定制脚本

- Pro: integrates with existing environments, easily understood results, reproducible

  优点:与现有环境集成,结果易于理解，可重复

- Con: manual scheduling, doesn't self-heal, doesn't scale, generally non-portable

  缺点:手动调度,无法自我修复,通常无法扩展，不可携带

Container orchestration

容器编排

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?

Docker 最初是在 Linux 容器之上构建的 CL 工具: 创建、启动、停止和执行容器

根据具体要求进行节点级别的管理

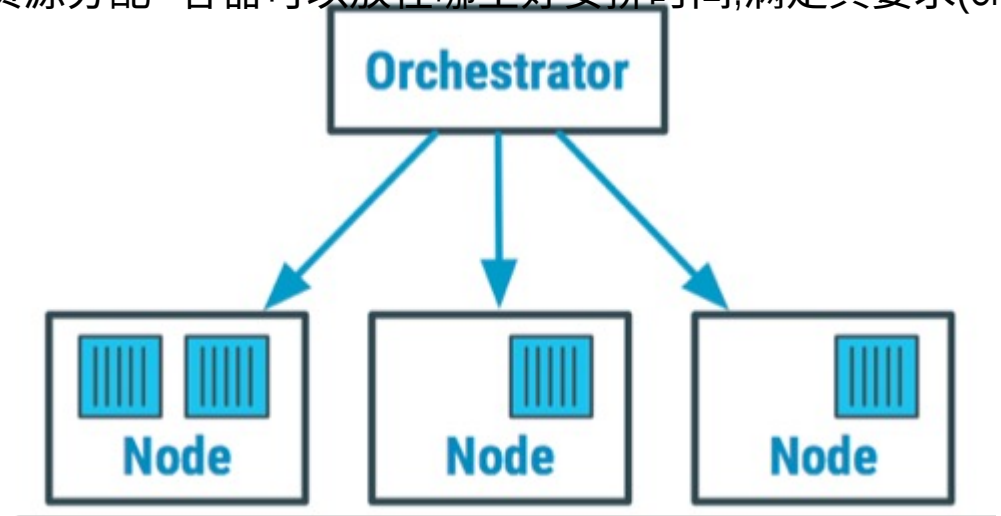
易于手动管理,最多可管理 100 个容器和 10 个节点

Orchestrator 编排器

Manage and organize both hosts and docker containers running on a cluster 管理和组织集群上运行的主机和 Docker 容器

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

主要问题 - 资源分配 - 容器可以放在哪里好安排时间,满足其要求(CPU/RAM/磁盘)+如何跟踪节点和规模



Some orchestrator tasks

任务包括：

Manage networking and access

管理网络和访问

Track state of containers

追踪容器状态

Scale services

扩展服务

Do load balancing

进行负载平衡

Relocation in case of unresponsive host

主机无响应时重新定位

Service discovery

发现服务

Attribute storage to containers

将存储属性归于容器

…

Kubernetes

What is Kubernetes?

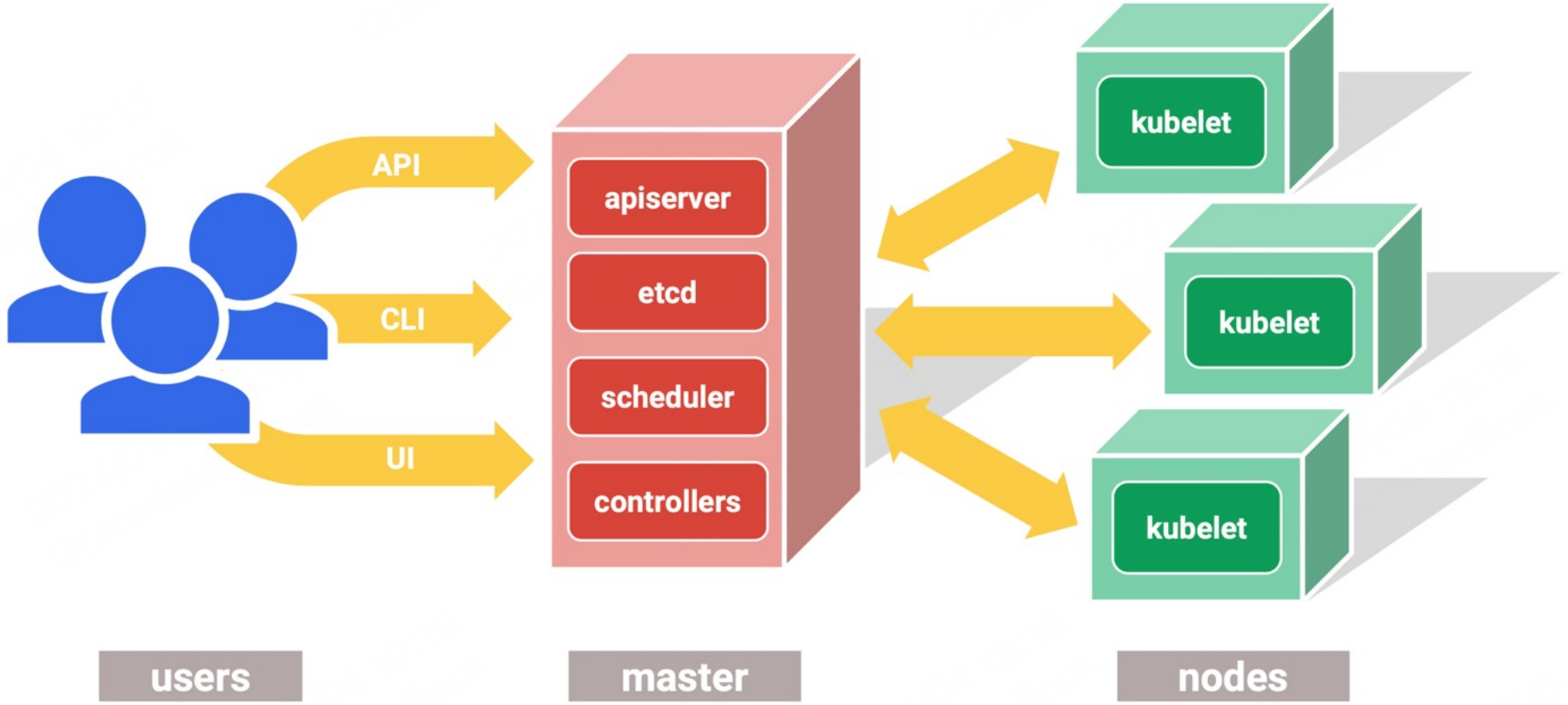
= A Production-Grade Container Orchestration System
生产级容器编排系统

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.
由 Google 开发的基于Borg和Omega 的系统,目前在 Google 内部运行,并且已被证明在 Google 运行了 10 多年。

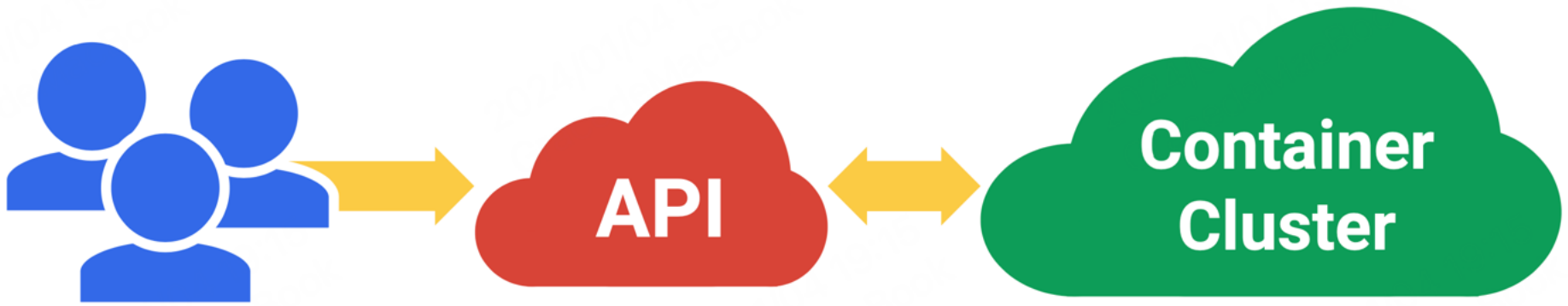
Google spawns billions of containers per week with these systems.
谷歌每周利用这些系统生成数十亿个容器。

Created by three Google employees initially during the summer of 2014; grew exponentially and became the first project to get denoted to CNCF.
最初由三名 Google 员工于 2014 年夏天创建;随后迅速发展并成为第一个获得 CNCF 批准的项目。

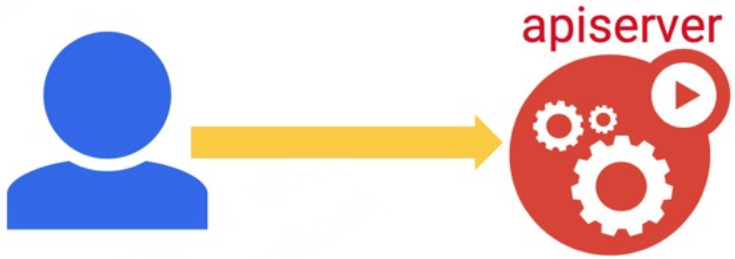
The 10000 foot view



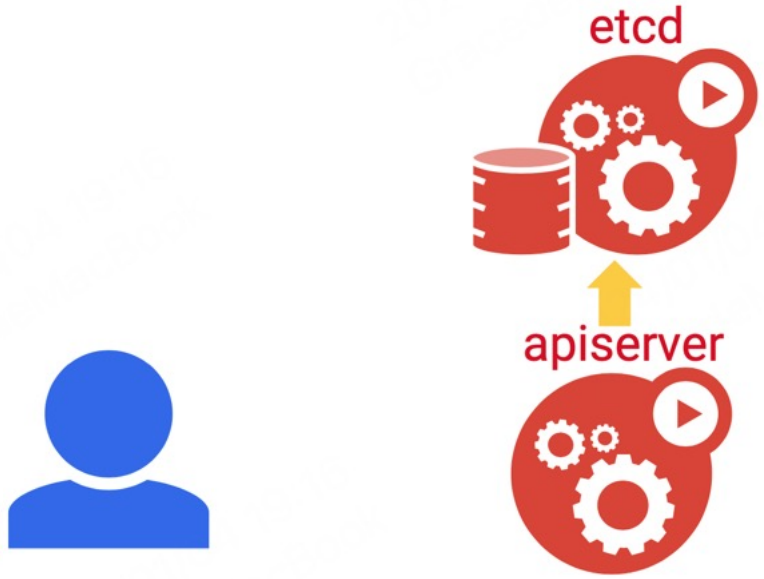
All you really care about



Running a container



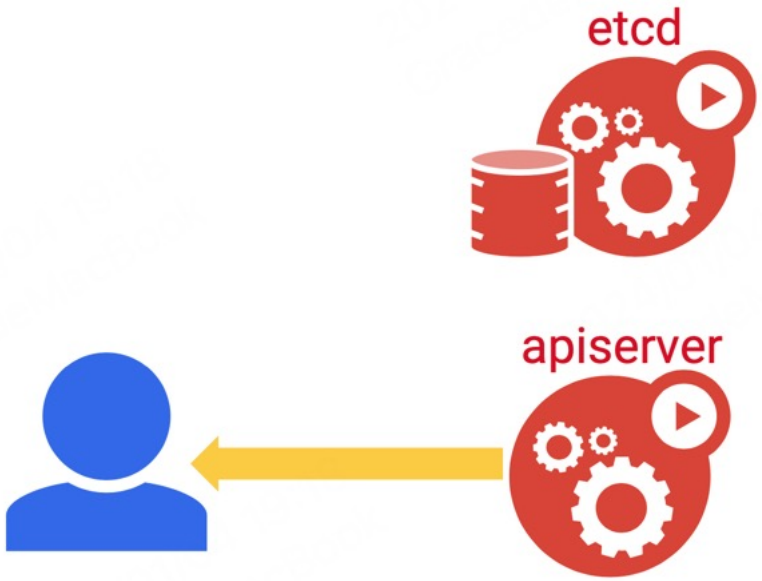
Running a container



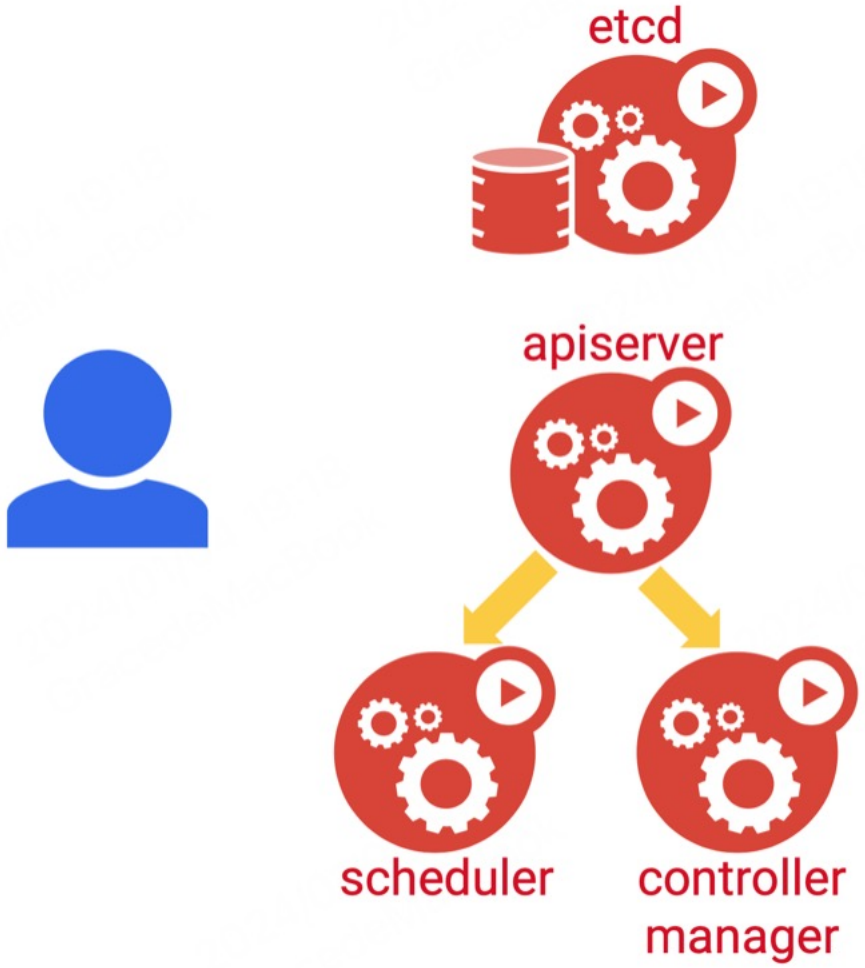
etcd: a consistent and highly-available key value store used as K8s' backing store for all cluster data

etcd:一个一致且高度可用的键值存储，用作 K8s 所有集群数据的后备存储

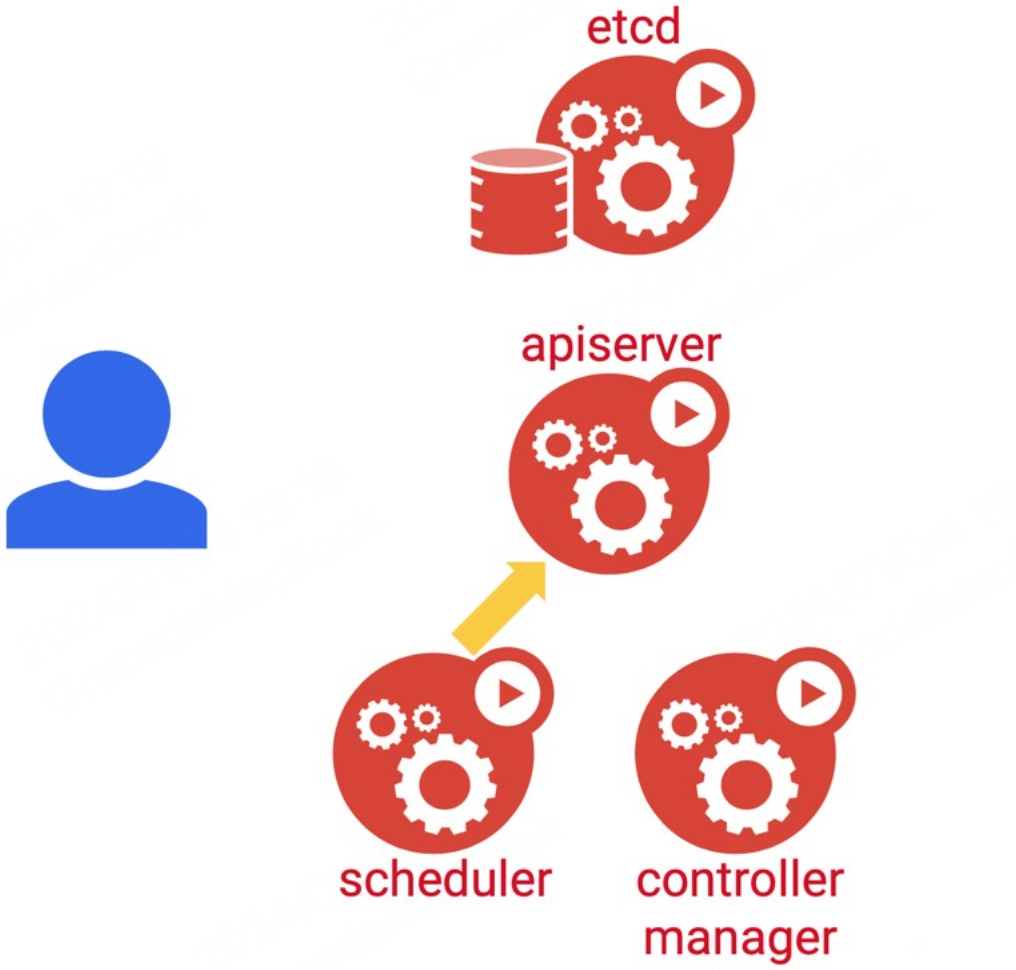
Running a container



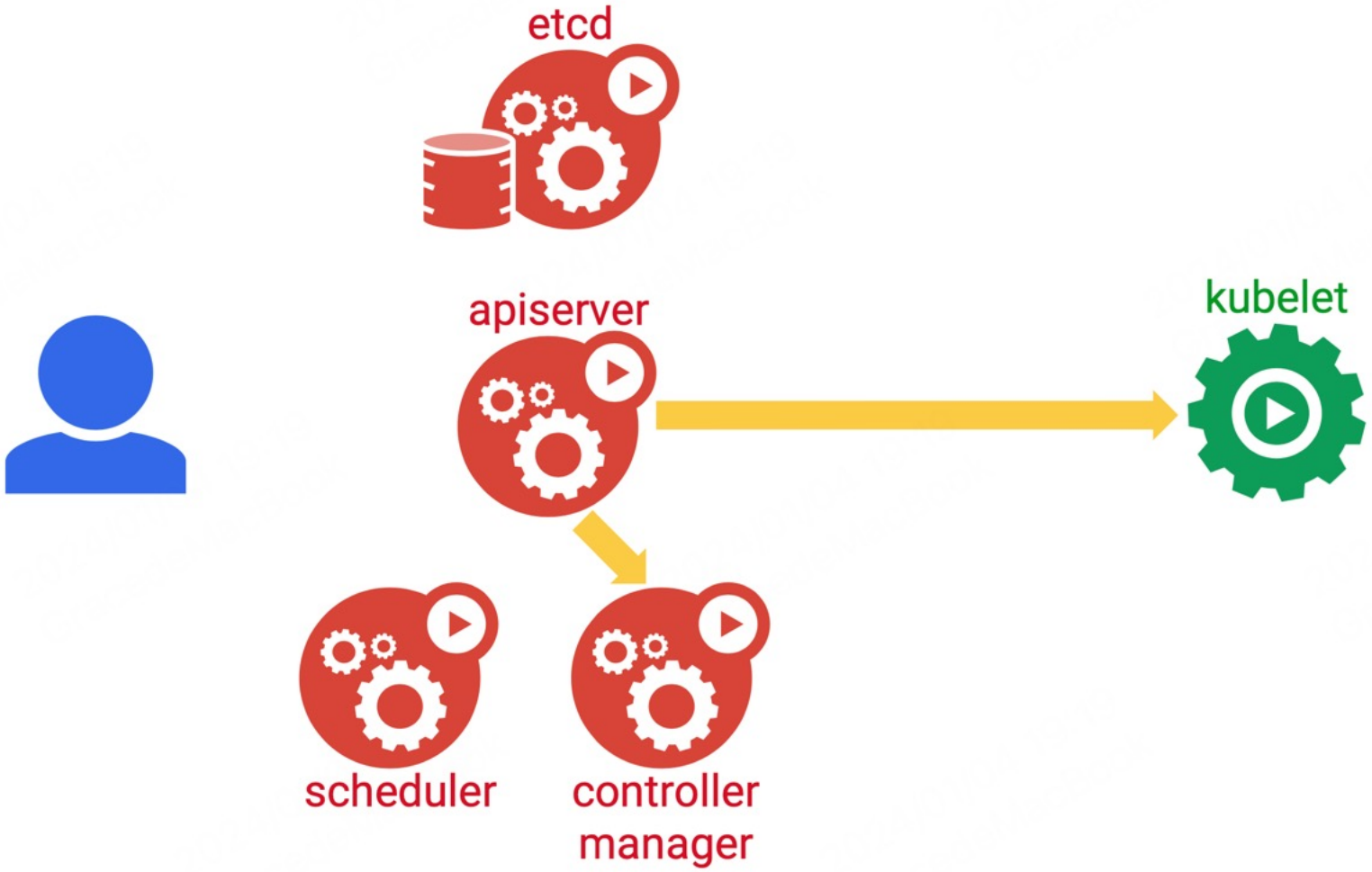
Running a container



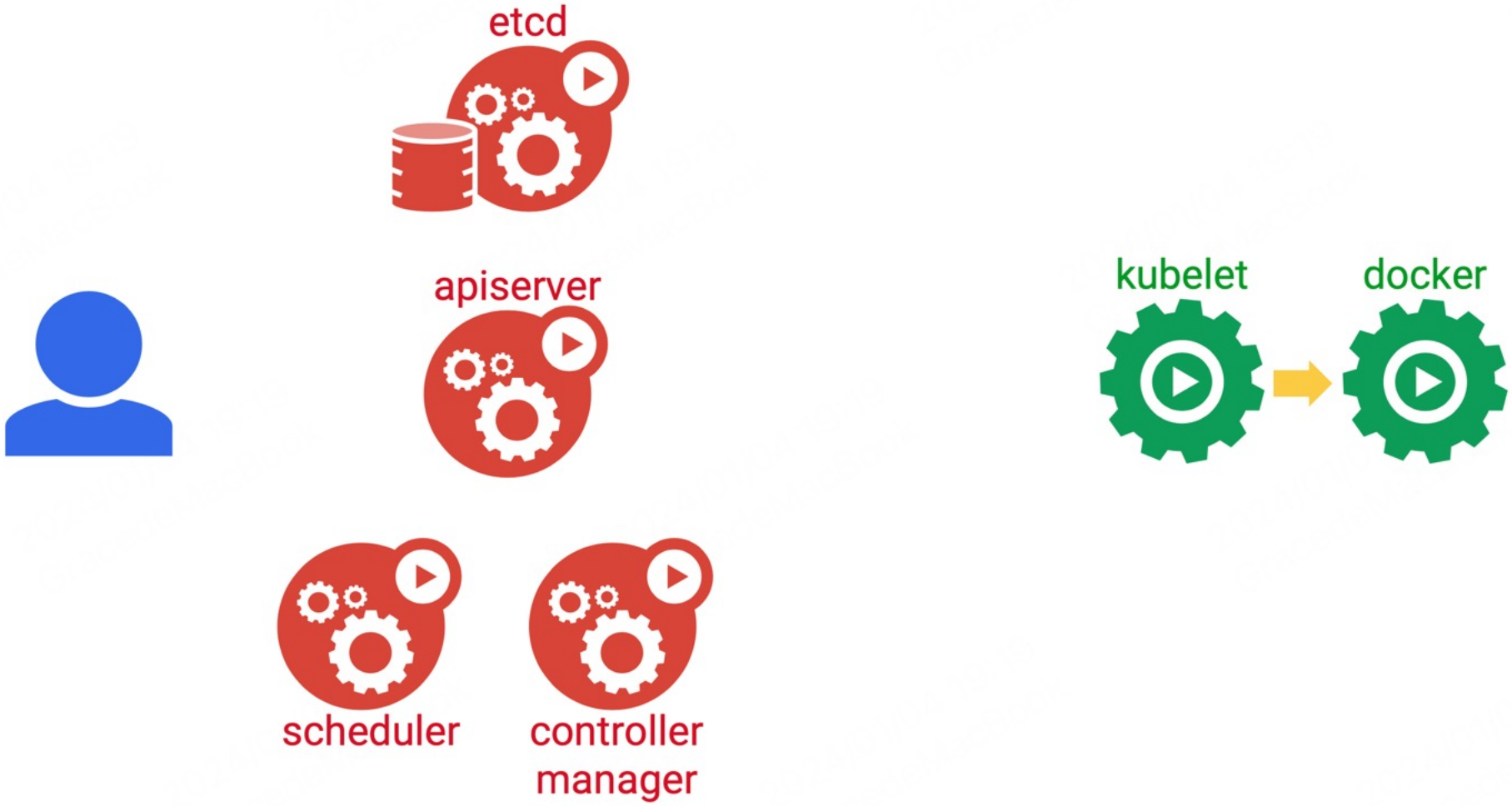
Running a container



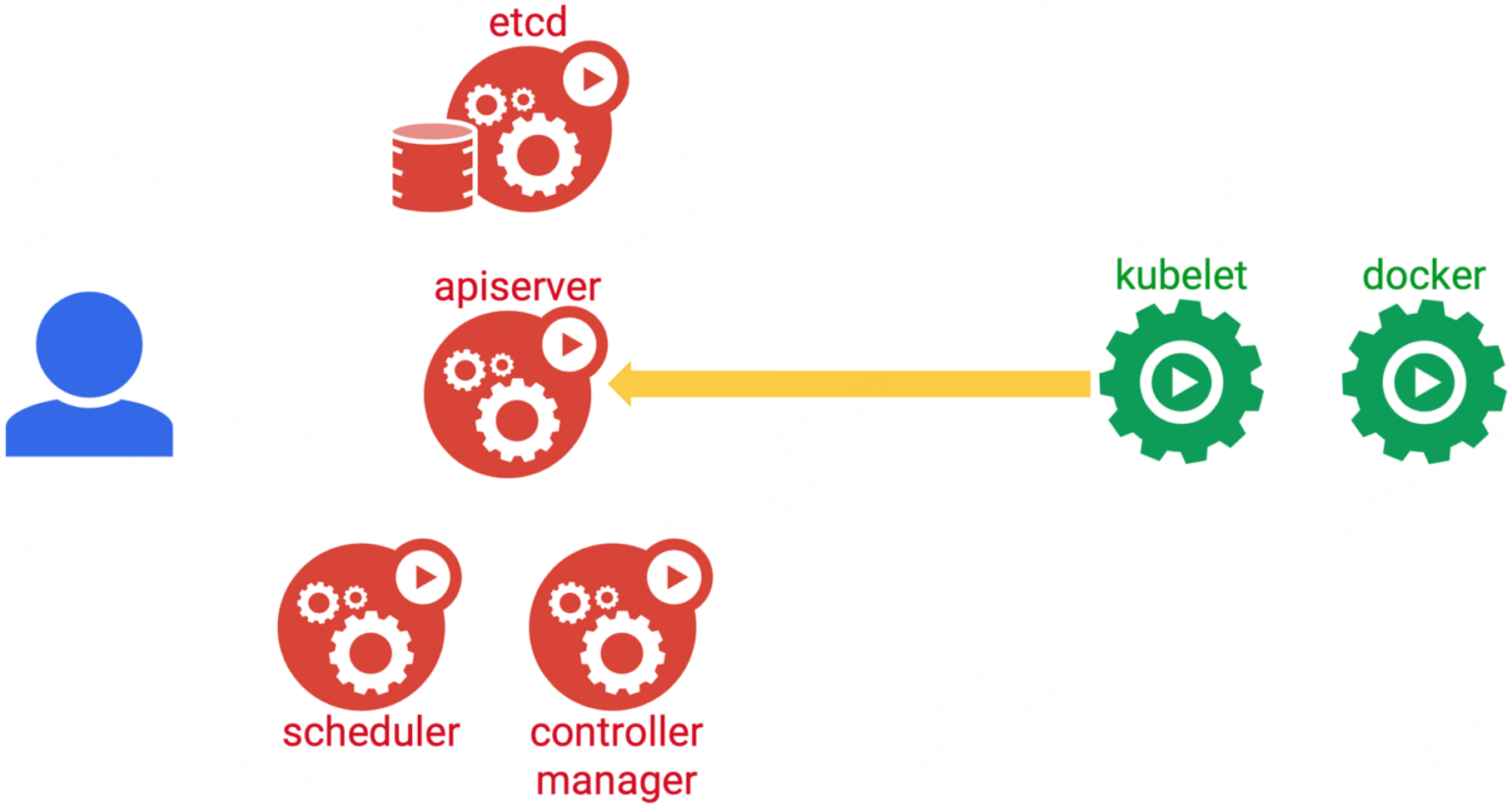
Running a container



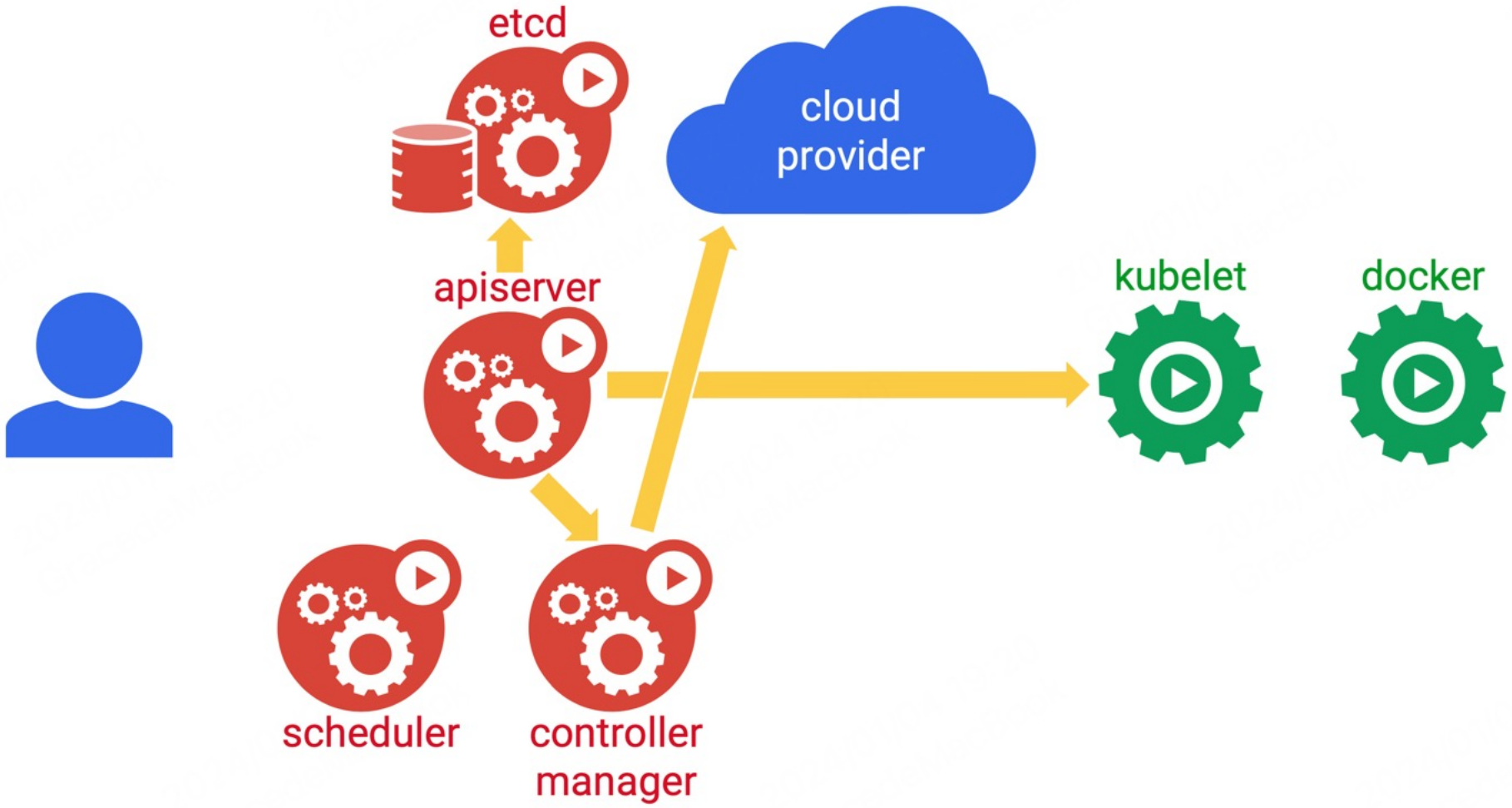
Running a container



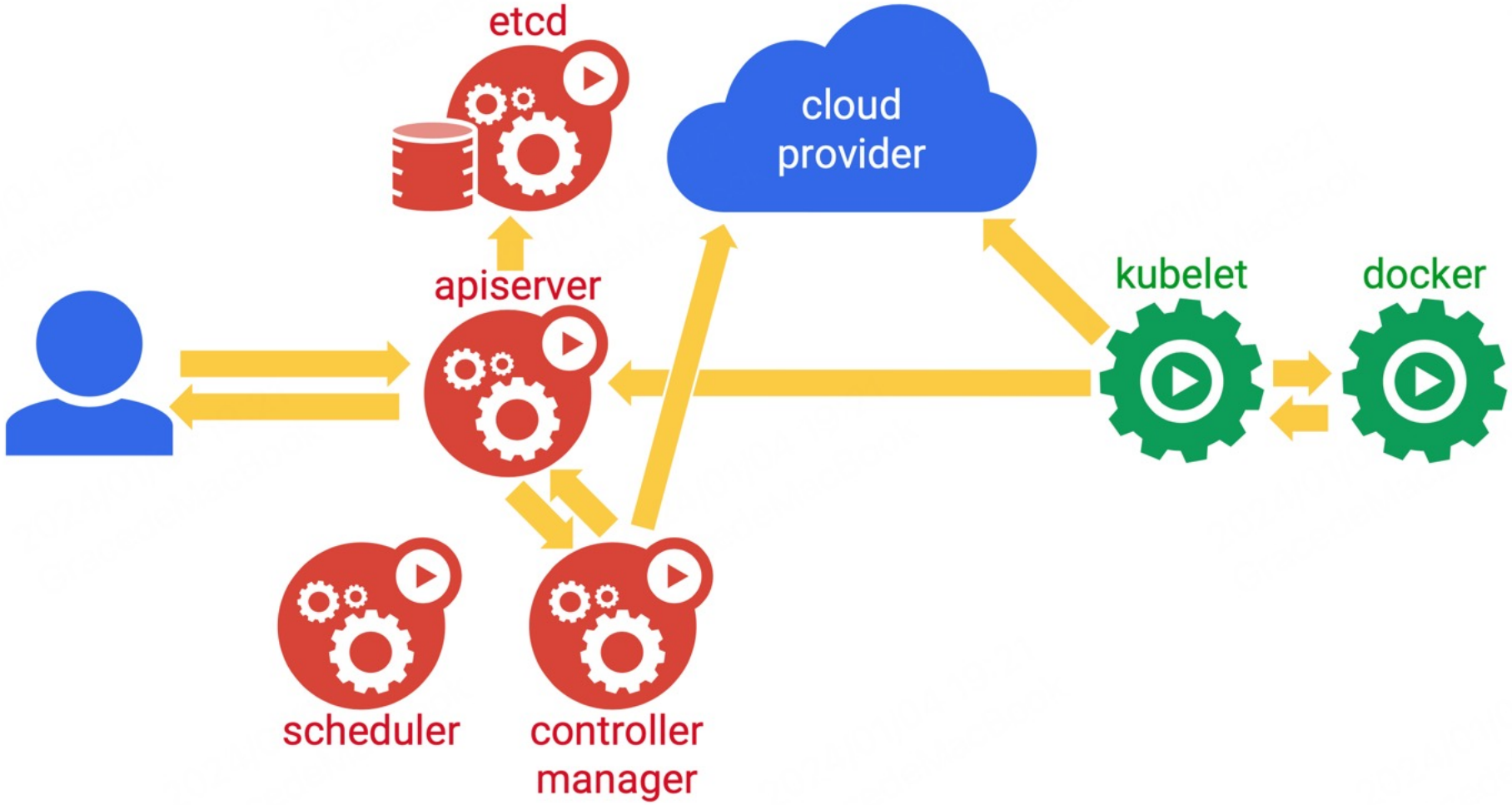
Running a container



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)　　不可以直接在 Kubernetes 中部署容器

Why not?

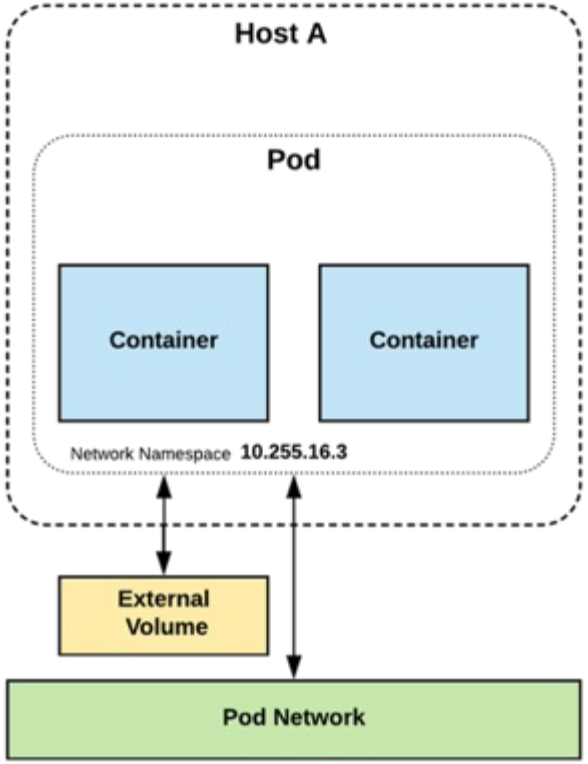
- Because the smallest deployable unit of computing is not a container, but a **pod**　　由于可部署的最小计算单元不是容器,而是pod

Pods

Atomic unit or smallest “unit of work” of Kubernetes K8S的最小工作单位

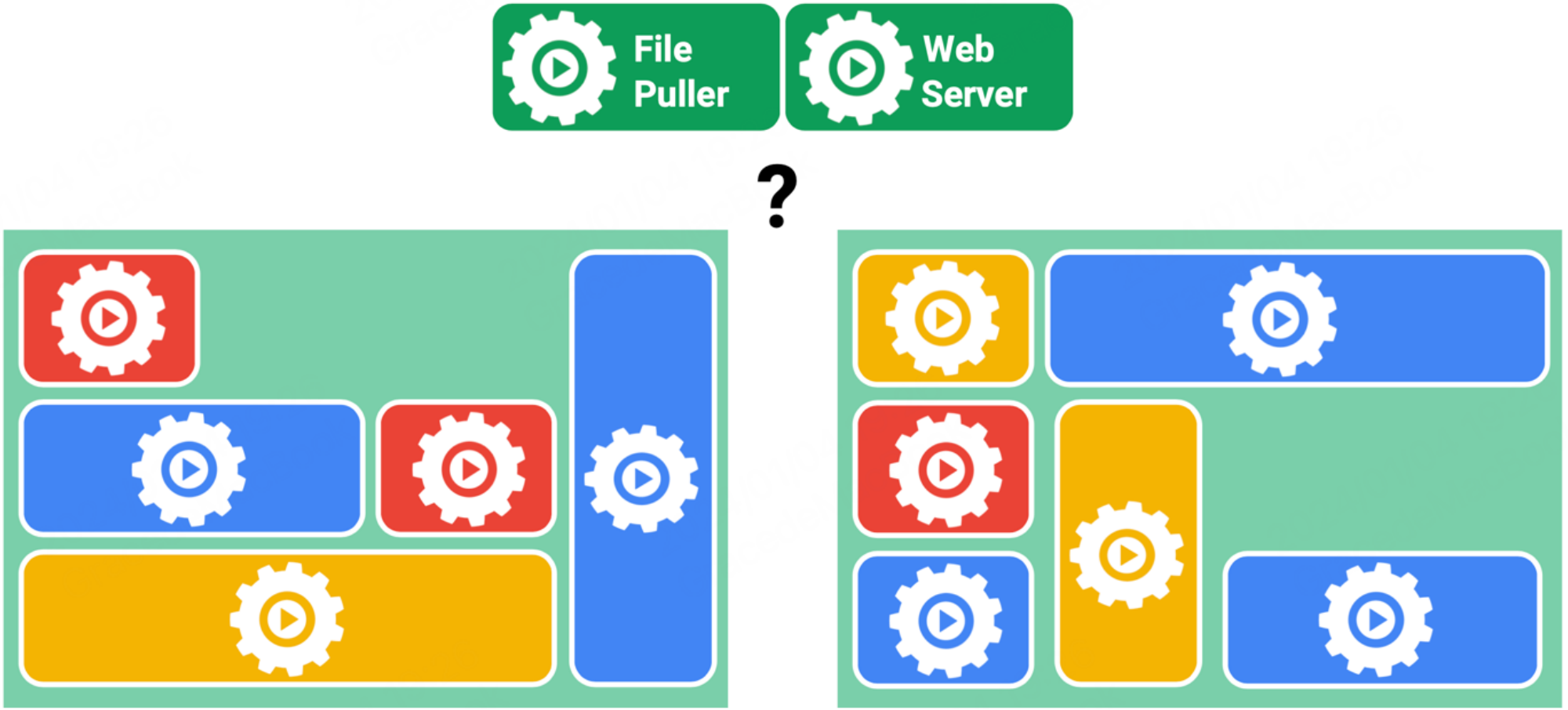
Pods are **one or MORE containers** that share volumes and namespace Pod 是一个或多个容器共享卷和命名空间

They are also ephemeral! 短暂存在

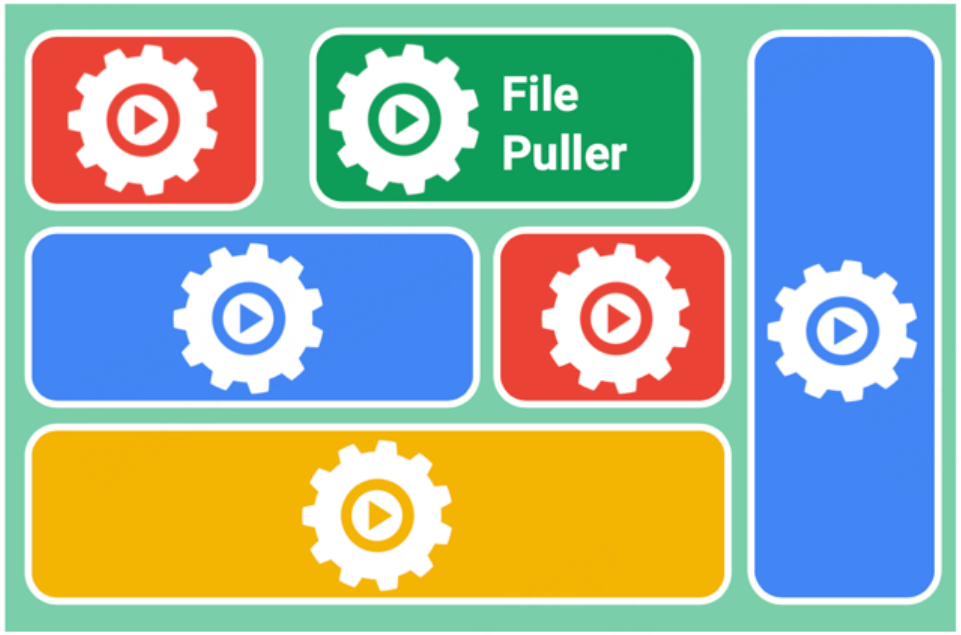


So, why a pod and not container directly?

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

对于需要始终保持在一起的一组共生容器,采用全有或全无方法

Pod considered running if all containers are scheduled and running

如果所有容器都已安排并正在运行,则认为 Pod 正在运行

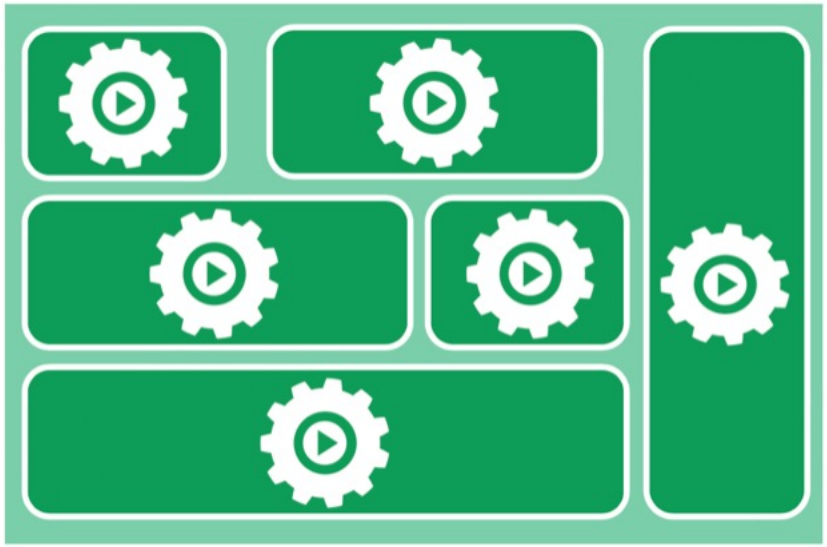
Can you deploy a container in Kubernetes?

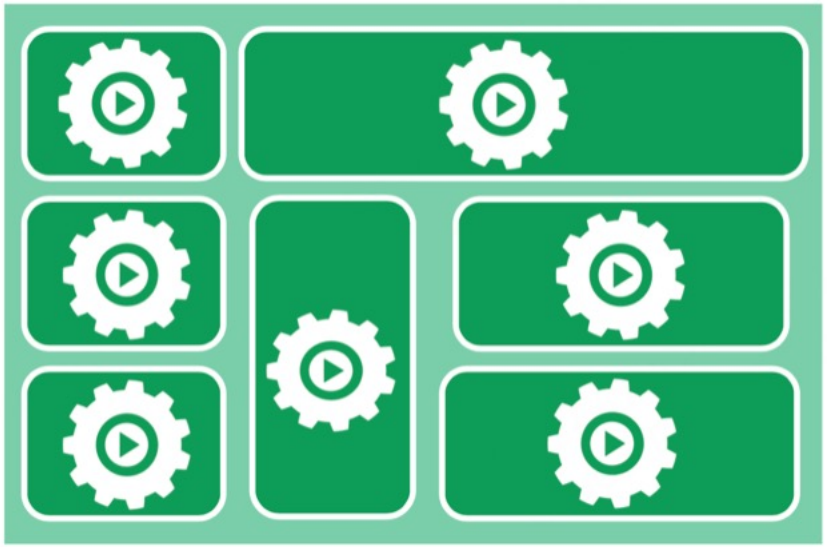
- Yes, inside a pod!

能在 Kubernetes的pod中部署容器

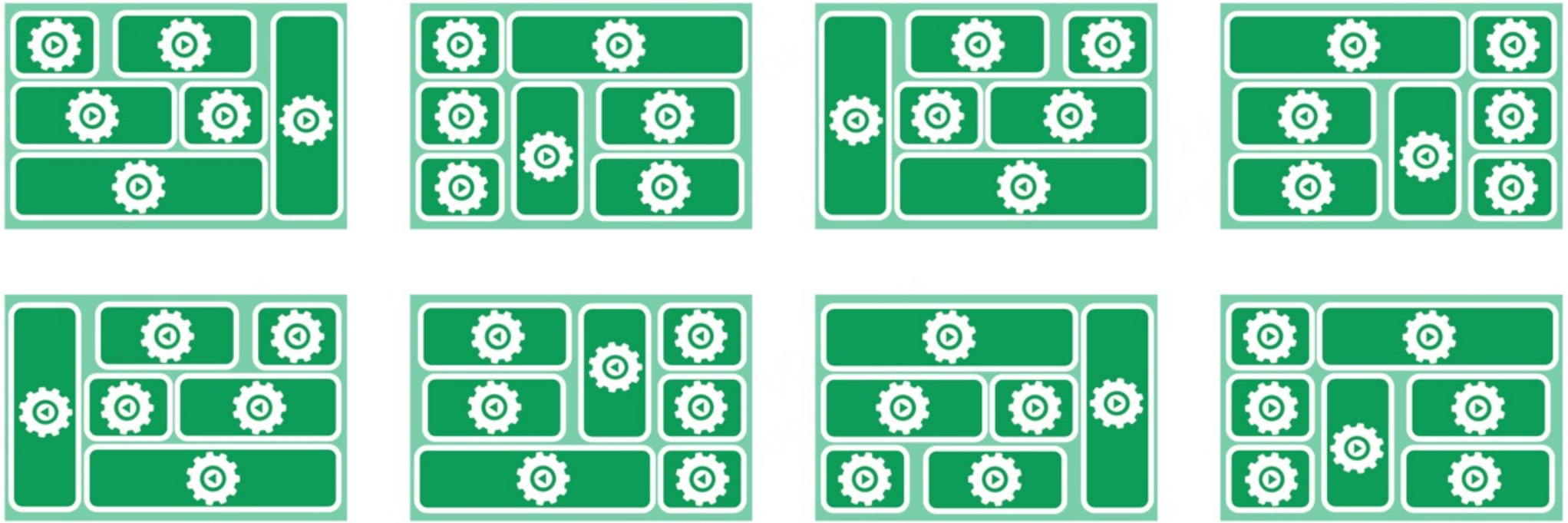
Finding things

Physical view

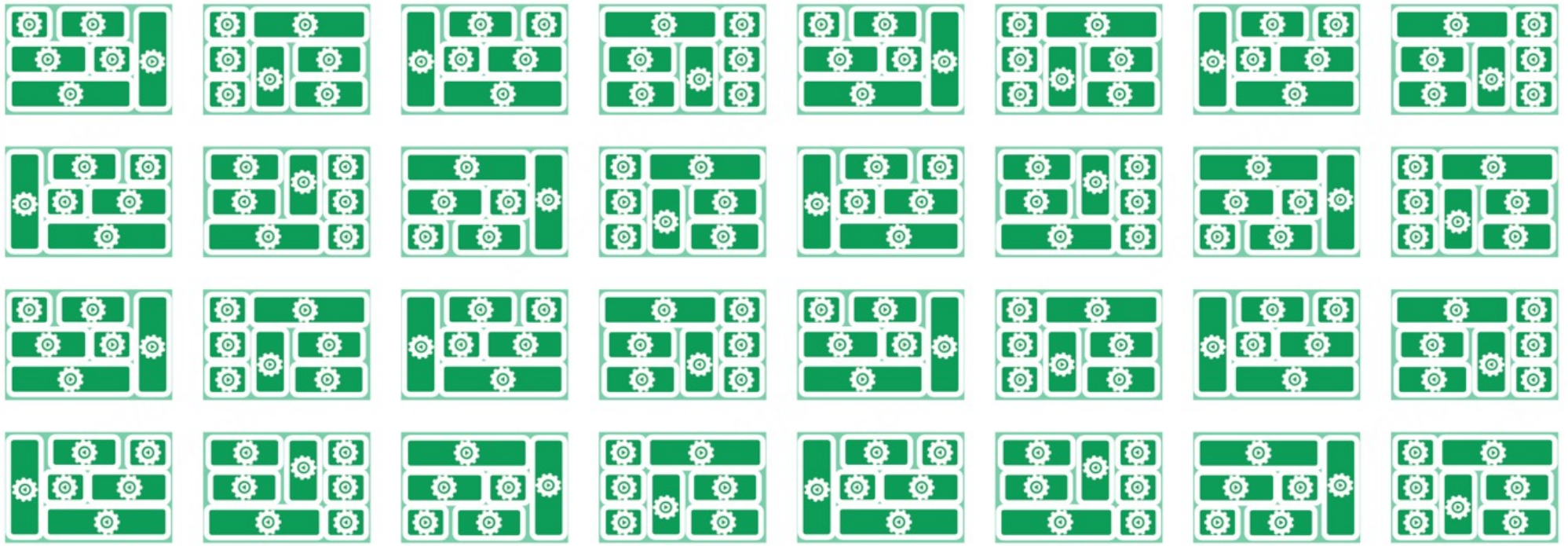




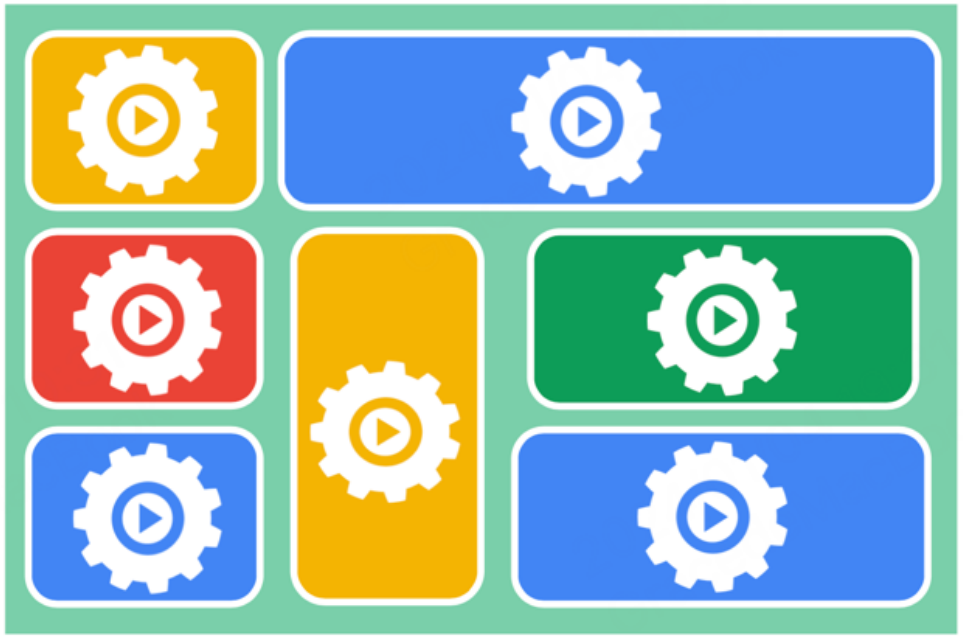
Physical view



Physical view



Logical view



Labels and selectors

Arbitrary metadata

任意元数据

Attached to any API object

附加到任何 API对象

Generally represent identity

一般代表身份

Queryable by selectors

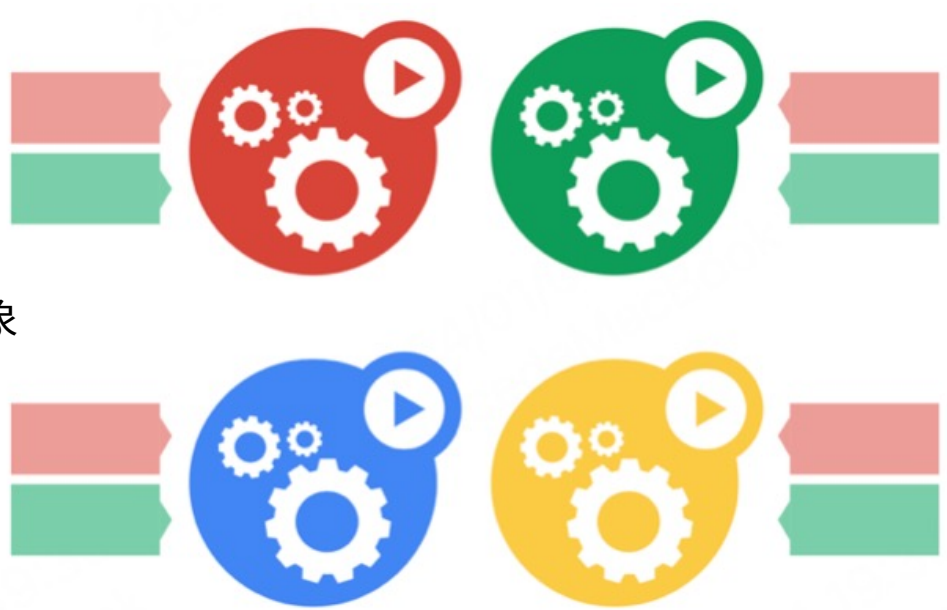
可通过选择器查询

- Think SQL ‘select … where …’

类似于 SQL

The only group mechanism

唯一的群组机制



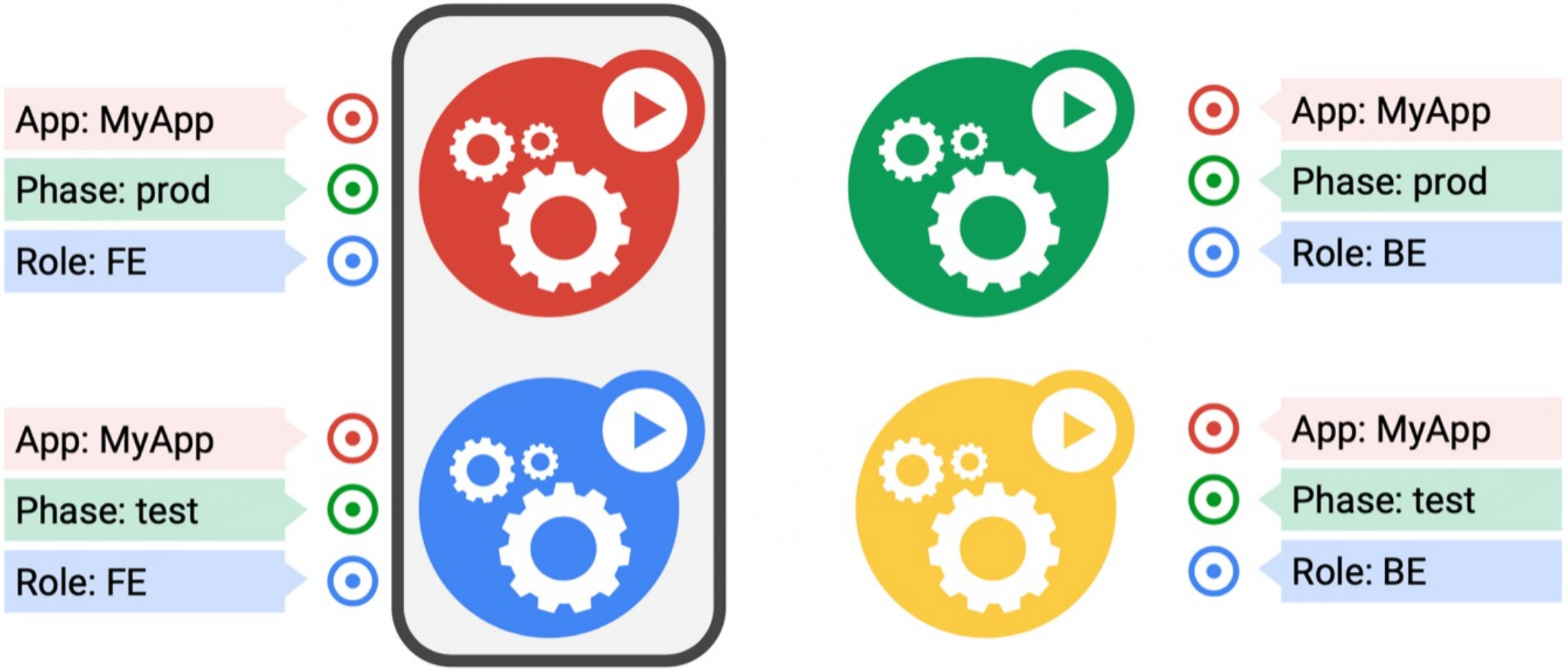
Selectors


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

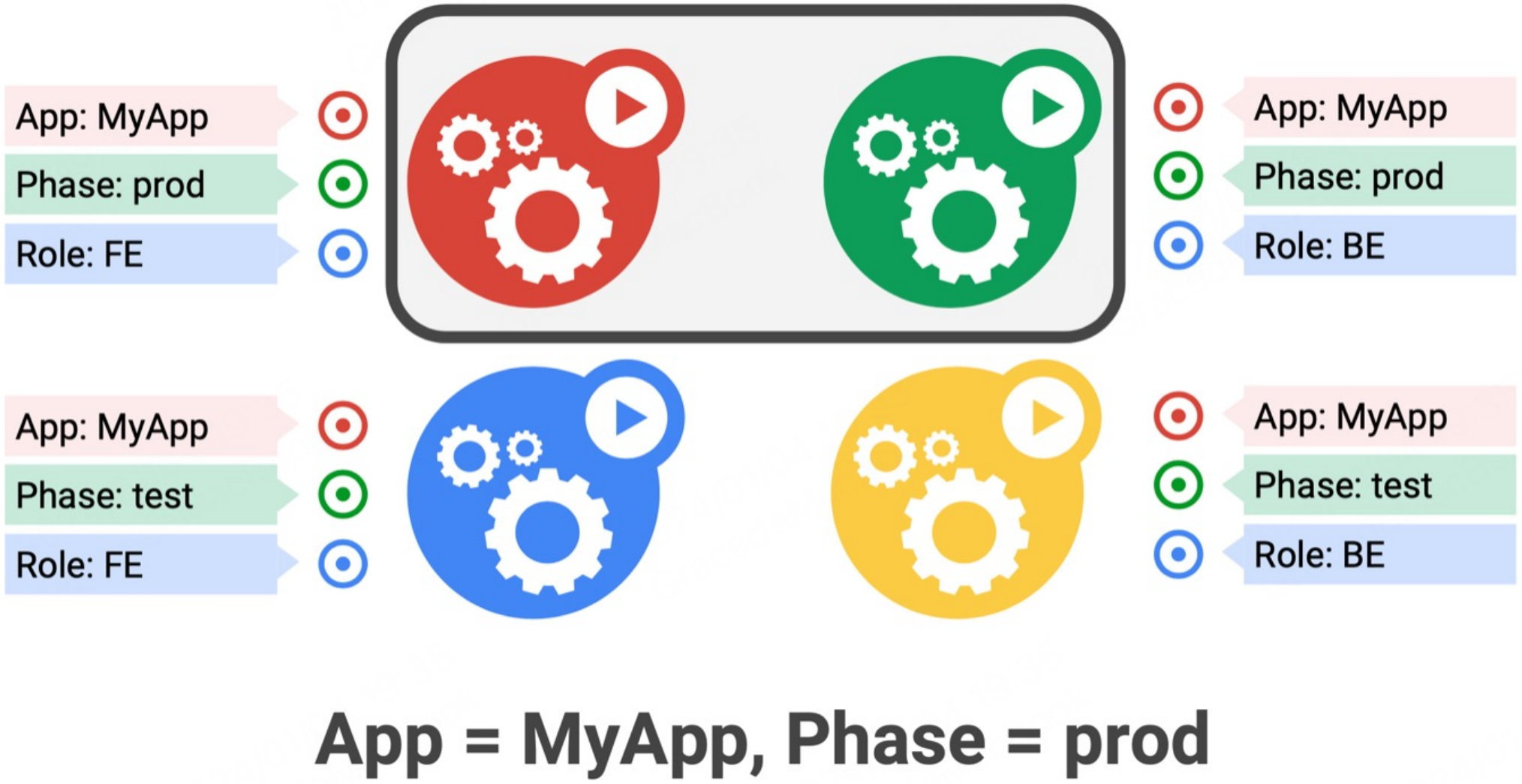
Selectors


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

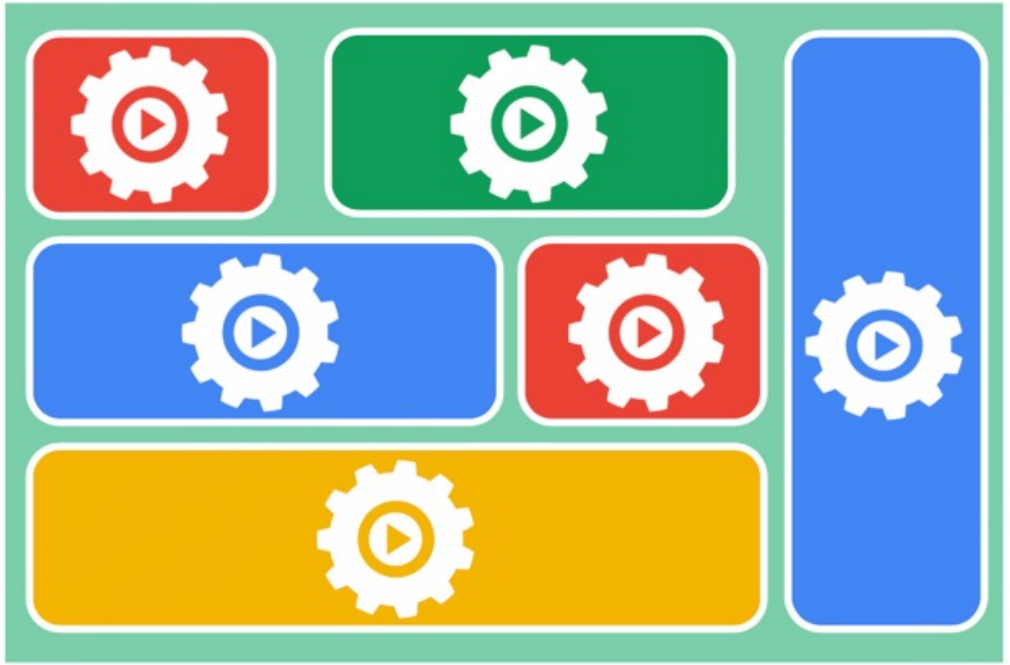
Selectors

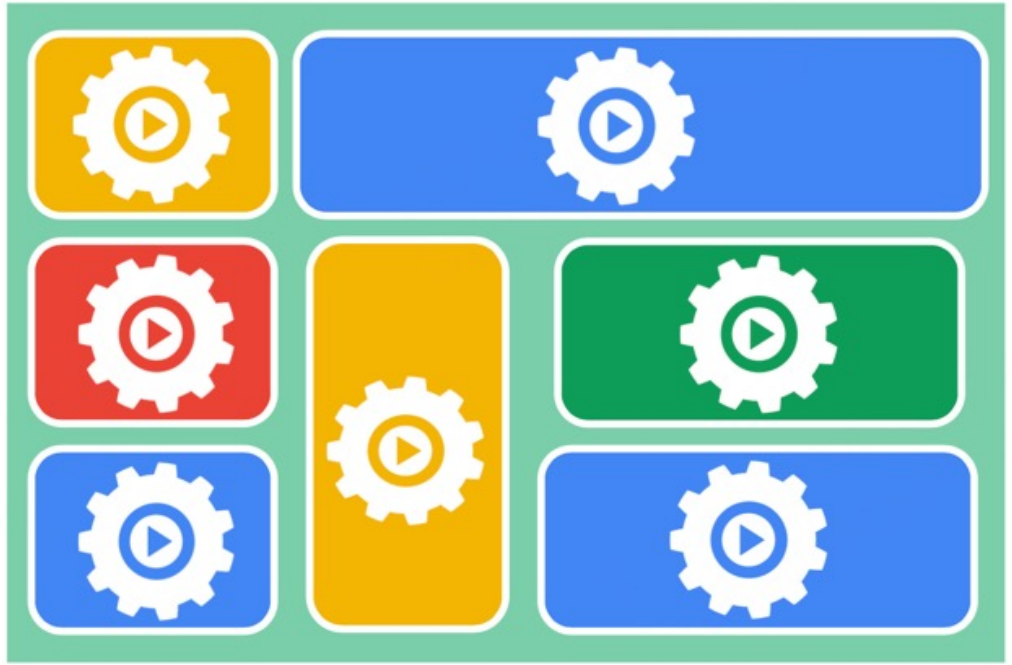

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE
App = MyApp, Phase = prod

Selectors

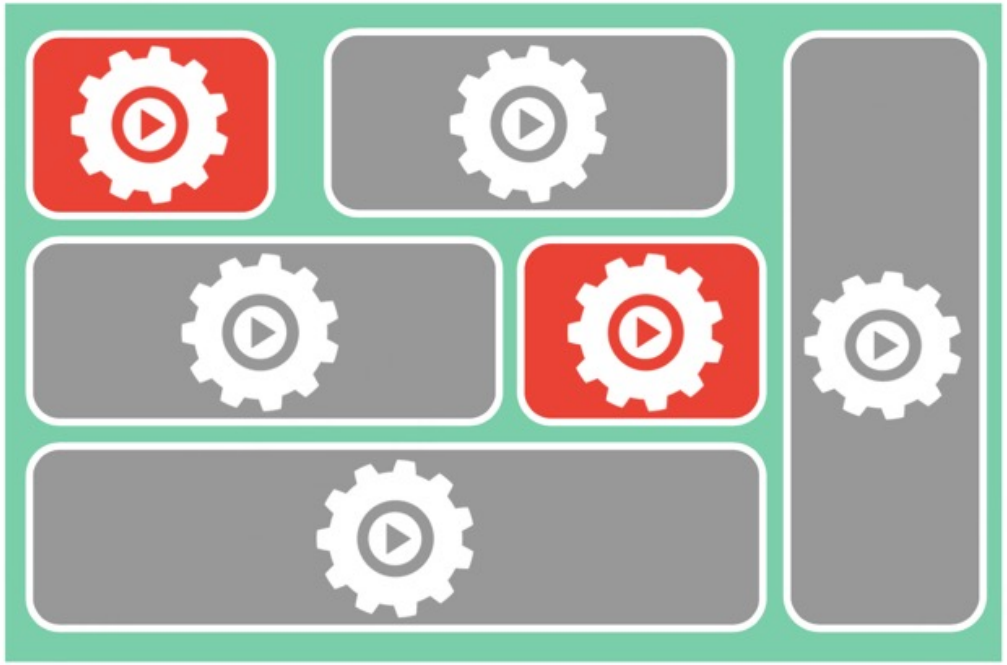

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE
App = MyApp, Phase = test

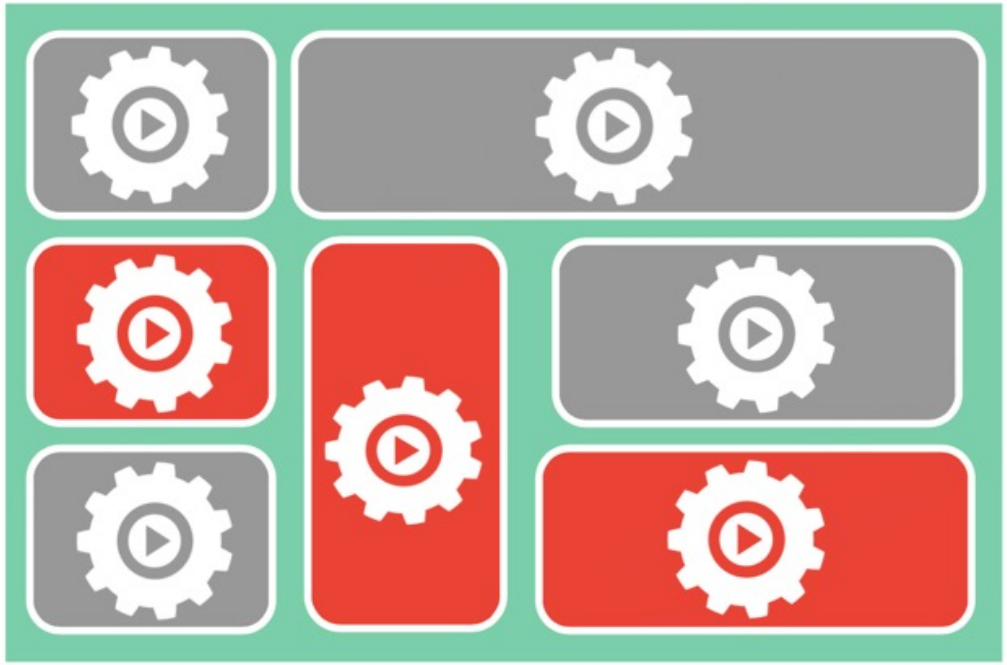
Logical view



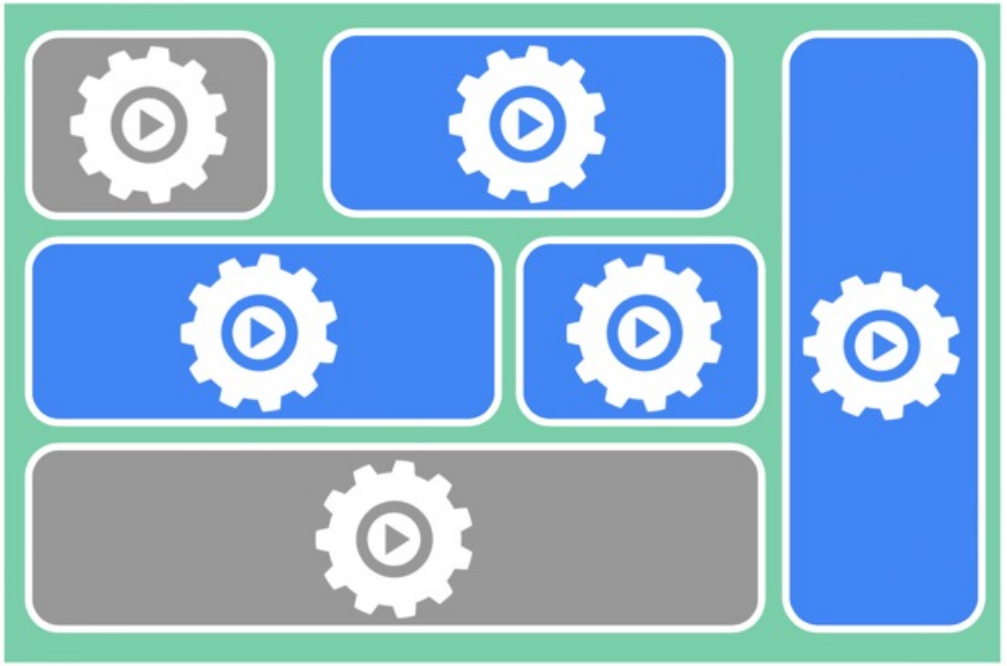


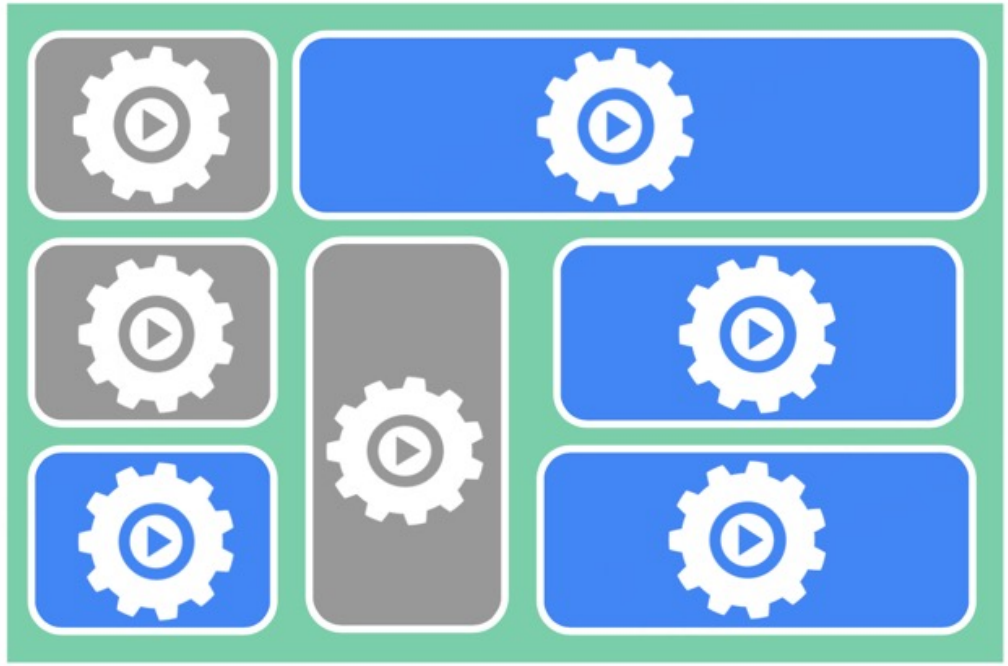
Logical view



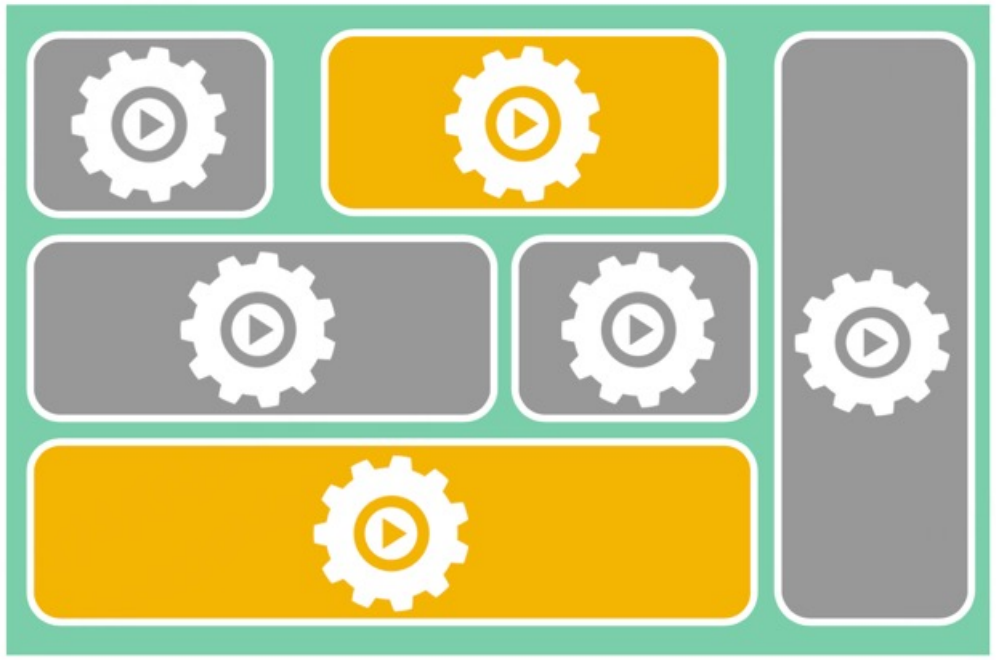


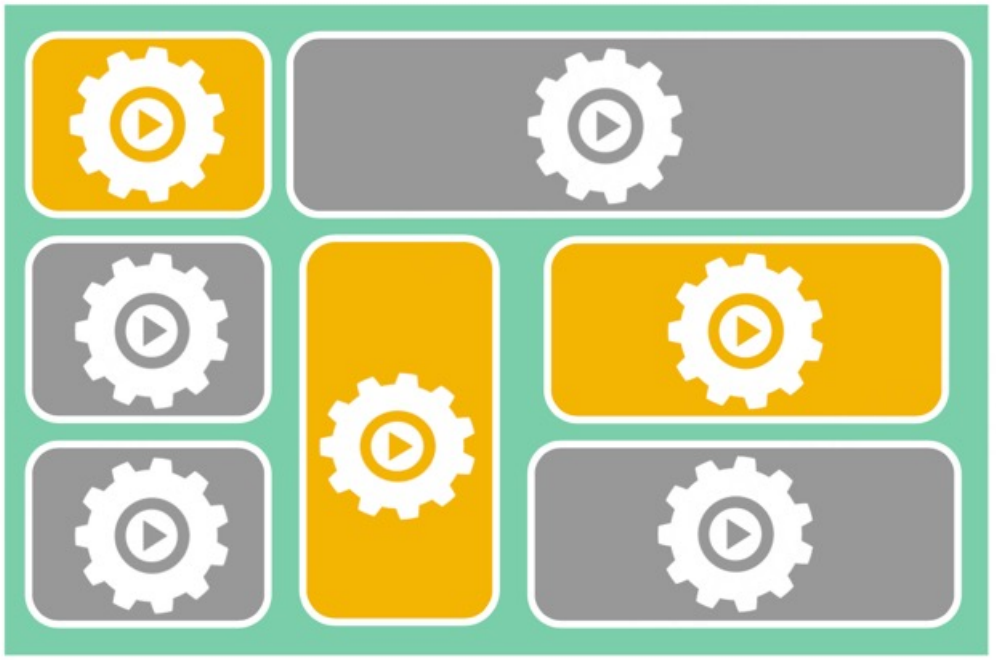
Logical view





Logical view





Discovery

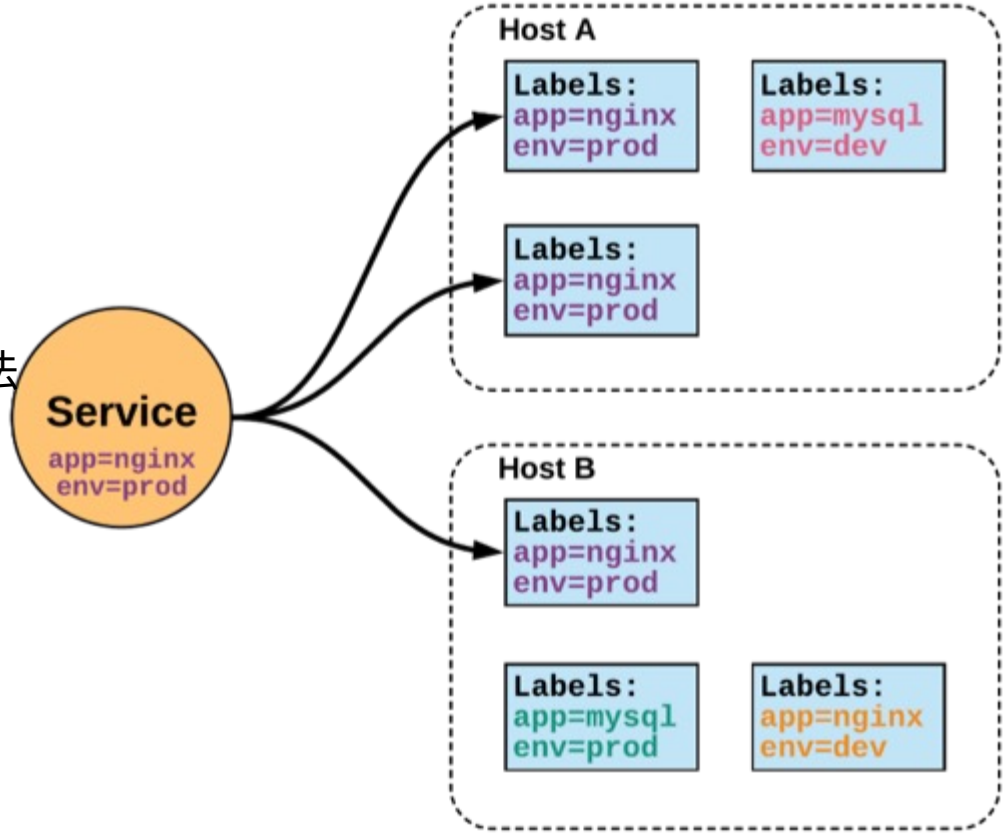
Services 服务

Unified method of accessing the exposed workloads of Pods 访问 Pod 暴露工作负载的统一方法

Durable resource

- Static cluster IP 耐用资源 静态集群 IP 静态命名空间 DNS 名称
- Static namespaced DNS name

NOT Ephemeral! 长期存在



Services

ClusterIP (default)	ClusterIP(默认)
NodePort	节点端口
LoadBalancer	负载均衡器
ExternalName	外部名称

Workloads

ReplicaSet	副本集
Deployment	部署
DaemonSet	守护程序集
StatefulSet	StatefulSet
Job	工作
CronJob	CronJob
…	

ReplicaSet

副本集

Primary method of managing pod replicas and their lifecycle

管理 pod 副本及其生命周期的主要方法

Includes their scheduling, scaling, and deletion

包括其调度、扩展和删除

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

他们的工作很简单:始终确保所需数量的 Pod 正在运行

ReplicaSet

replicas: The desired number of instances of the Pod.

replicas:所需的 Pod 实例数量。

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

选择器: ReplicaSet的标签选择器将管理所有它所针对的 Pod 实例;无论是否需要。

```
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**.

通过 ReplicaSet 管理 Pod 的方式。

Provide rollback functionality and update control.

提供回滚功能和更新控制。

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

更新通过pod-template-hash标签进行管理。

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

每次迭代都会创建一个唯一的标签,该标签分配给 ReplicaSet和后续 Pod。

Deployment

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

revisionHistoryLimit:要保留的Deployment 先前迭代的次数

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

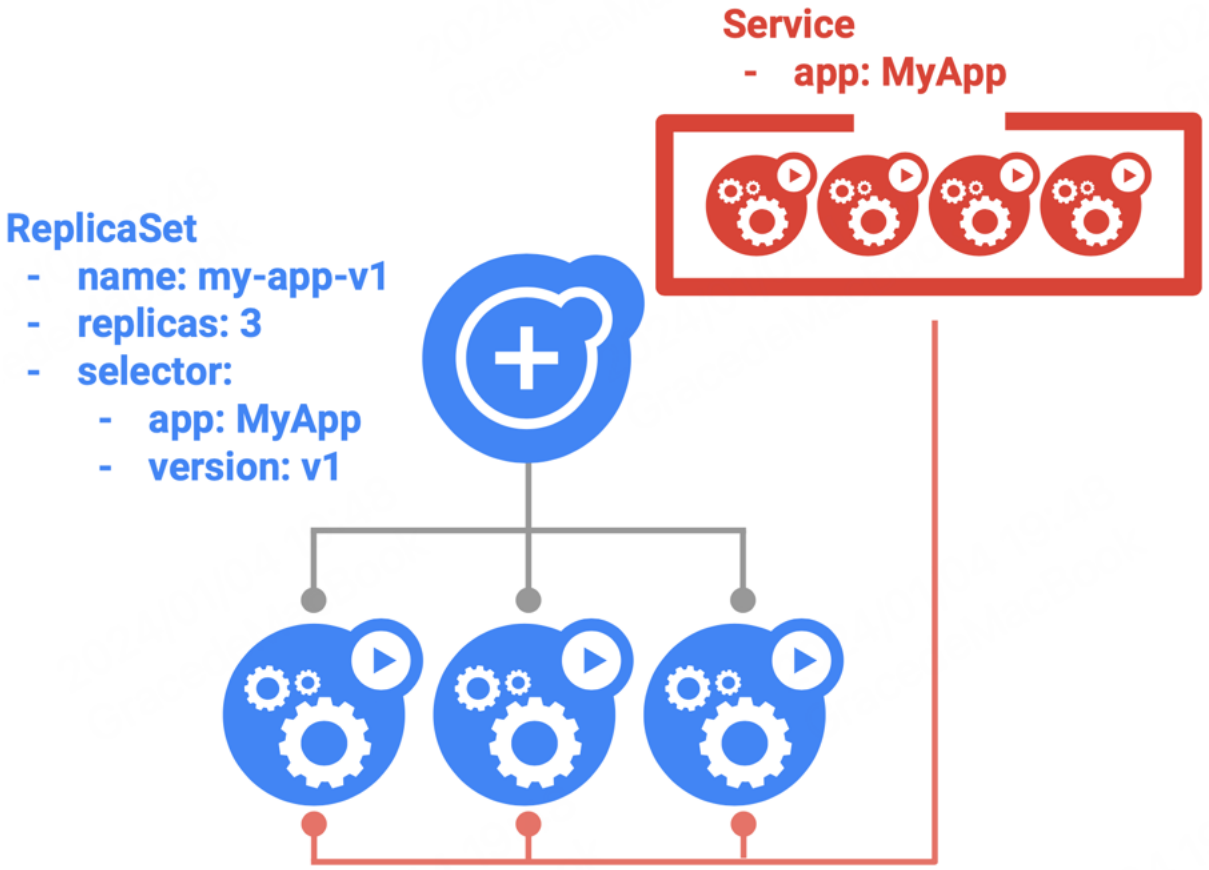
策略:描述根据类型更新Pod的方法。

- Recreate
- RollingUpdate

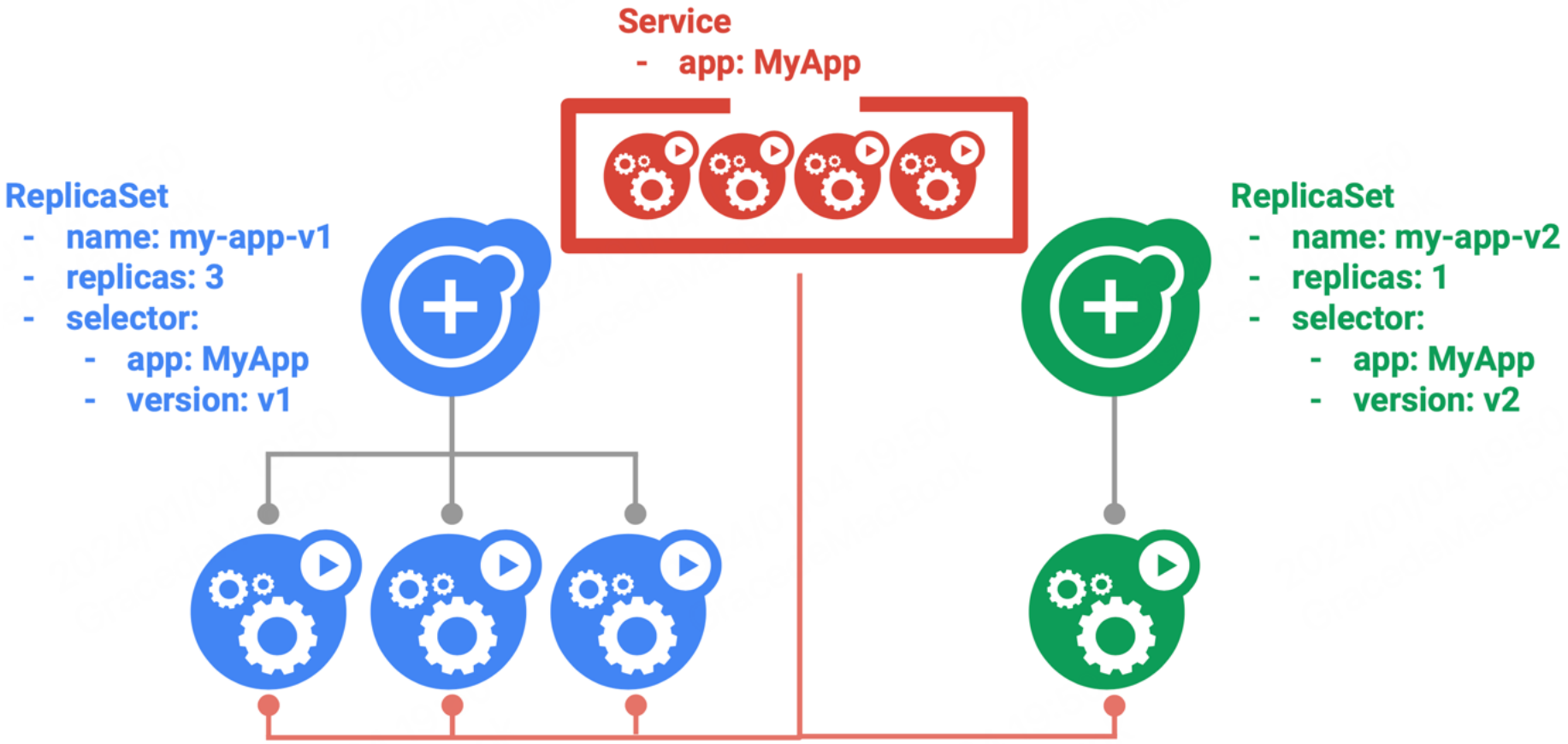
重新创建&滚动更新

```
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>
```

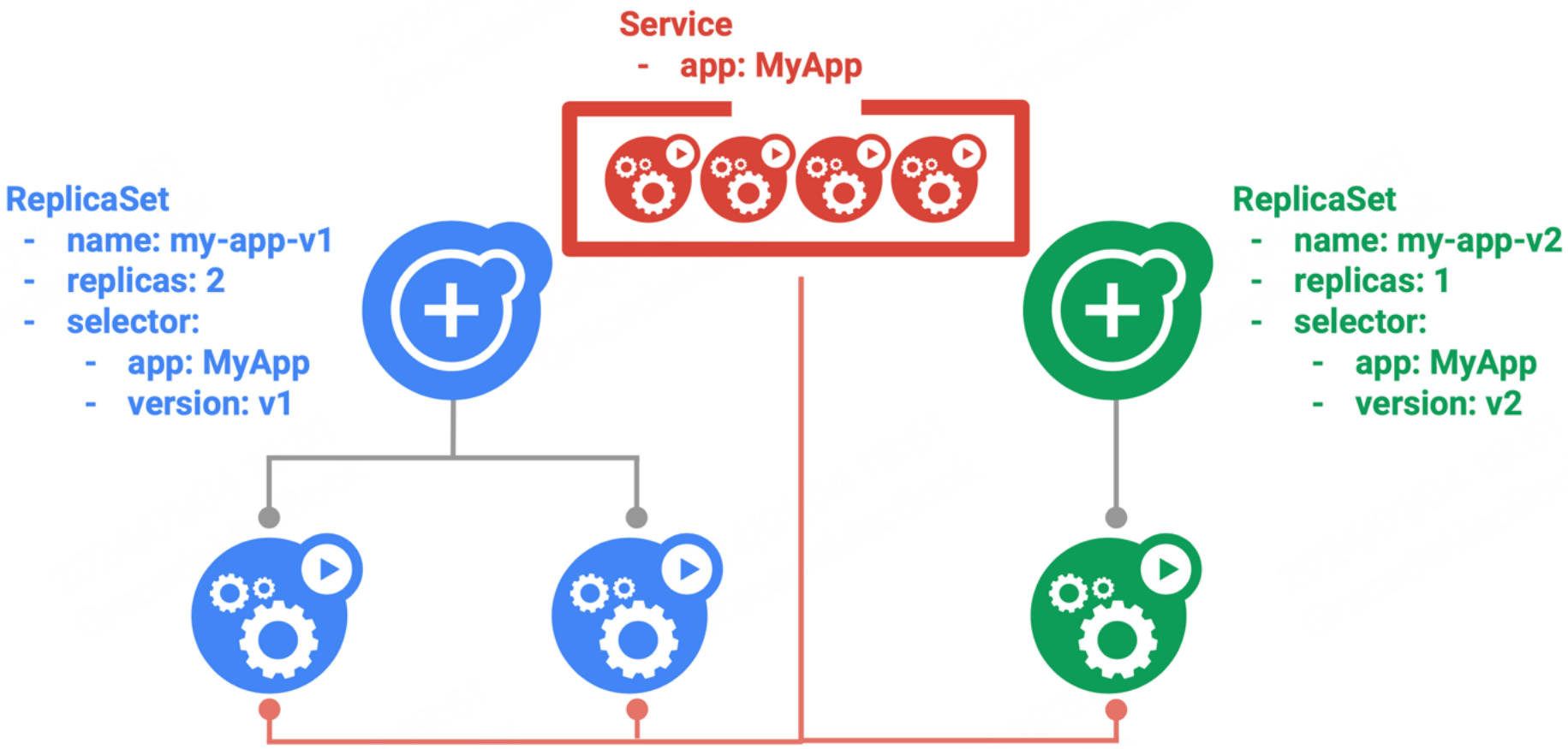
Rolling Update



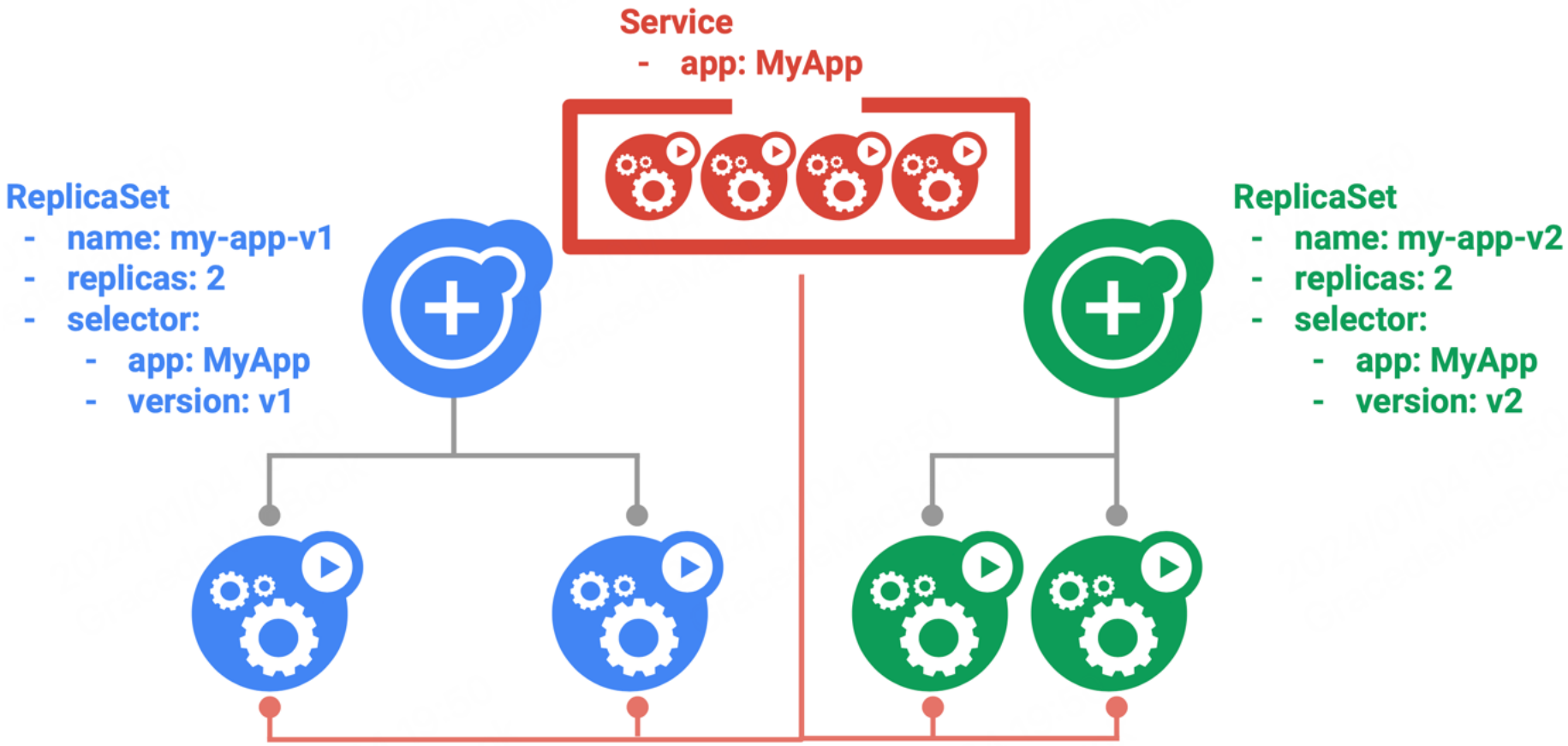
Rolling Update



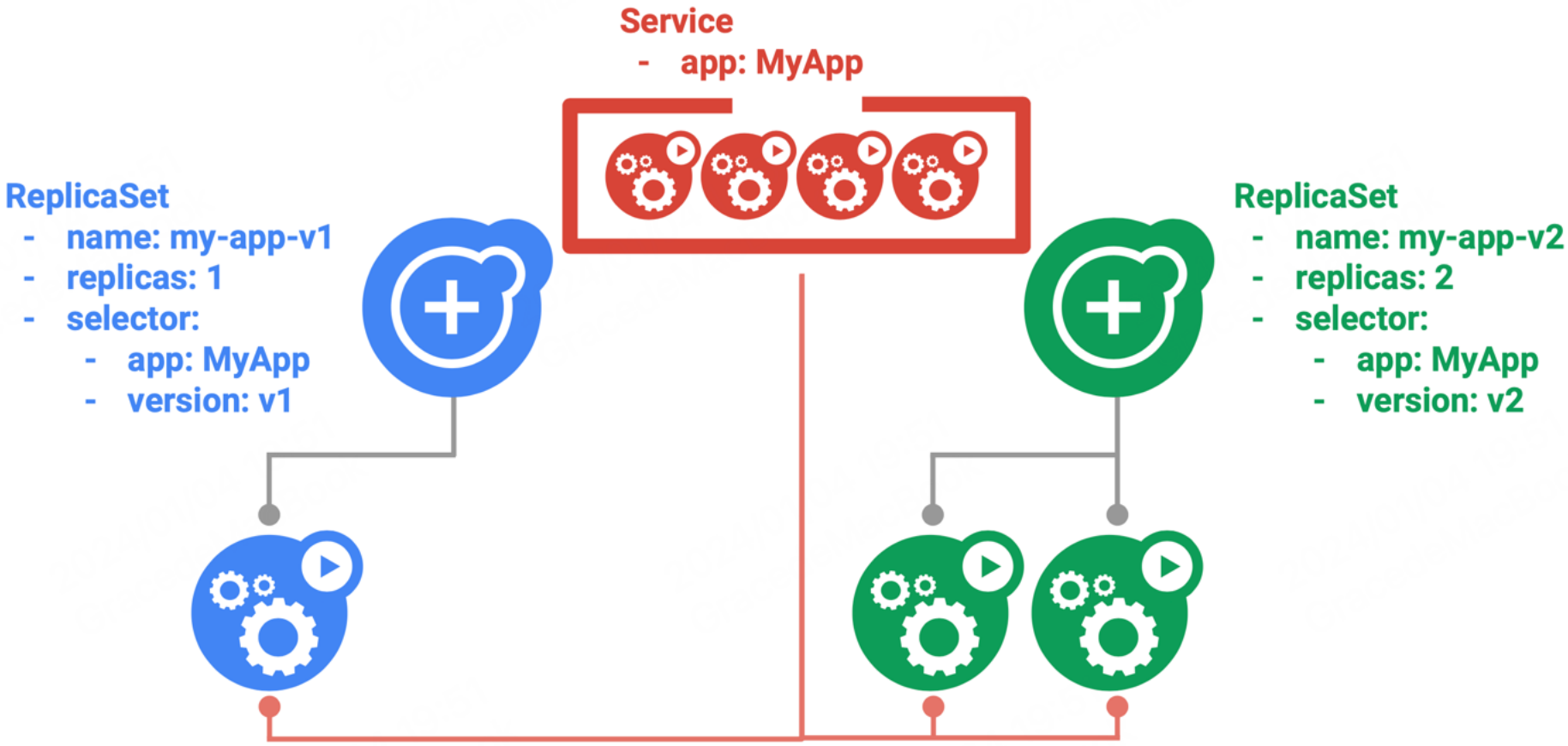
Rolling Update



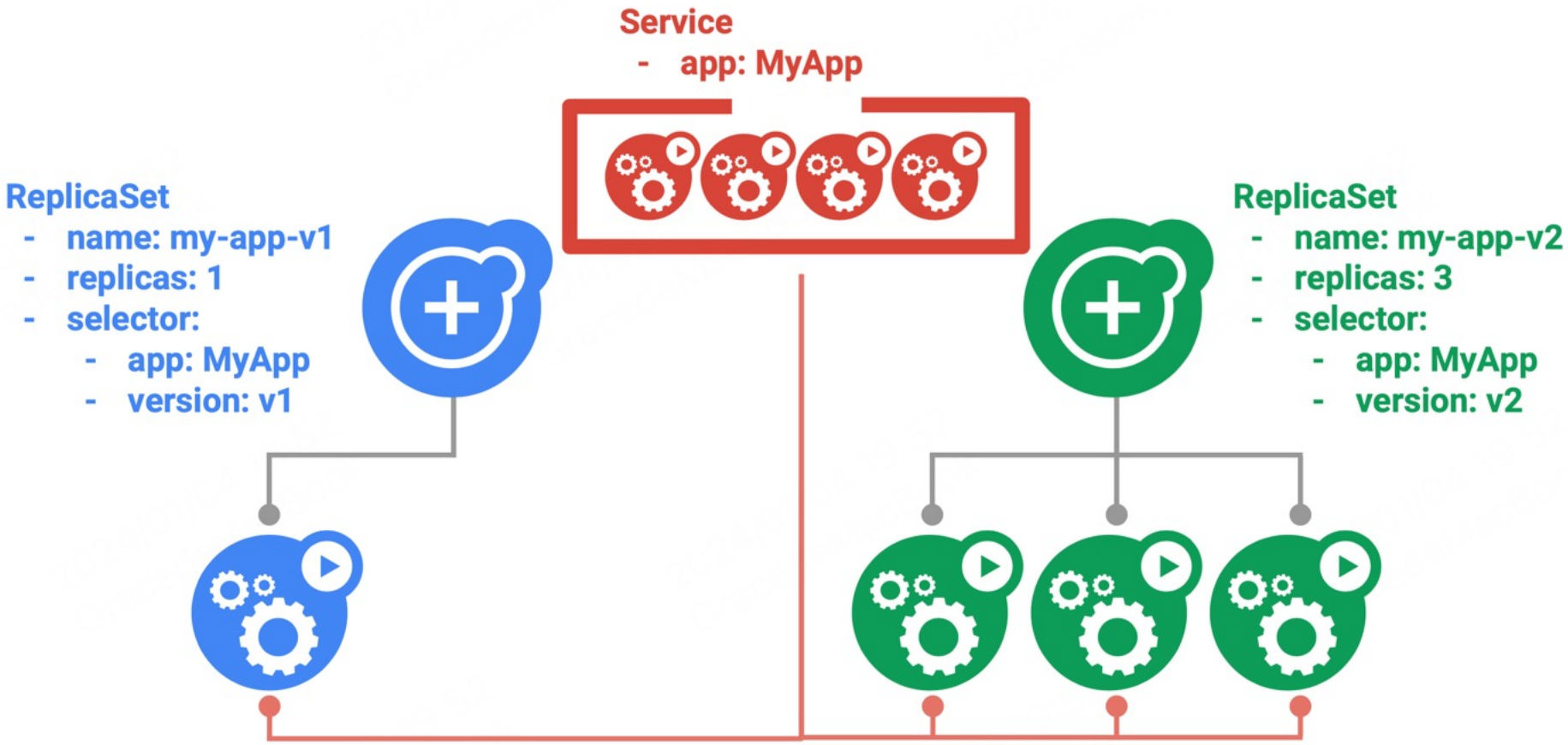
Rolling Update



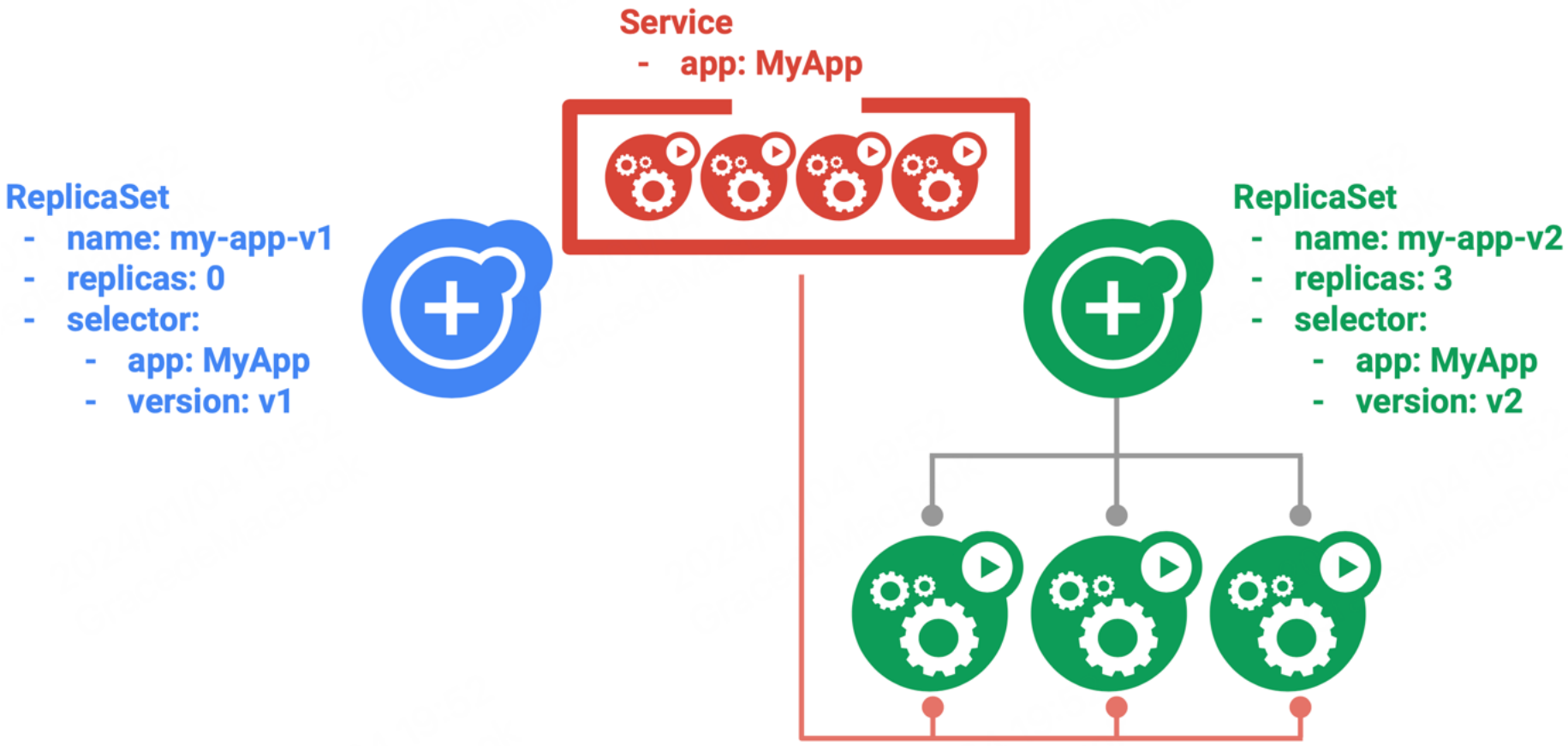
Rolling Update



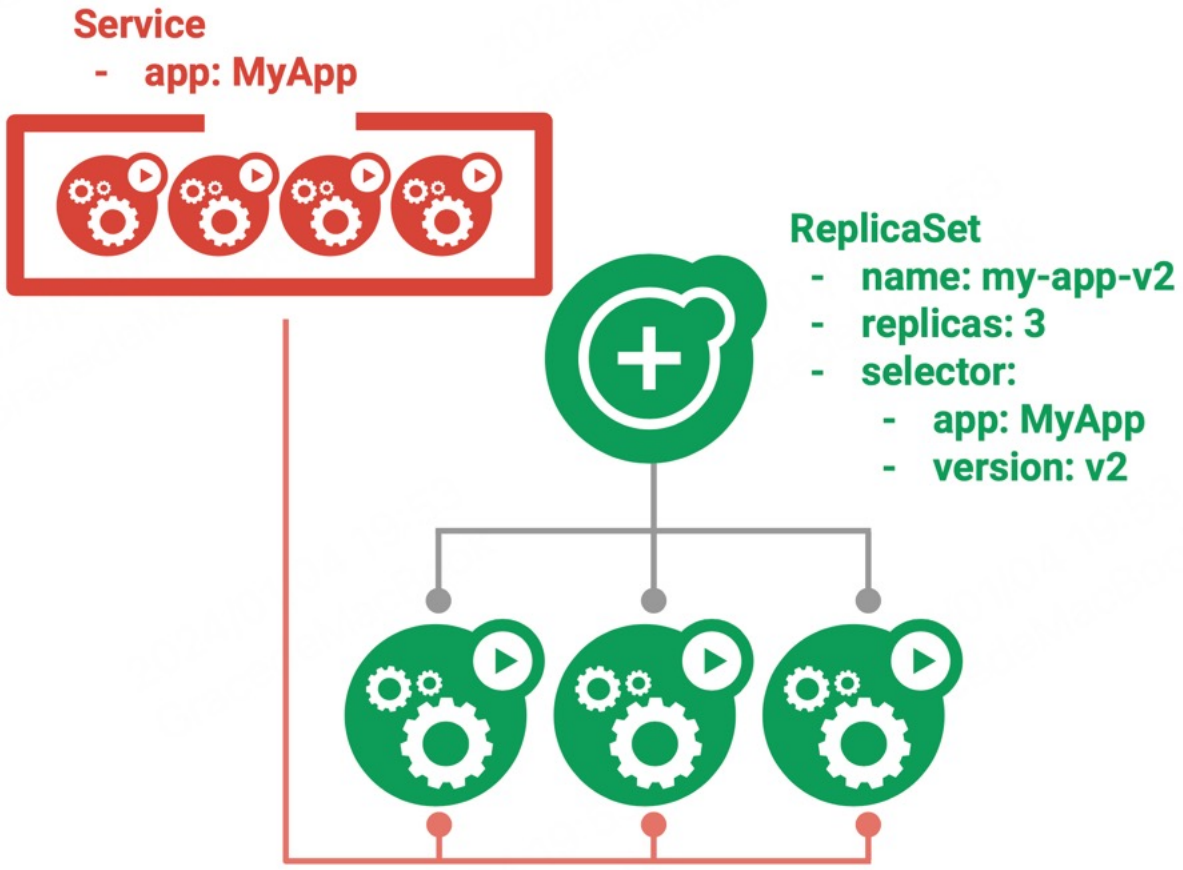
Rolling Update



Rolling Update



Rolling Update



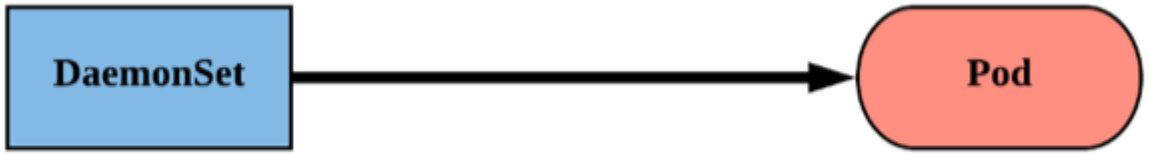
DaemonSet

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

确保符合特定条件的所有节点都将运行所提供 Pod 的实例。

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

非常适合日志转发或监控等集群范围的服务。



StatefulSet

Tailored to managing Pods that must persist or maintain state.

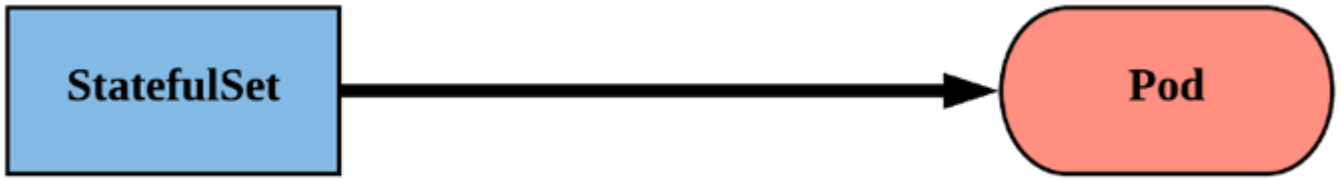
适用于管理必须持久或维持状态的 Pod。

Pod lifecycle will be ordered and follow consistent patterns.

Pod 生命周期将有序并遵循一致的模式。

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

按照 “ <statefulset 名称>-<序数索引> ” 的约定分配唯一的序数名称。



Job

Create one or more pods for carrying out batch processes

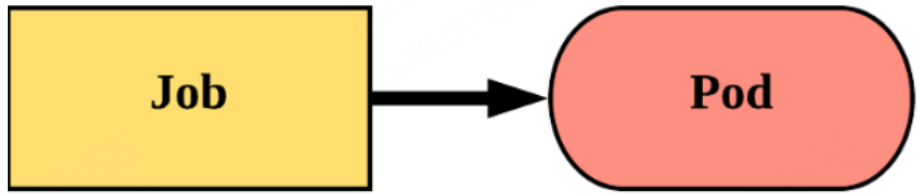
创建一个或多个 Pod 用于执行批处理

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

它会继续重试执行 Pod,直到指定数量的 Pod 成功终止

Can run multiple Pods in parallel

可以并行运行多个 Pod



CronJob

An extension of the Job Controller

Job Controller的扩展

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

提供按 cron 类计划执行作业的方法,即在指定时间运行

Metrics and Monitoring

HPA

Horizontal Pod AutoScalers

水平 Pod 自动度量器

Automatically scale number of pods as needed

根据需要自动扩展 Pod 数量

- Based on CPU utilization

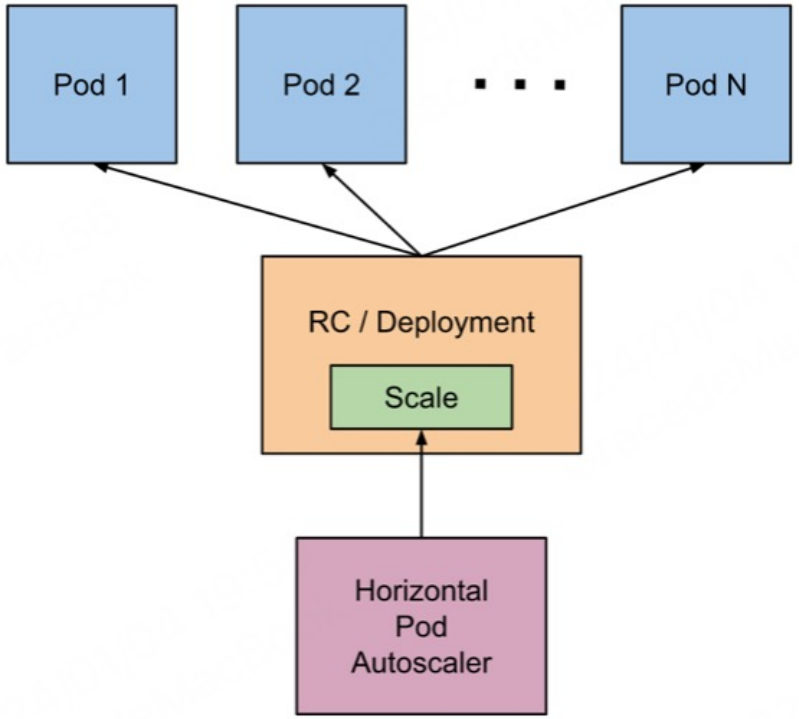
根据 CPU 利用率

- Custom metrics coming

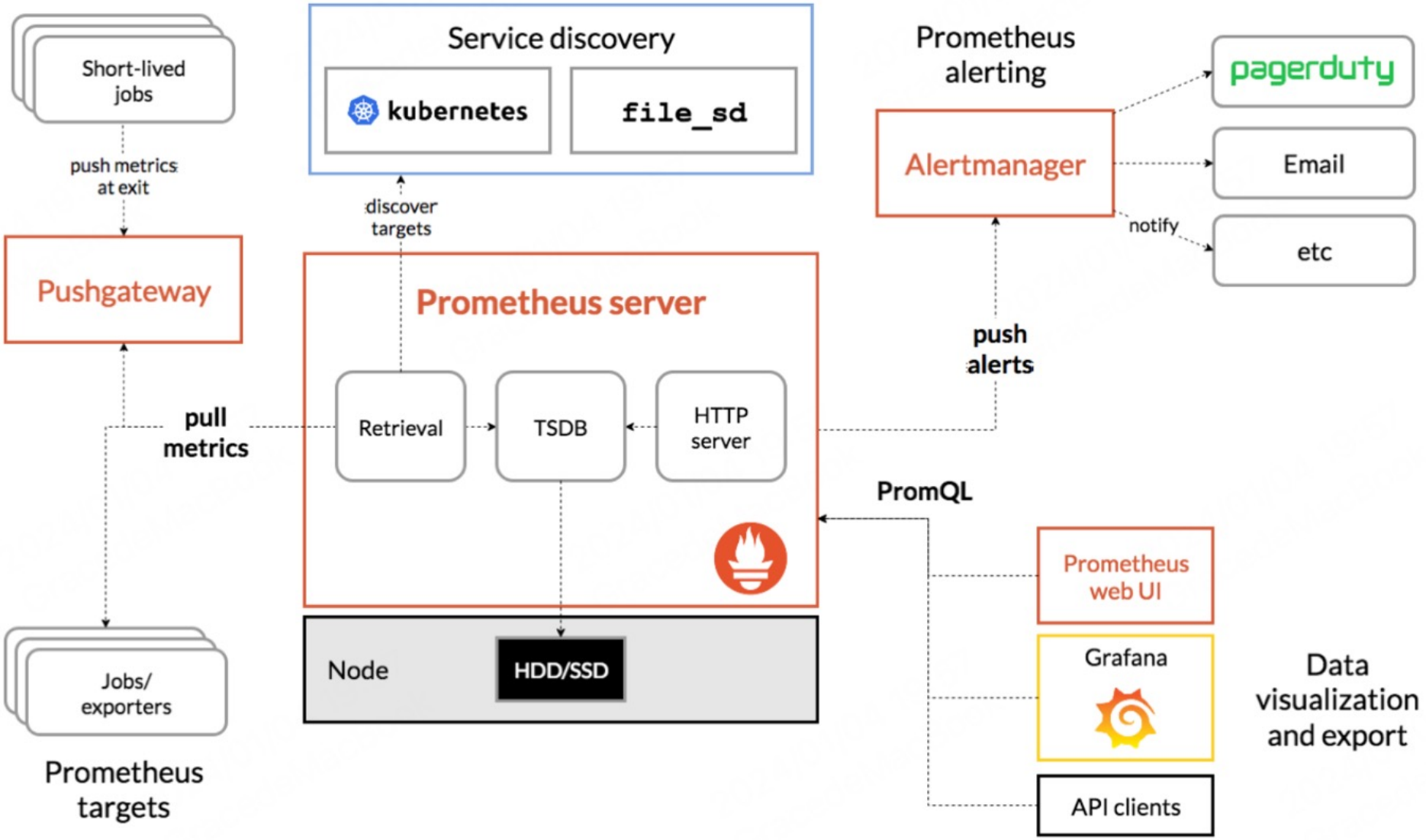
自定义指标即将推出

Operate within user-defined min/max bounds

在用户定义的范围内操作最小/最大界限



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

Kubernetes高度可配置、可扩展

- Adapt the Kubernetes cluster to the needs of the specific work environment

  使 Kubernetes 集群适应特定工作环境的需求

- Support new types and new kinds of hardware

  支持新类型和新种类的硬件

Extension

Kubectl plugins

API server

Custom Resources

Scheduler

Controllers

…

Credits

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