


云原生基础架构

——Container、Kubernetes

目录

CONTENTS

目录

CONTENTS

01 背景

云原生是什么？

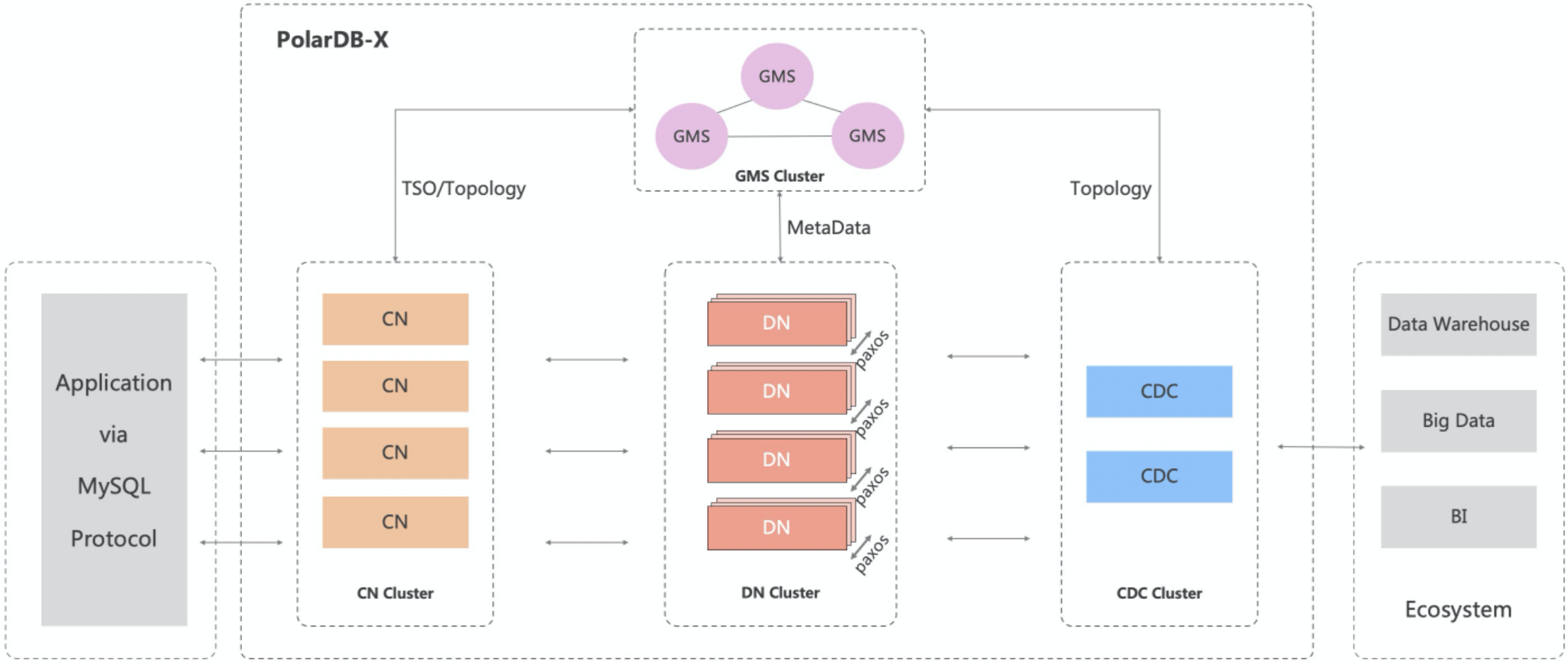
01 背景

是一种**构建**和**运行**应用程序的方法。

01 背景

各个组件作为 Container 在 Kubernetes 中运行。

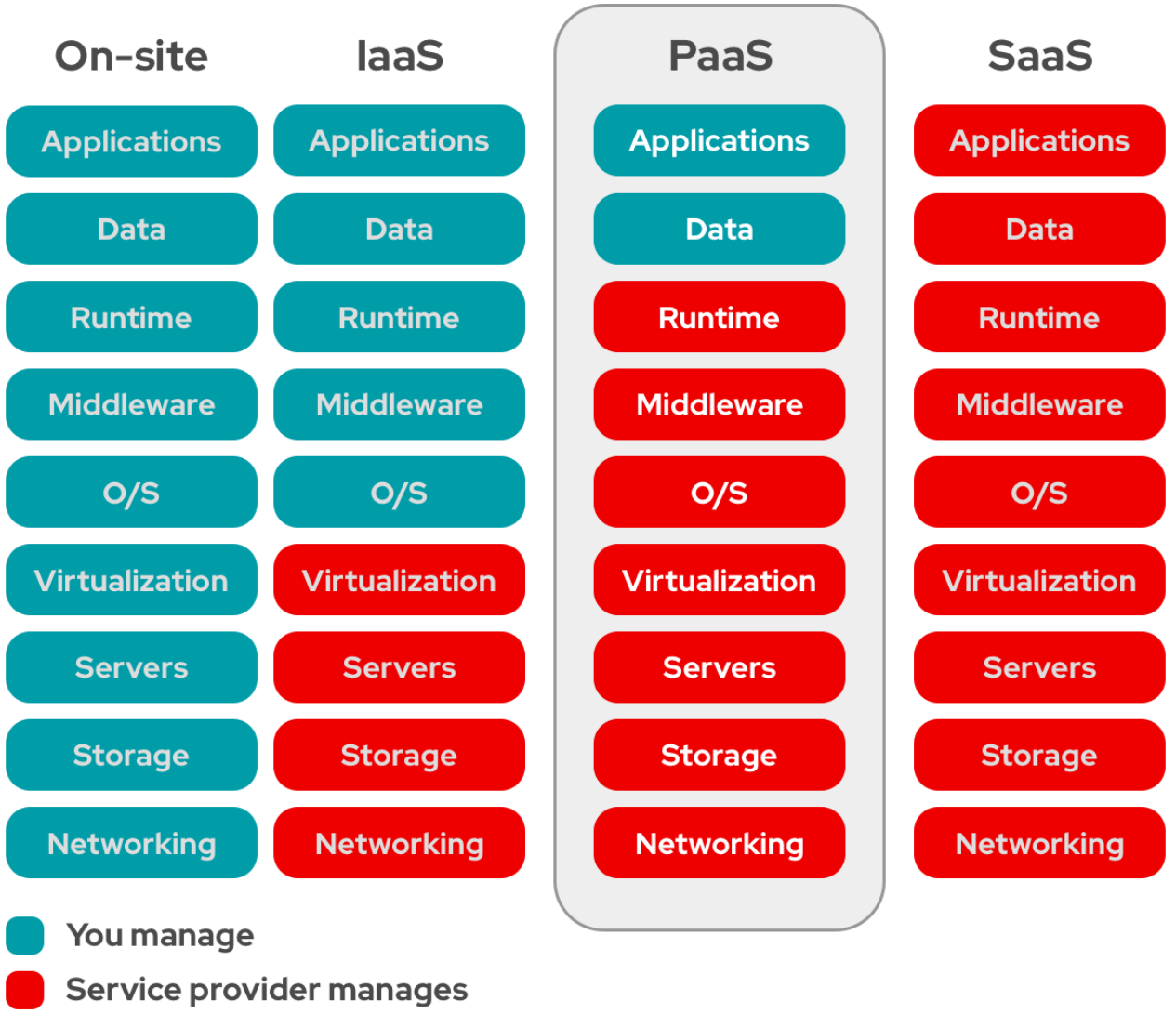
01 背景


PolarDB-X
GMS
GMS
GMS
GMS Cluster
TSO/Topology
MetaData
Topology
Application via MySQL Protocol
CN
CN
CN
CN
CN Cluster
DN
DN
DN
DN
paxos
paxos
paxos
DN Cluster
CDC
CDC
CDC Cluster
Data Warehouse
Big Data
BI
Ecosystem

01 背景

云原生是一种**构建**和**运行**应用程序的方法。

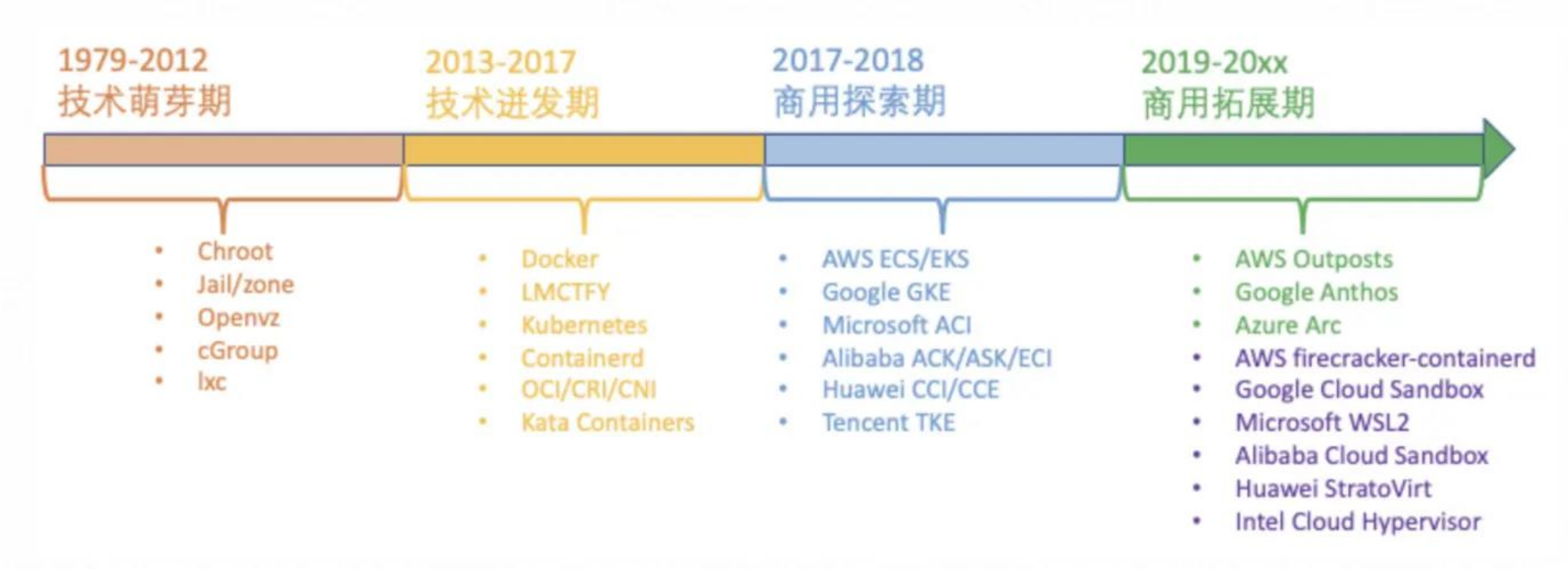
01 背景


On-site
IaaS
PaaS
SaaS
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
Service provider manages

01 背景

容器技术最核心的需求就是为应用创建一个“沙盒”隔离环境。

01 背景

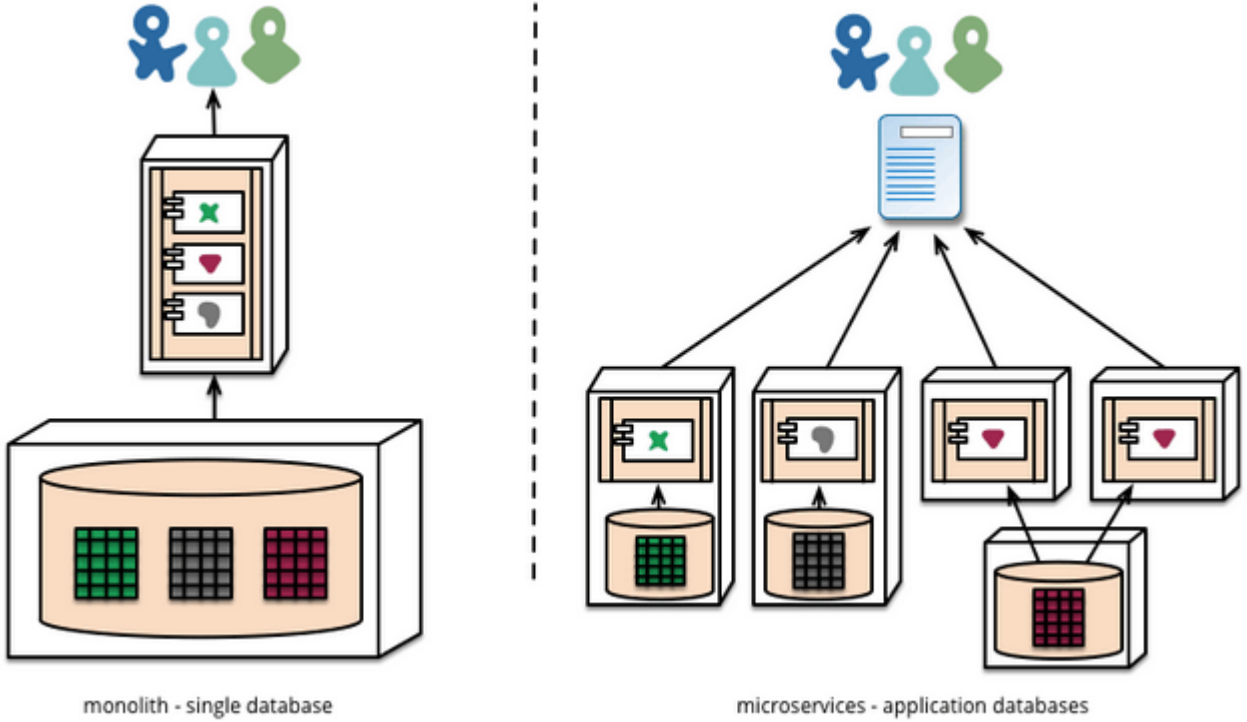


2008 年，Linux在其内核中包含容器功能时，容器技术变得**广泛可用**；并随着2013年 Docker 开源容器项目的到来而被**广泛使用**。

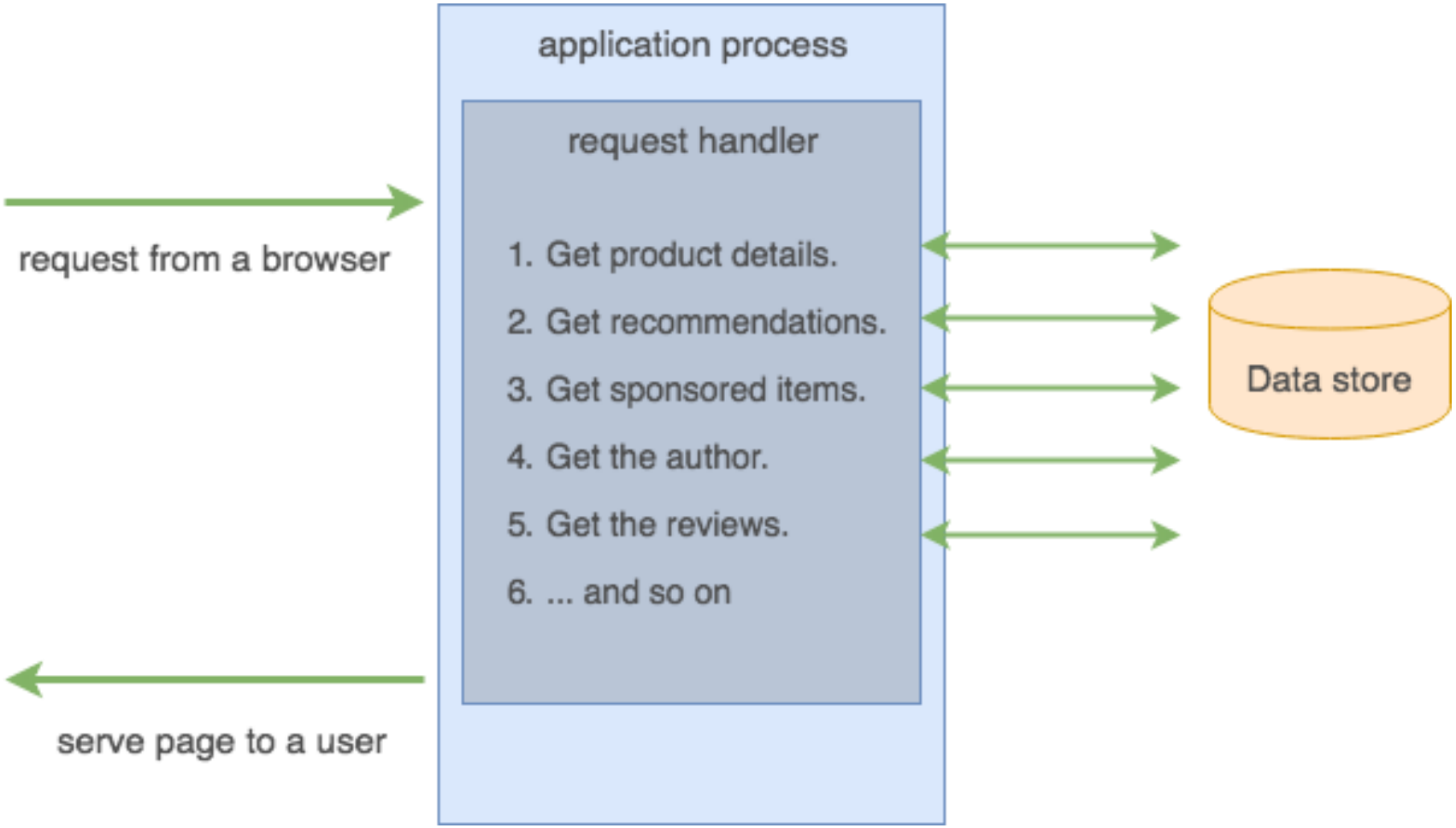
01 背景

微服务架构

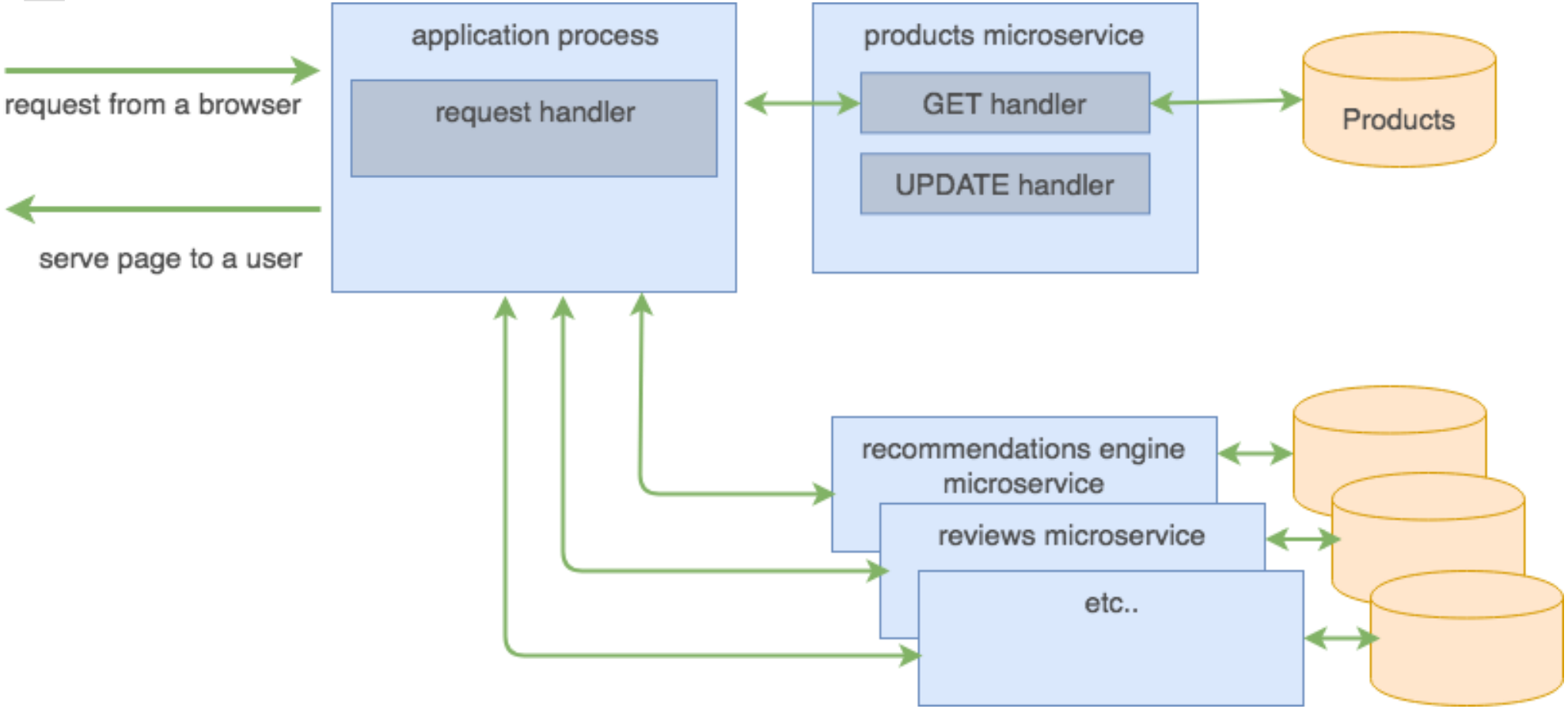
微服务架构是将**单一应用程序**开发为**一组小型服务**的方法。这些服务围绕业务能力构建，可用不同的语言开发，也可使用不同的存储技术。



01 背景


application process
request handler
request from a browser
1. Get product details.
2. Get recommendations.
3. Get sponsored items.
4. Get the author.
5. Get the reviews.
6. ... and so on
Data store
serve page to a user

01 背景


request from a browser
serve page to a user
application process
request handler
products microservice
GET handler
UPDATE handler
Products
recommendations engine microservice
reviews microservice
etc..

01 背景

Kubernetes 是用于自动部署，扩展和管理容器化应用程序的开源系统。

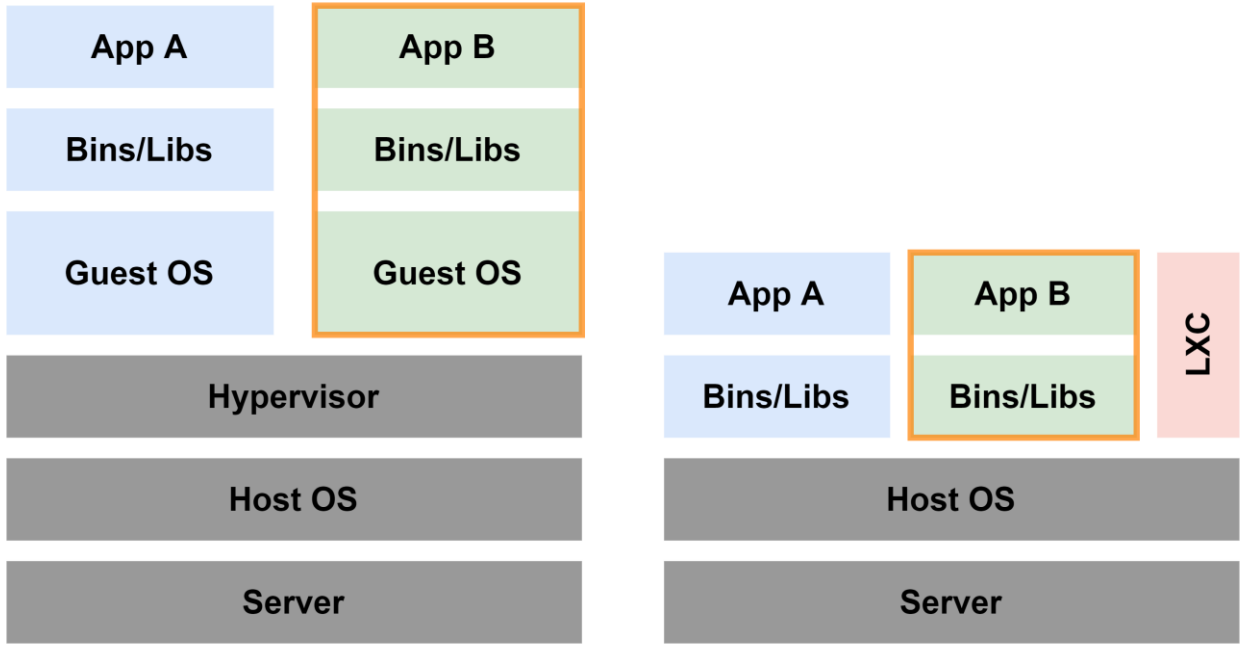
它将组成应用程序的容器组合成逻辑单元，以便于管理和服务发现。Kubernetes 源自Google 15 年生产环境的运维经验，同时凝聚了社区的最佳创意和实践。

目录

CONTENTS

02 容器——LXC

为应用创建一个“沙盒”隔离环境。



虚拟机　　容器

02 容器——LXC

对于应用来说，它的静态表现就是存放在磁盘上的程序文件；

而应用运行起来，就变成了一系列进程。

02 容器——LXC

约束和修改进程的动态表现，从而为其创造出一个“边界”。

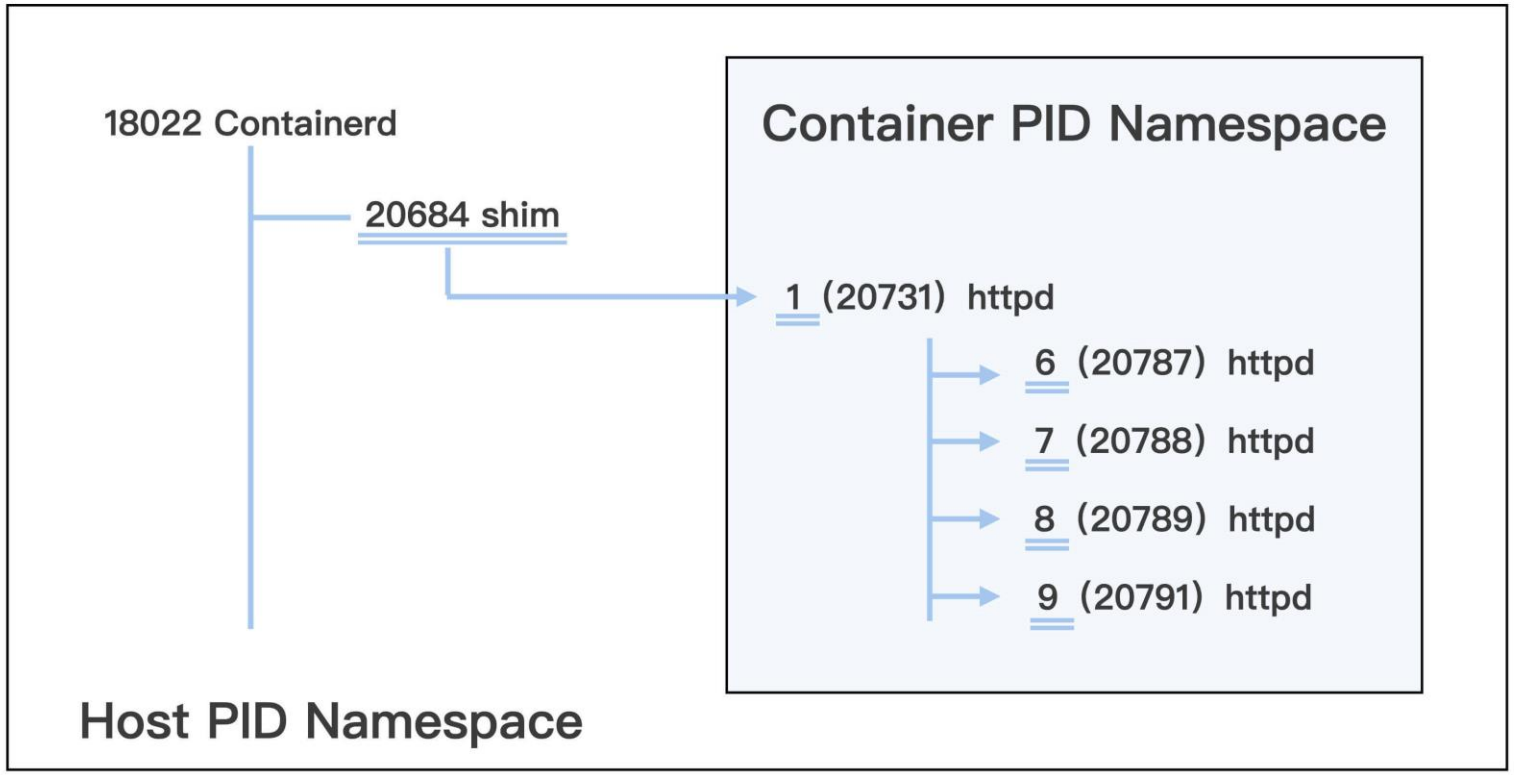
Namespace

Linux内核提供了如下6 种 Namespace 隔离的系统调用：

Namespace	系统调用参数	隔离内容
UTS	CLONE_NEWUTS	主机名与域名
IPC	CLONE_NEWIPC	信号量、消息队列和共享内存
PID	CLONE_NEWPID	进程编号
Network	CLONE_NEWNET	网络设备、网络栈、端口等等
Mount	CLONE_NEWNS	挂载点（文件系统）
User	CLONE_NEWUSER	用户和用户组

所以，实际上是在创建容器进程时，指定了这个进程需要启用的一组 Namespace 参数。

02 容器——LXC


18022 Containerd
20684 shim
Container PID Namespace
1 (20731) httpd
6 (20787) httpd
7 (20788) httpd
8 (20789) httpd
9 (20791) httpd
Host PID Namespace

02 容器——LXC

通过**隔离**能够创建出容器，但是还需要对其使用资源进行**限制**。

02 容器——LXC

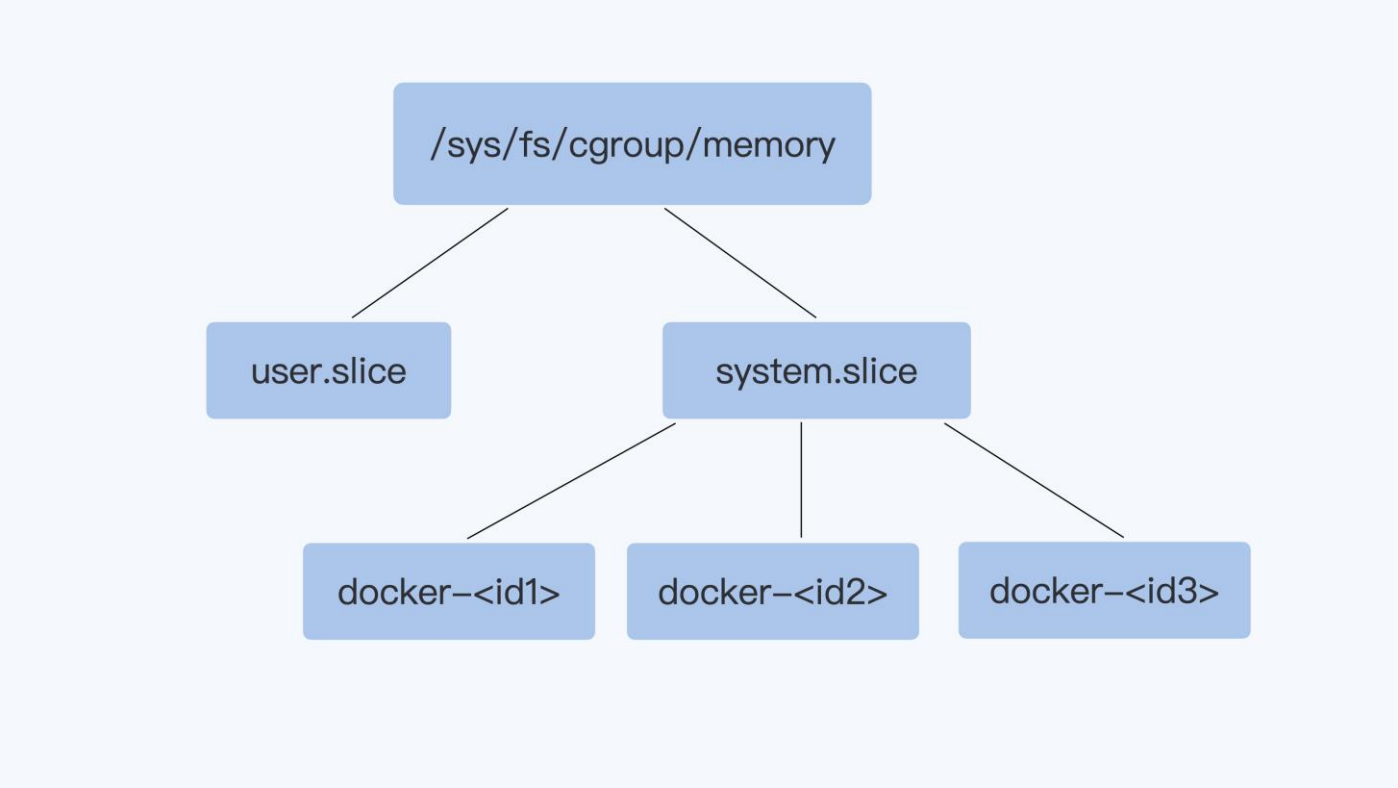
cGroups

cgroups 的全称是 control groups (控制组)。cgroups 为每种可以控制的资源定义了一个子系统。每一个子系统都需要与内核的其它模块配合来完成资源的控制。

它最主要的作用，就是限制一个进程组能够使用的资源上限，包括 CPU、内存、磁盘、网络带宽等等。

02 容器——LXC

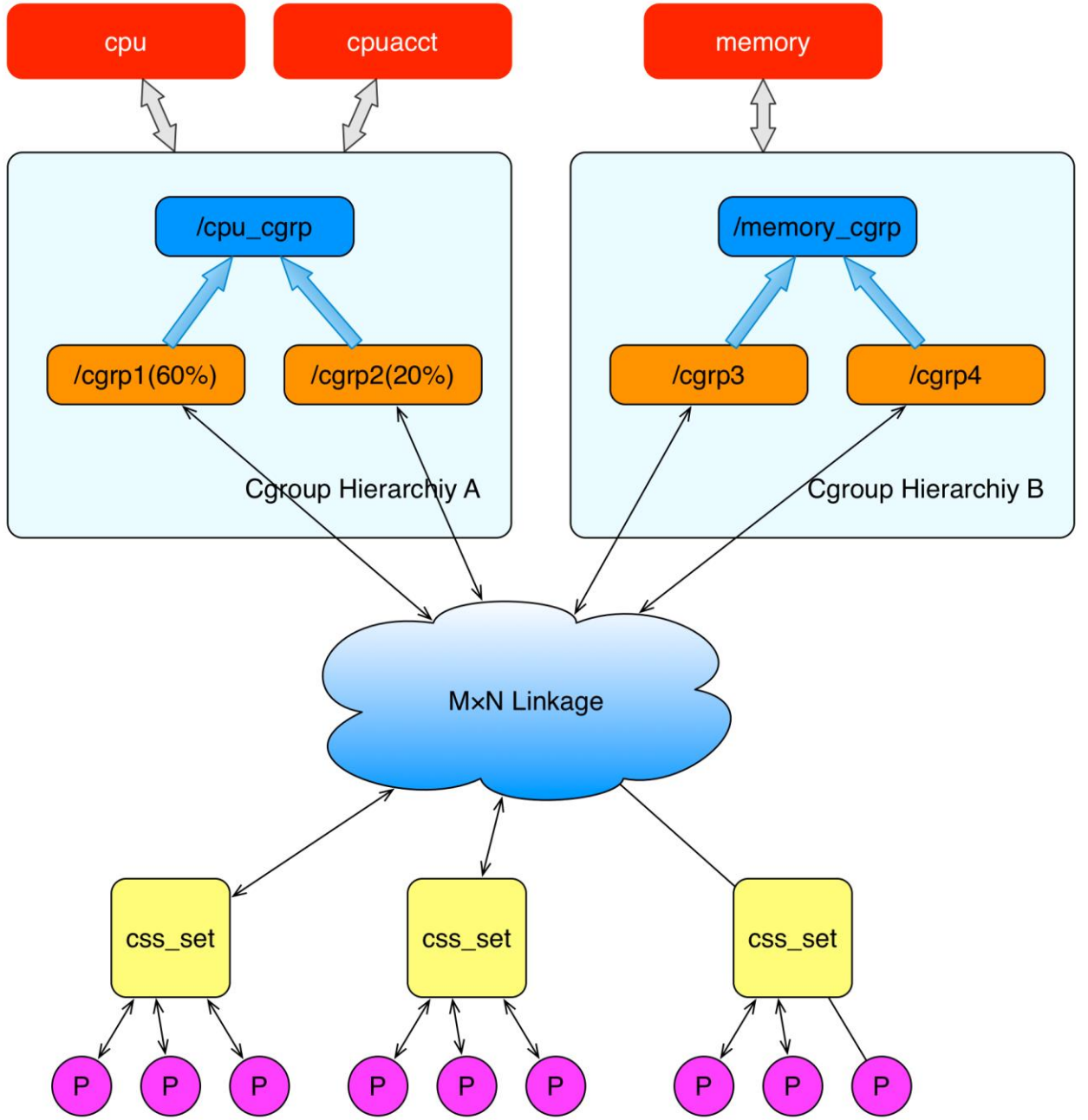
cGroups——memory子系统



02 容器——LXC

cGroups

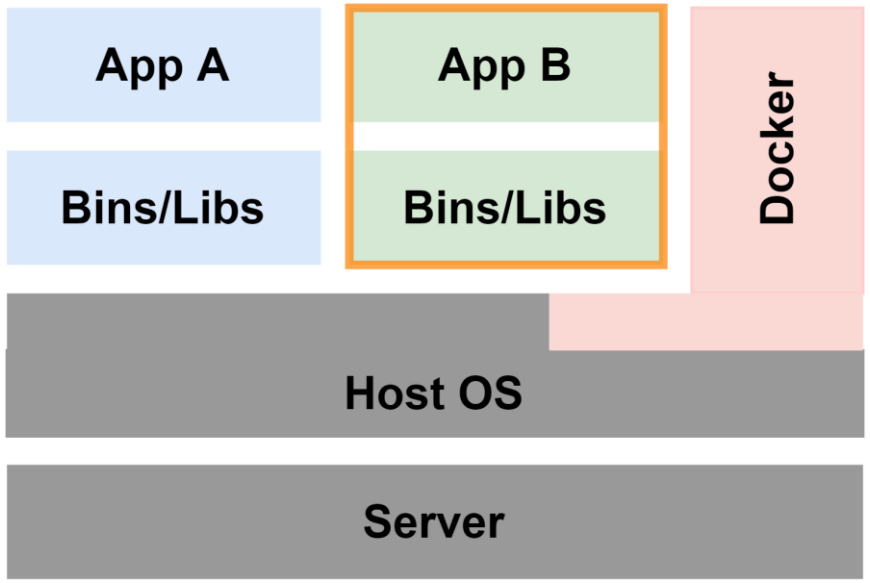
- cgroup 结构体组织成树。
- 进程和 cgroup 结构体是多对多的关系。



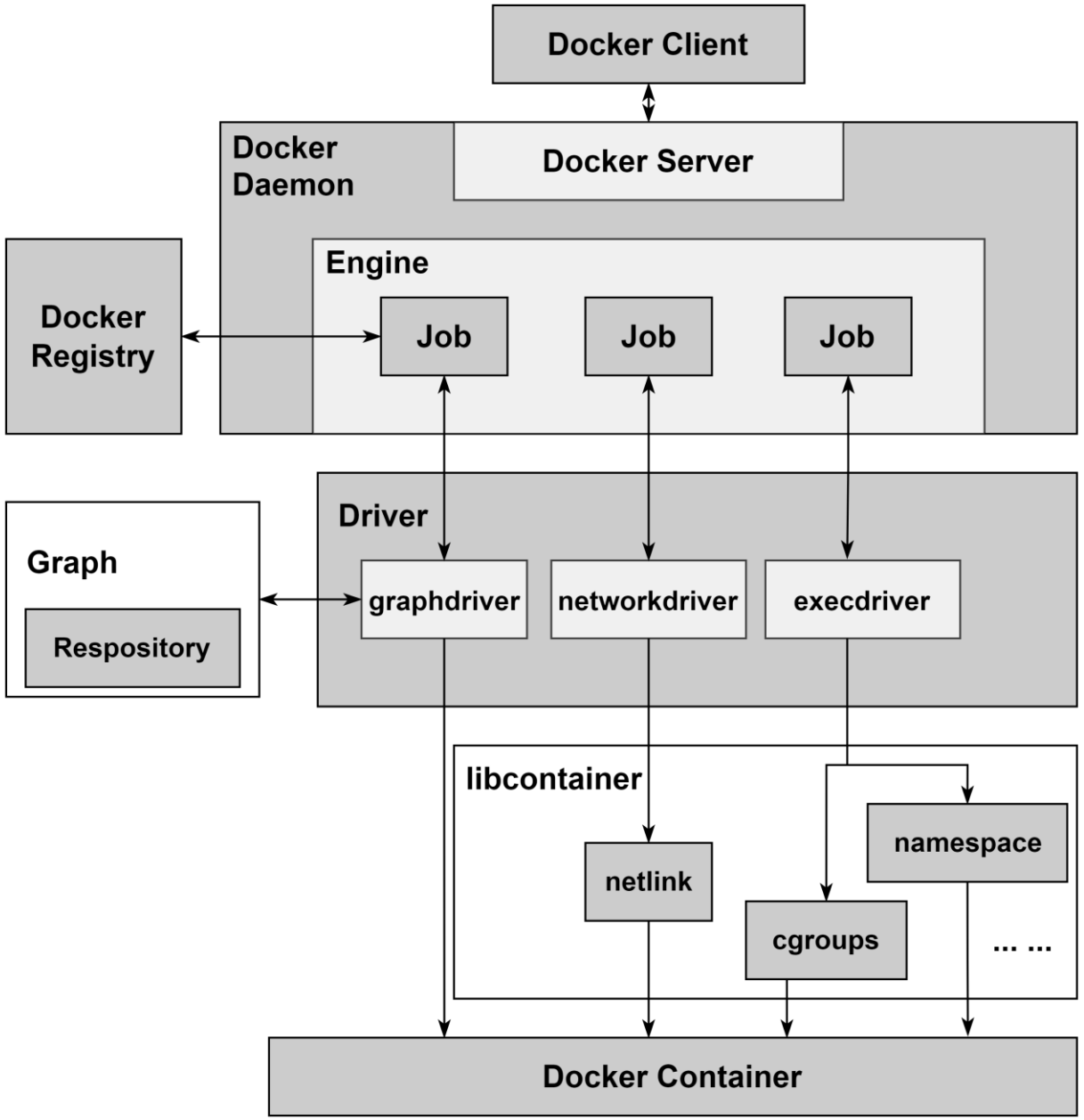
cGroups 就是一个子系统目录加上一组资源限制文件的组合

02 容器——Docker

Docker 可以看作是对 LXC 的改进。



02 容器——Docker


Docker Client
Docker Daemon
Docker Server
Engine
Job
Job
Job
Docker Registry
Driver
graphdriver
networkdriver
execdriver
Graph
Respository
libcontainer
netlink
namespace
cgroups
... ...
Docker Container

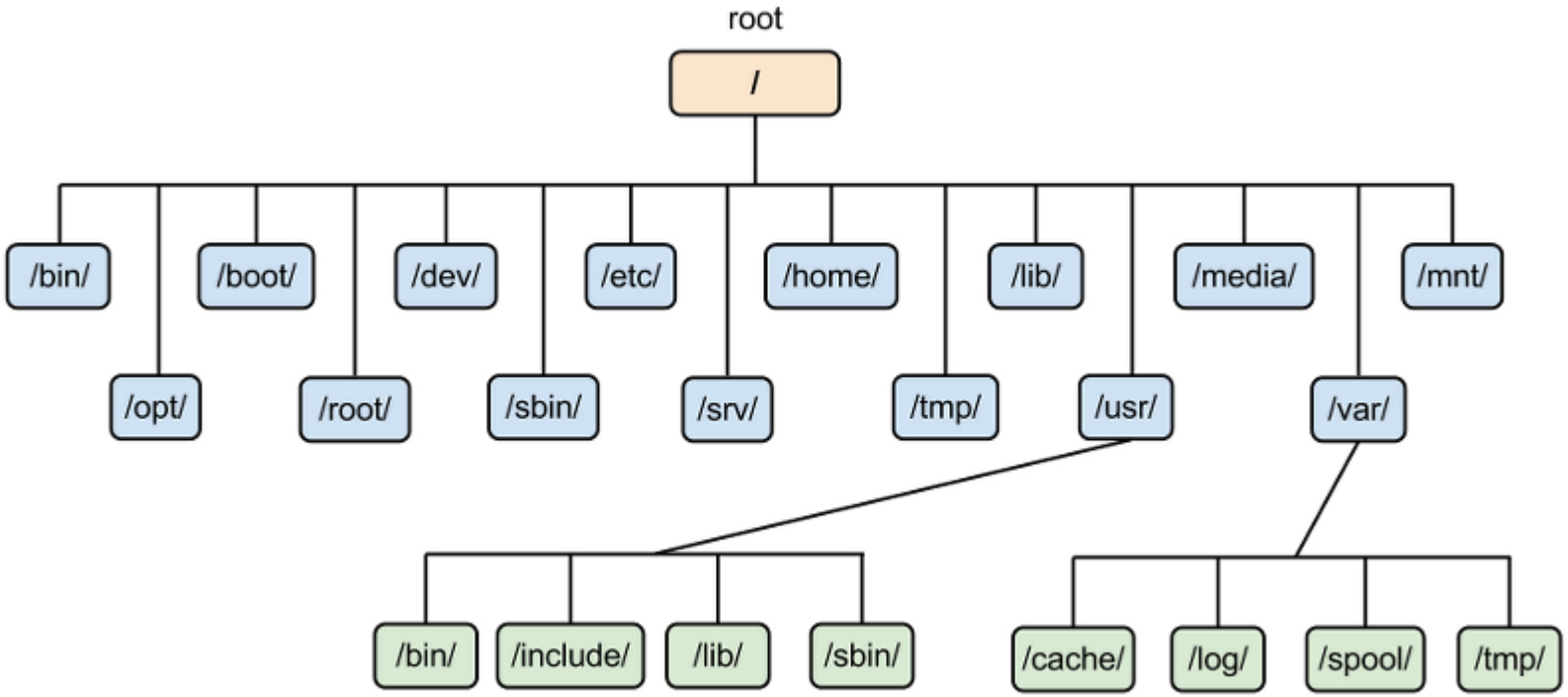
02 容器——LXC

但 Docker 能从一众容器产品中脱颖而出，很大程度上是得益于其对容器镜像的组织方式——Docker image

02 容器——Docker

容器镜像是挂载在容器根目录上、用来为容器进程提供隔离后执行环境的文件系统。

对于操作系统来说，这些文件也叫做 rootfs（根文件系统）。



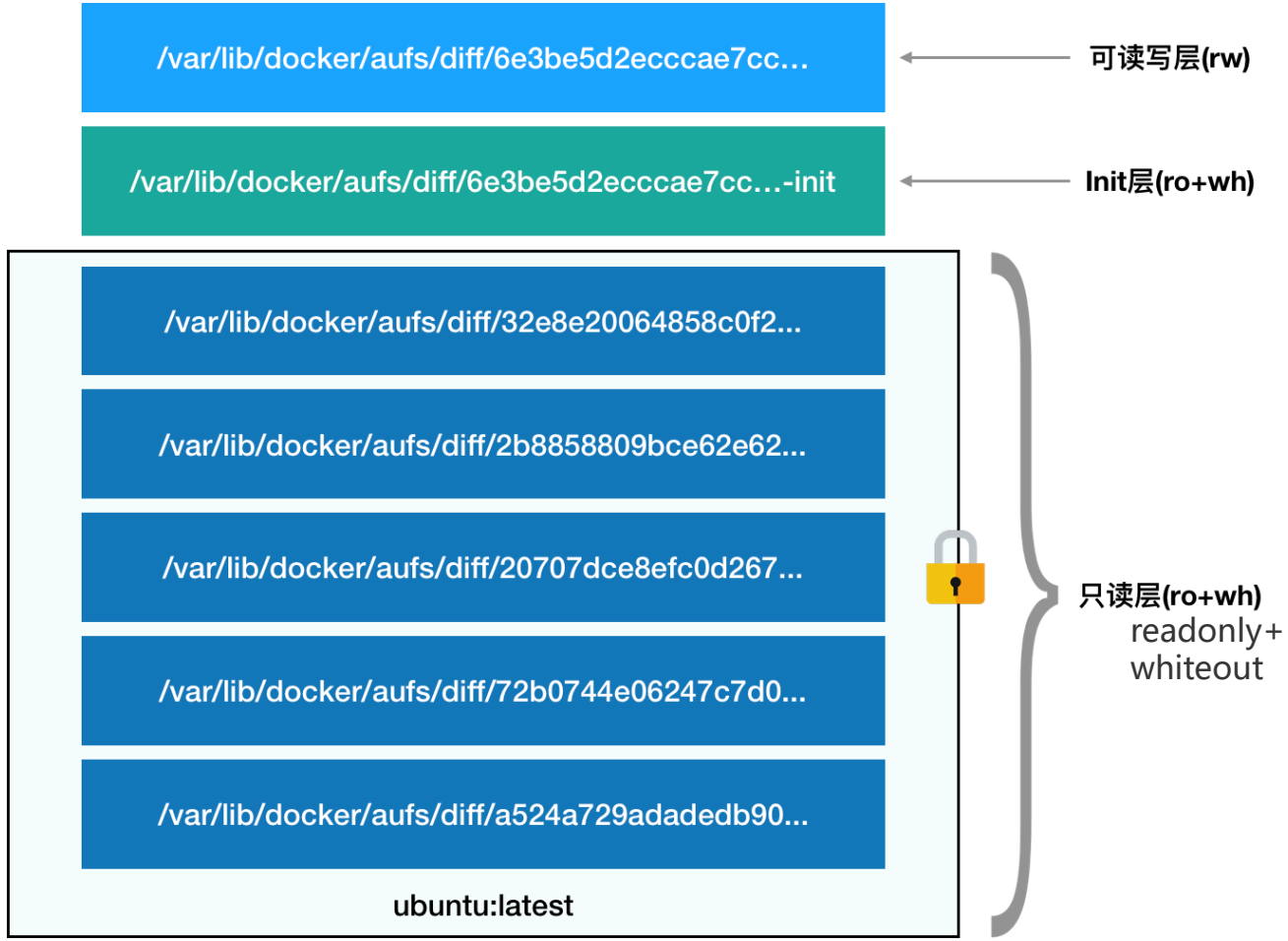
意味着容器一个非常重要的特性：**一致性。**

02 容器——Docker

但是，rootfs 会不会太大了？

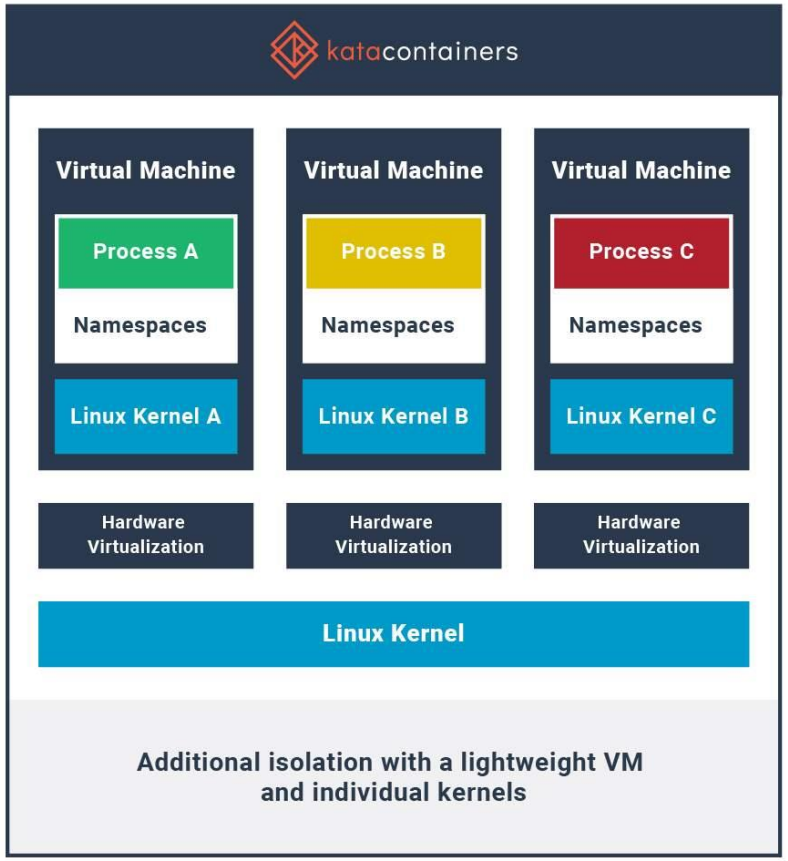
Docker 在镜像的设计中，引入了层的概念。

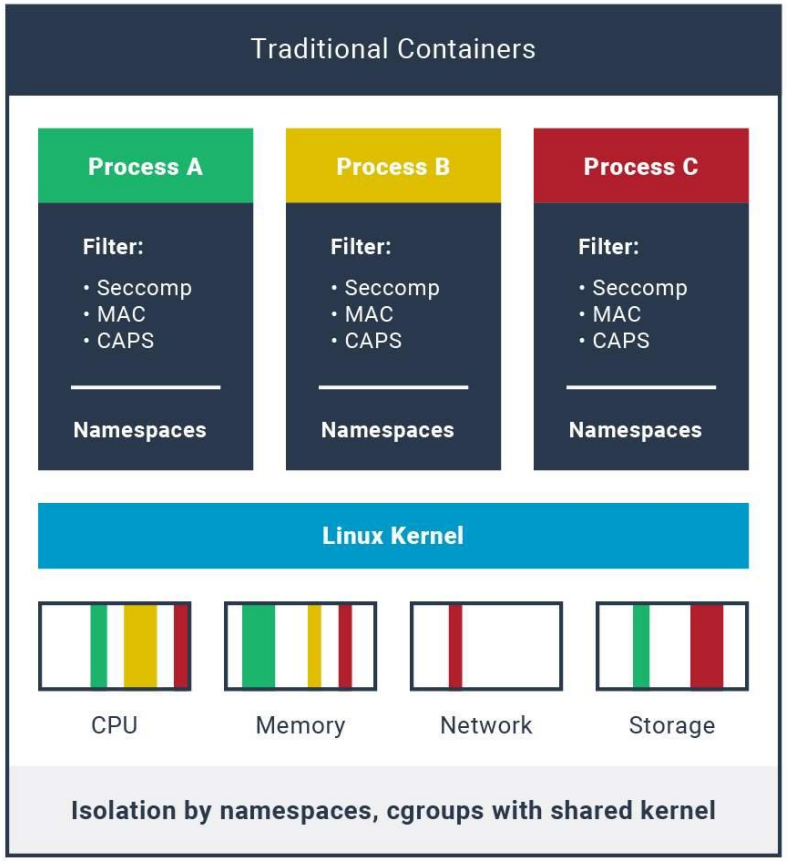
用户能够轻松地去分享和获取镜像，形成了庞大而有活力的容器镜像生态。



02 容器

容器技术仍在不断发展.....





目录

CONTENTS

03 容器编排

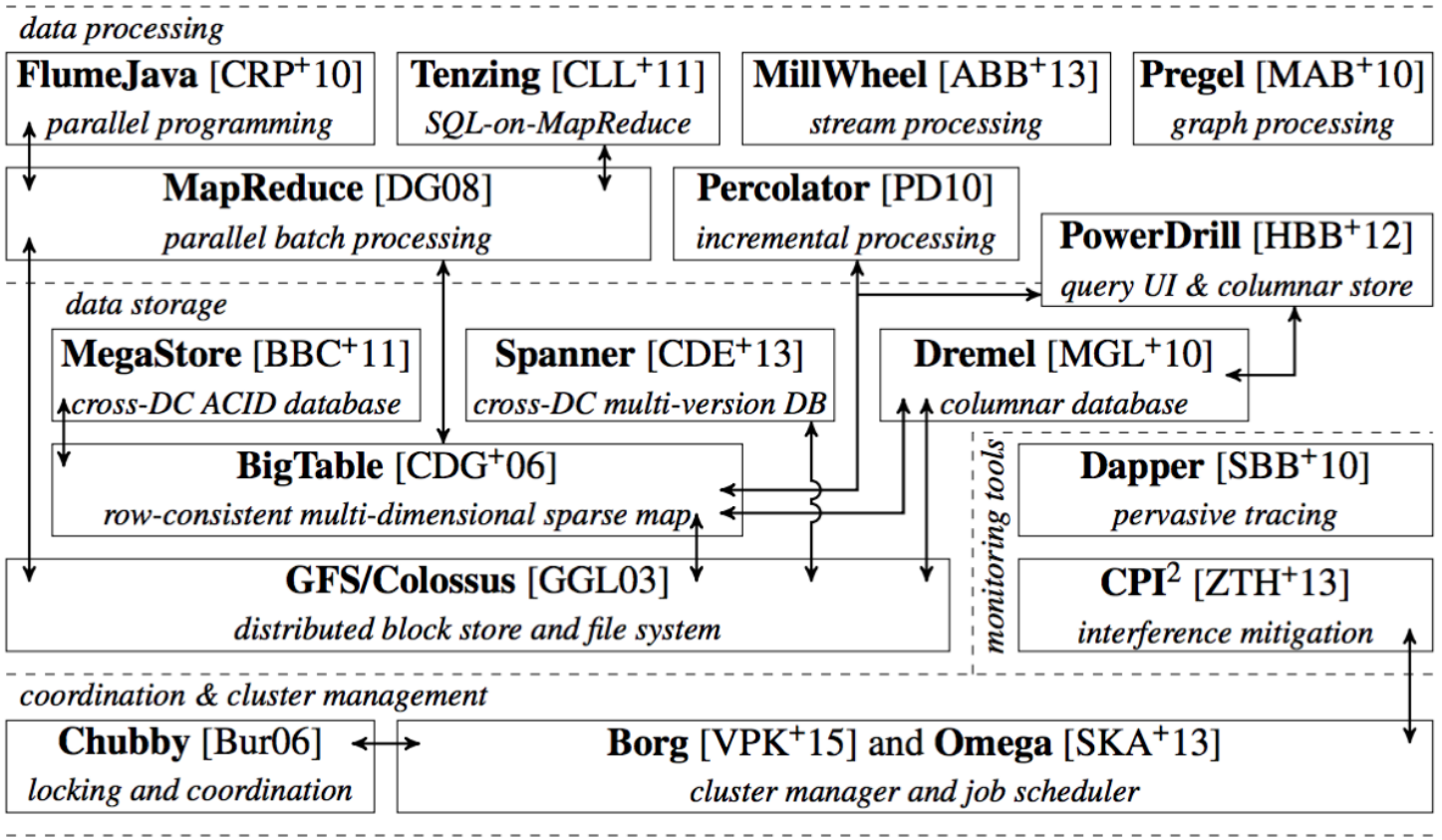
Kubernetes 是用于自动部署，扩展和管理容器化应用程序的开源系统。

它将组成应用程序的容器组合成逻辑单元，以便于管理和服务发现。Kubernetes 源自Google 15 年生产环境的运维经验，同时凝聚了社区的最佳创意和实践。

Borg ⟶ Kubernetes

03 容器编排——Borg

Google 一直在生成环境中运行容器化工作负载。



Google 基础设施栈

而诞生于 2003 年的 Borg 就是负责这些容器的调度和管理的容器编排系统。

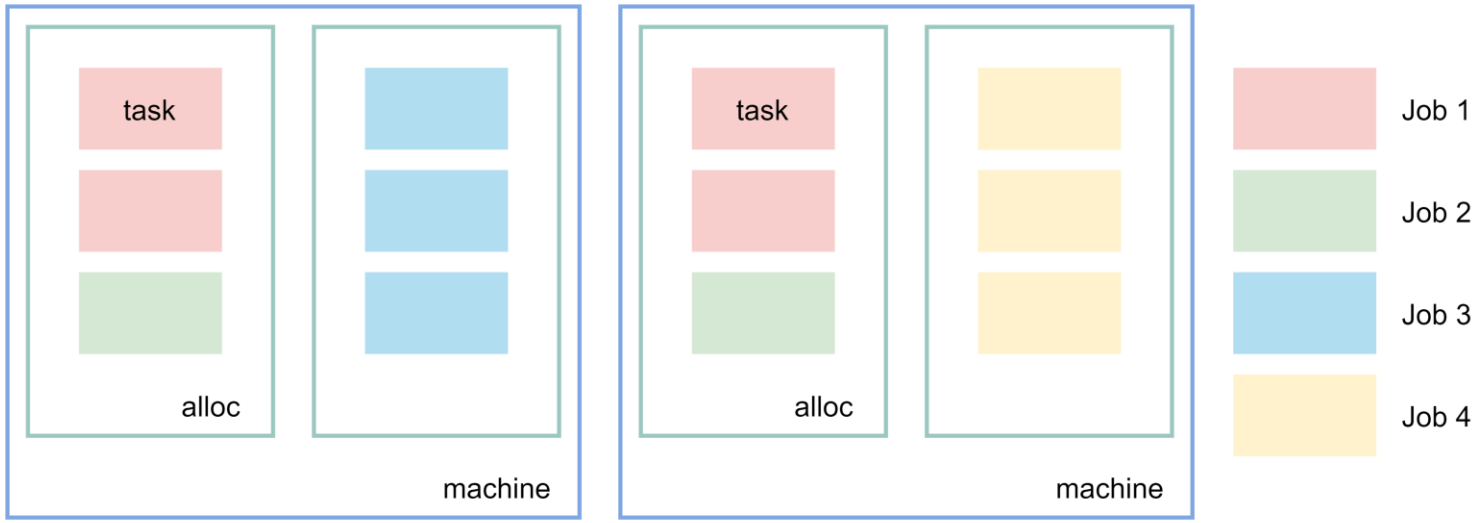
03 容器编排——Borg

Borg 要实现的最核心功能是将用户提交的应用在集群上运行起来。

容器编排——Borg

The User Perspective

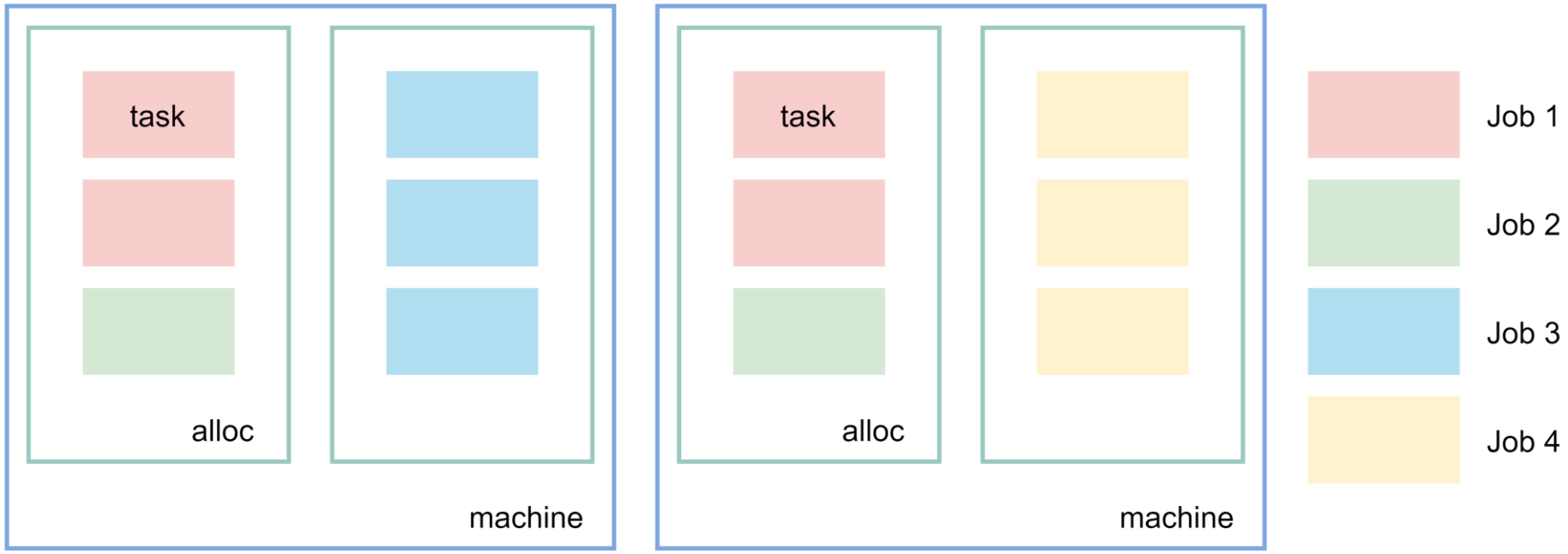
- task
  task 表示一个容器及其附加的一些属性
- job
  job 用来从用户角度将 tasks 分组。
- alloc
  alloc 表示物理机上一组 tasks 及其拥有的资源。



- 一个物理机集群命名为 Borg cell。

03 容器编排——Borg

The User Perspective

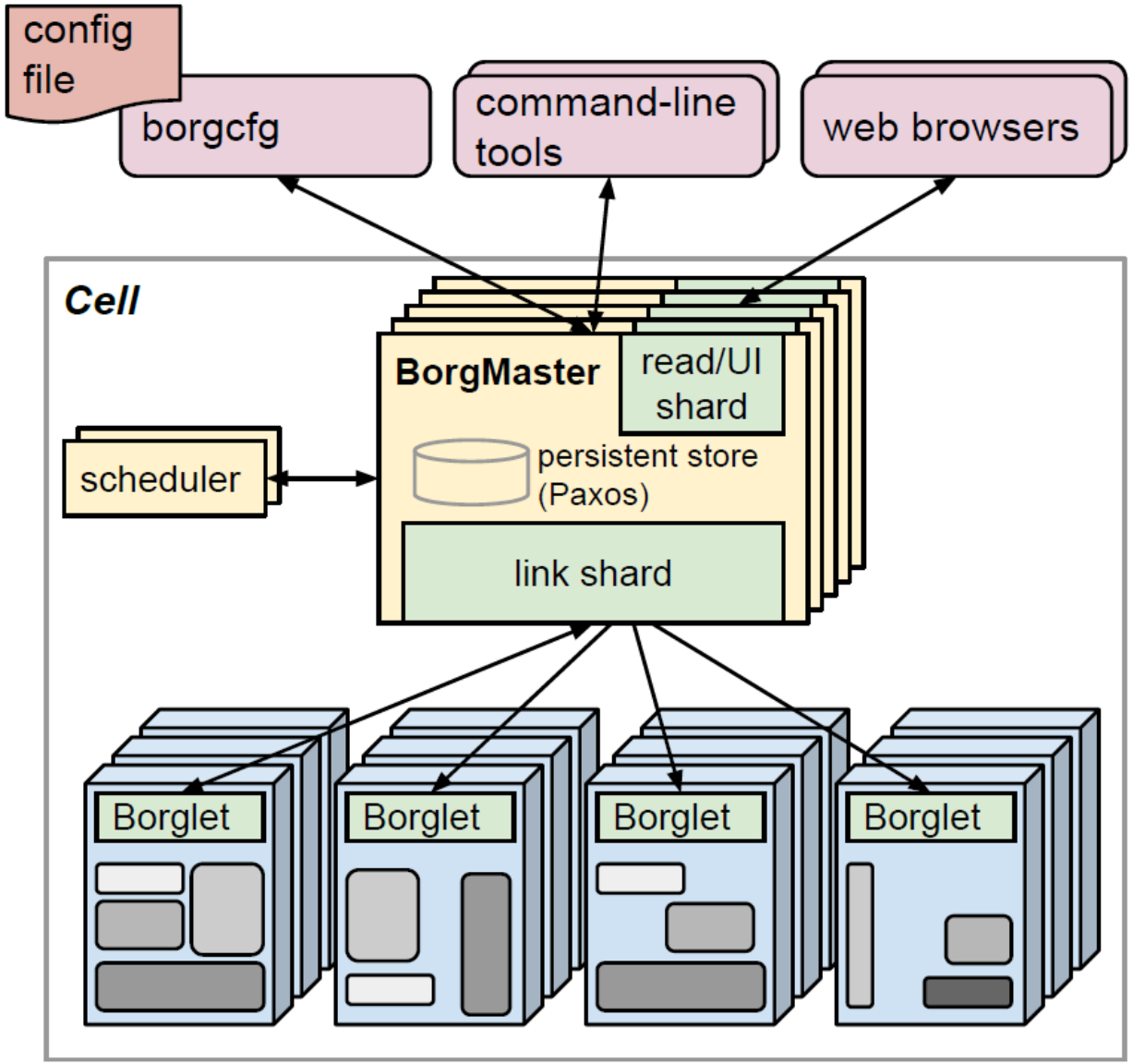


- 用户以 job 的形式将他们的工作提交给 Borg。
- 用户为每个 job 描述其中的 task。
- Borg 为了将 task 运行，为其分配 alloc，在其中运行 task。

03 容器编排——Borg

Borg Architecture

一个 Borg cell 由一组物理机，一个逻辑中心控制器（Borgmater），和运行在每个物理机上的代理进程组成（Borglet）。

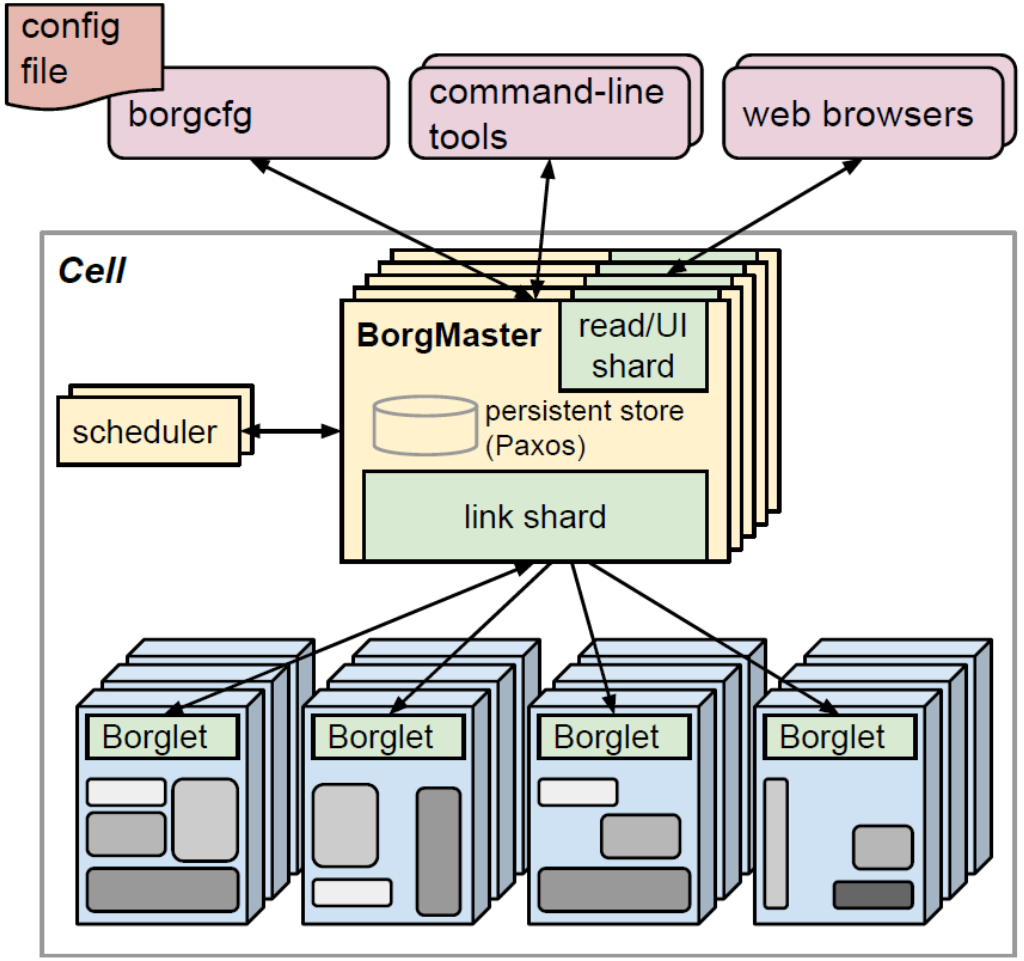


03 容器编排——Borg

BorgMaster

- Borgmaster 由两个进程构成：一个主进程和一个独立的调度器。

- Borgmaster 逻辑上是单个进程，但实际上有 5 个副本。cell 的状态会基于 Paxos 保存在这些副本的本地磁盘上。其中会有一个被选举出来的 master，同时作为 **Paxos 的 leader 和 cell 状态的修改者**，只有它能**发起**所有改变 cell 状态的操作。



03 容器编排——Borg

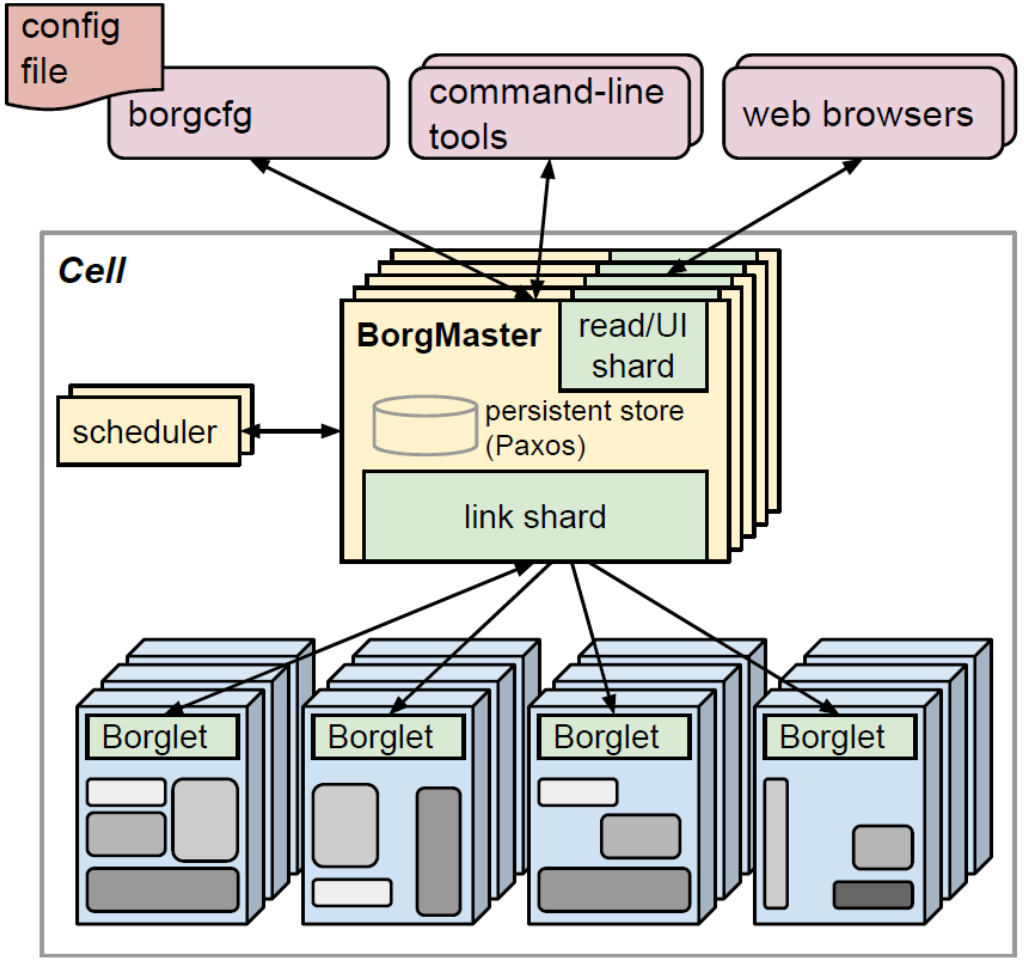
Scheduling

- Scheduling 是一个单独的进程。其调度的流程如下：
  - 当提交 job 时，BorgMaster 会将其记录在 Paxos 存储中，并将 job 的 tasks 添加到挂起队列中。
  - 调度器会**异步扫描**，从挂起队列取出 tasks，通过调度算法选出物理机。
- 调度算法分为两部分：
  1）**可行性检验**。寻找一组满足 task 约束并且拥有足够可用资源的物理机。
  2）**评分**。对这组可用物理机按**评分算法**评分，选出分数最高的一台机器。

容器编排——Borg

Borglet

- Borglet 是存在于每个物理机上的 Borg 代理。
- 用来启动或停止 task；重启失败的 task；调用本地资源；记录 debug 日志；向 Borgmaster 或其它监控系统报告物理机的状态。
- 其报告状态的机制是 Borgmaster 每隔几秒会轮询 Borglet 来收集状态。

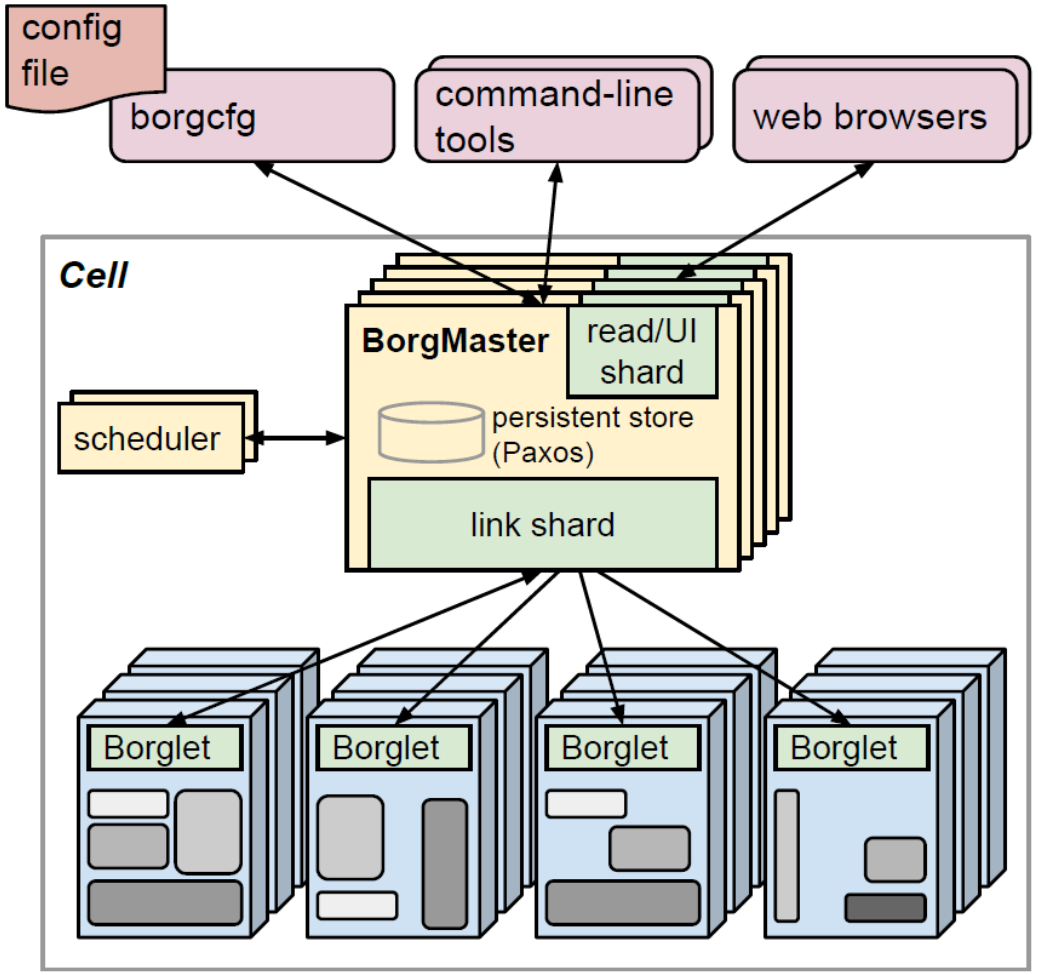


03 容器编排——Borg

为了支持大规模集群，Borg 做了以下优化：

Borglets 被分区，由一个BorgMaster follower 负责一个分区。

- 报告状态时。Borglets 将完整的状态报告给 BorgMaster follower，follower 会把这些信息聚合并压缩，只向 BorgMaster master 报告与之前的不同之处。
- 发起操作时。所有操作都由 BorgMaster master 发起，由 Paxos 同步日志之后，只由对应分区的 BorgMaster follower 向 Borglets 发出指令，然后 Borglets 在物理机上完成操作。



03 容器编排——Borg

资源管理

Borg 中用户的工作负载主要有两种。

- 第一种是长时间运行的服务，其一直运行，并且需要处理延迟敏感的请求。例如 Gmail、Google Docs、Web 搜索等面向用户的产品，和 Bigtable 等内部基础设施服务。

- 第二种是批处理作业，完成时间从几秒到几天不等，并且它们对短期性能波动不敏感。

合理调度这两种工作负载是高效利用资源的关键，在服务的低谷期，回收资源给批处理作业使用；并在服务的高峰期，抢占批处理作业的资源。

03 容器编排——Borg

资源管理

首先，用户需要为 task 指定一个资源上限（**limit**），也代表资源的请求量。

定义 task 的资源预留值（**reservation**），即实际分配的资源，该预留值是从 Borglet 收集的 task 对资源的使用率计算得出。

limit – reservation 就是回收的资源，这些资源将被用来运行批处理任务。

03 容器编排——Borg

最后，Google 总结了其在 Borg 上的一些经验，为 Kubernetes 的发展提供了指引：

- 将 job 作为 task 的唯一分组机制会有诸多限制
- 一台物理机上的 task 都使用主机 IP 地址会增加系统的复杂度
- 不应给用户暴露太多控制权。Borg 为"高级用户"提供了大量的特性，和很高的控制权，但是这导致"临时用户"的体验变得非常糟糕。需要再构建自动化工具和服务来控制。
- **allocs 非常有用。**其能够管理资源，并且能将紧密交互的 task 绑定在一起。
- **集群管理不仅是容器的管理。**还需要管理这些容器对外暴露的服务，管理负载均衡等，即还需要提供更多的管理器（控制器），管理更为抽象的对象。

03 容器编排——Kubernetes

K8s 和 Borg 是一脉相承，但又大不一样。

03 容器编排——Kubernetes

The User Perspective

Container

容器仍是运行的最小单位，但是容器不是最小调度单元。

Pod

Pod 是 Borg 中 alloc 概念的延申，**Pod 是 Kubernetes 的最小调度单元和最小控制单元。**

Pod 中的容器共享 IP，共享 Volume，一起调度。

Label

Borg 中将一组容器的集合抽象为 Job，而 Kubernetes 通过 Label 来支持比 Borg 更灵活的集合。

使用 **标签选择器** 来更灵活的查询和筛选对象

03 容器编排——Kubernetes

集群管理不仅是容器的管理，需要更多的控制器来管理更抽象的对象。

- 无状态应用 Deployment
- 有状态应用 StatefulSet
- 守护进程 DaemonSet
- 批处理任务 Job
- 周期性批处理任务 CronJob

03 容器编排——Kubernetes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

扩容

```
kubectl scale deployment nginx-deployment --replicas 10
```

自动扩容

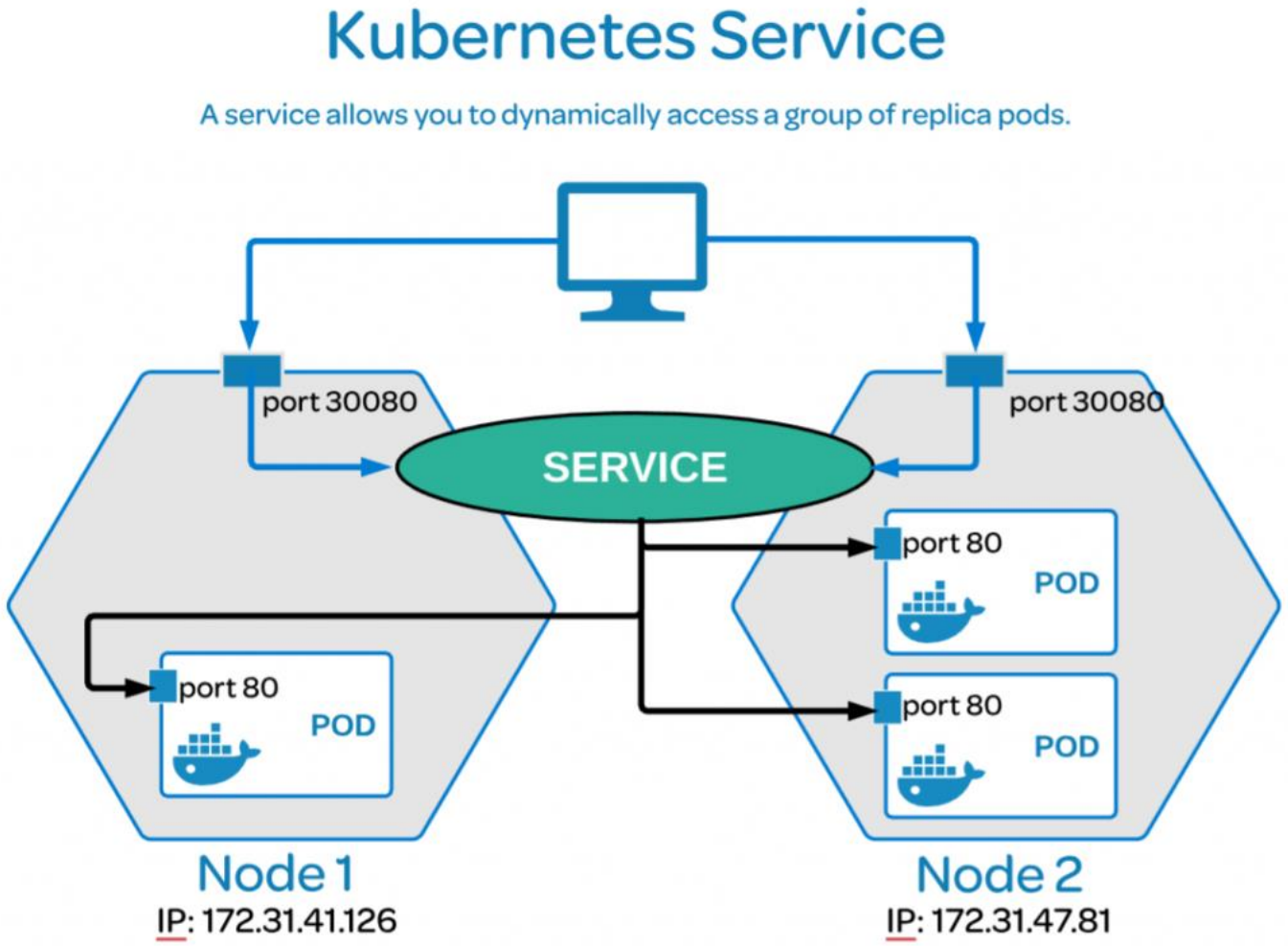
```
kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80
```

升级

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

03 容器编排——Kubernetes

Service 对象。

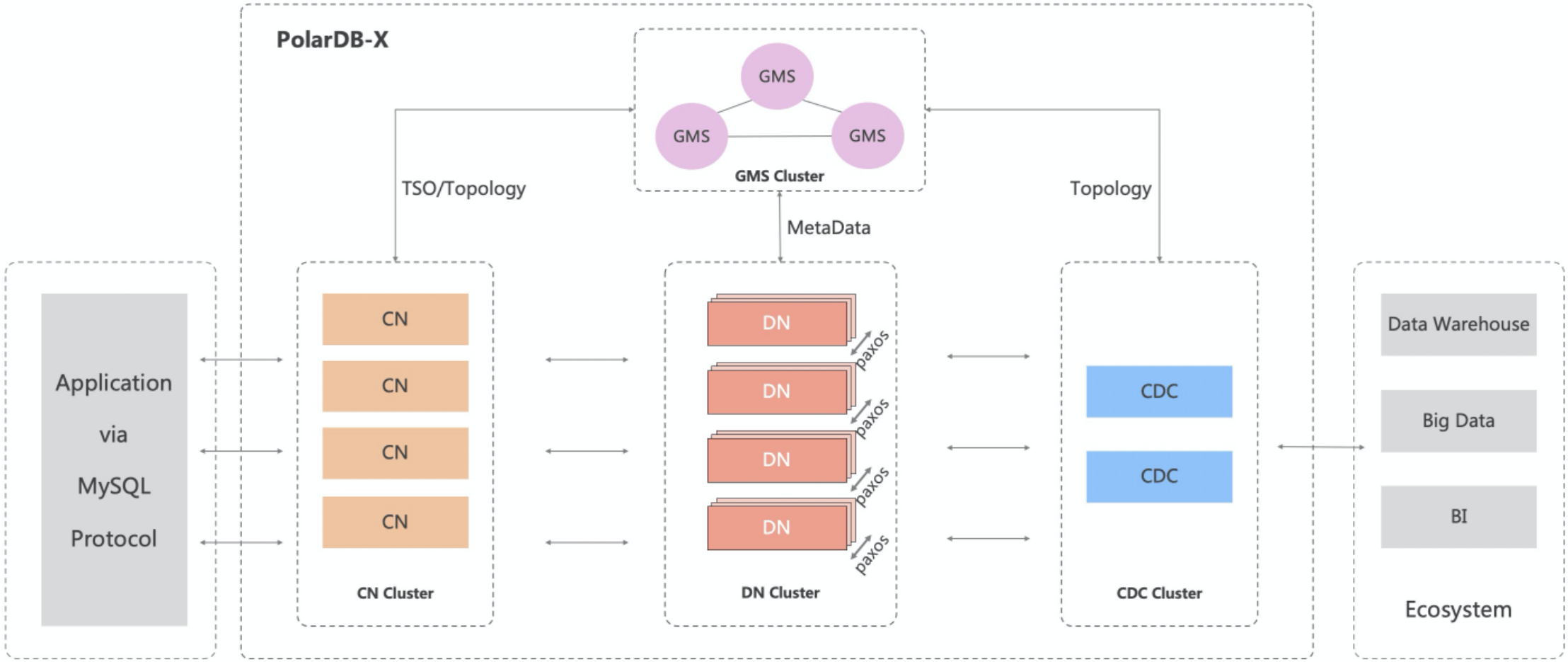


03 容器编排——Kubernetes

例如，假定有一组 Pod，它们对外暴露了 9376 端口，同时还被打上了 app=MyApp 的 lable。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

03 容器编排——Kubernetes



03 容器编排——Kubernetes

Operator =

自定义资源

+

自定义控制器

TiDB Operator

provided by PingCAP

TiDB Operator manages TiDB clusters on Kubernetes and automates tasks related to

Kubeflow

provided by Kubeflow

Kubeflow Operator for deployment and management of Kubeflow

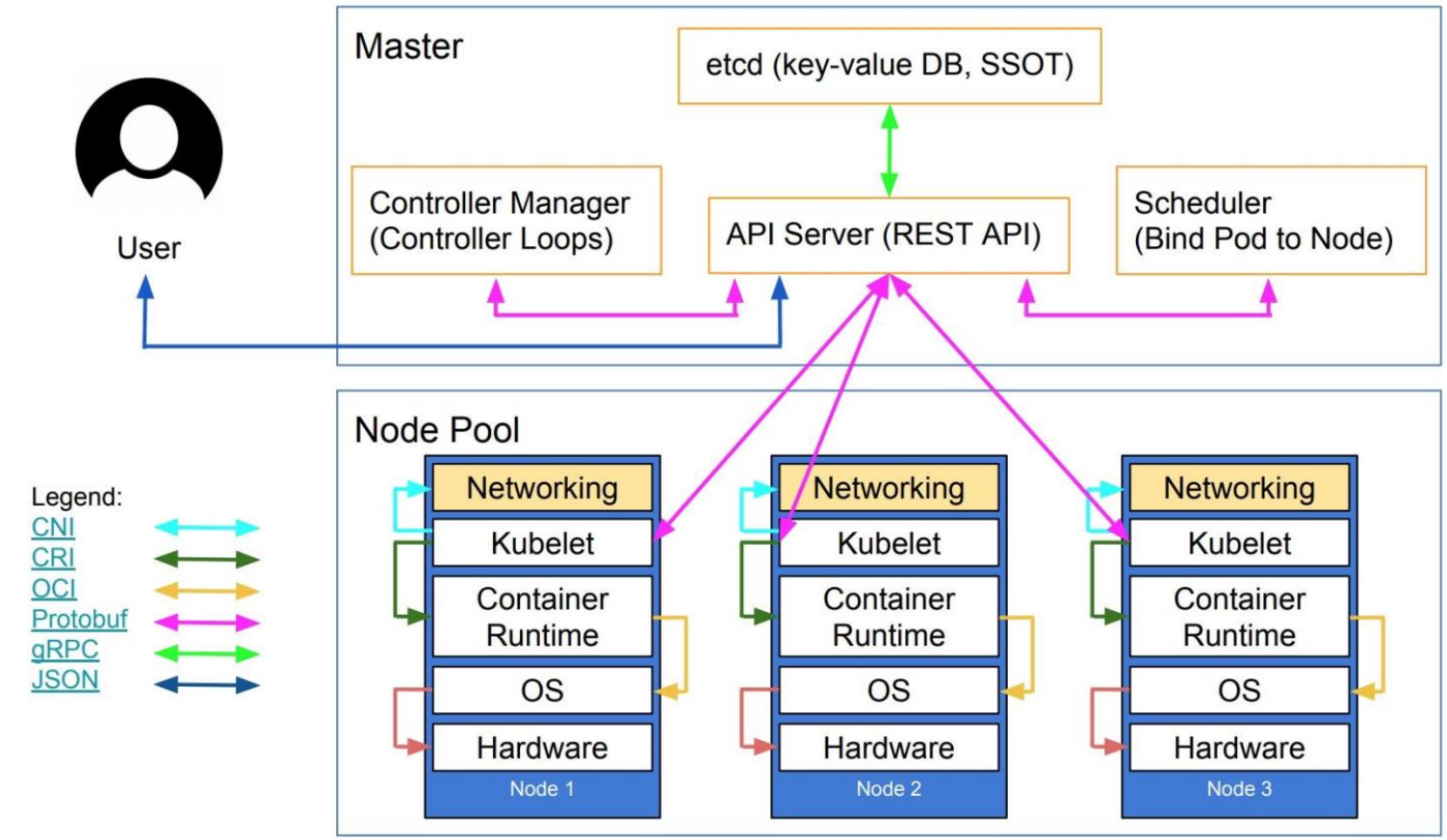
03 容器编排——Kubernetes

DSC Lab

How?

03 容器编排——Kubernetes

Architecture



Etcd——中心数据库

API Server——资源操作入口

Controller Manager——控制器中心

Scheduler——集群调度器

03 容器编排——Kubernetes

Kubernetes Control Plane（Master）

- Controller-manager

  是一系列控制器的集合，包括前面提到的不同对象都有一个相应的控制器。

```
$ cd kubernetes/pkg/controller/
$ ls -d */
deployment/             job/                    podautoscaler/
cloud/                  disruption/             namespace/
replicaset/             serviceaccount/         volume/
cronjob/                garbagecollector/       nodelifecycle/
replication/            statefulset/            daemon/
...
```

03 容器编排——Kubernetes

Kubernetes Control Plane（Master）

- Scheduler

  kube-scheduler 功能是调度 pod，为 pod 选择 node，将 nodeName 绑定到 pod 上。

  调度算法同样分为两步：

  1）**可行性检验。**寻找一组满足约束并且拥有足够可用资源的物理机。

  2）**评分。**对每个可用物理机按评分标准评分，选出分数最高的一台机器。

多调度器混合调度。得益于微服务架构，Kubernetes 支持多调度器，支持为不同 Pod 指定不同调度器。

03 容器编排——Kubernetes

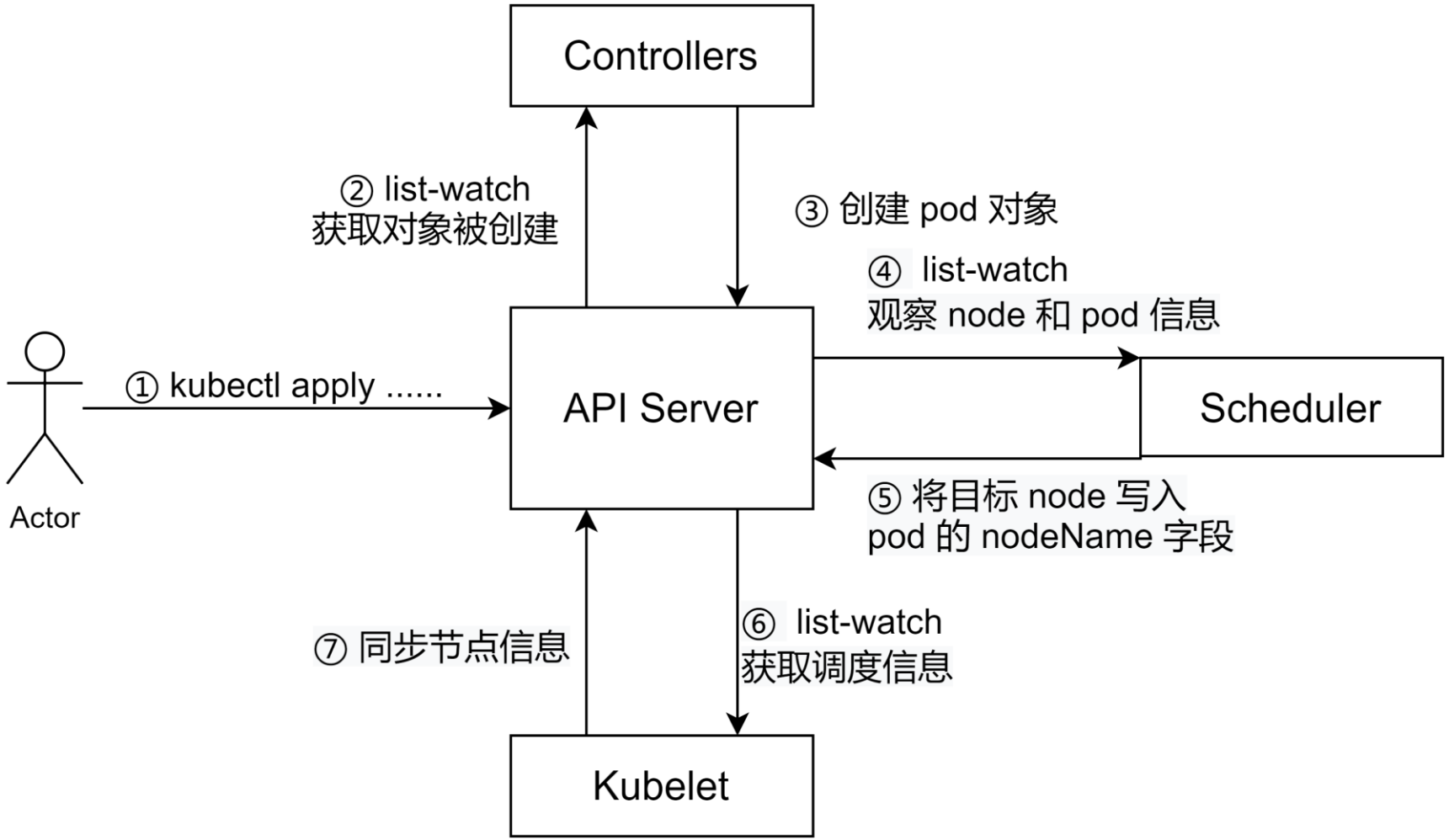
Kubernetes Node

- Kubelet

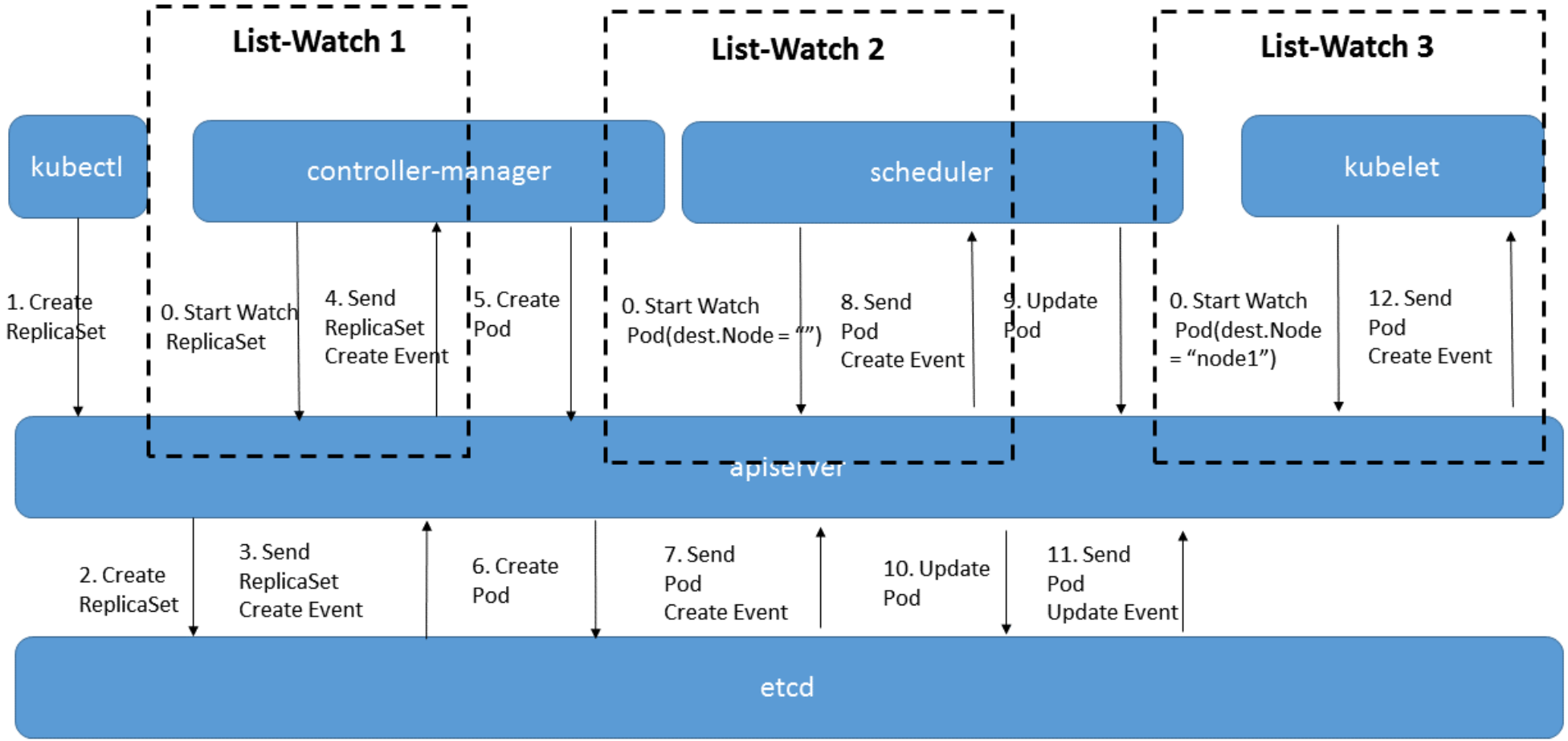
  kubelet 负责 Node 上 Pod 的创建、修改、监控、删除等全生命周期的管理。并且和 API Server 通信，上报 Node 的状态信息给 API Server。

03 容器编排——Kubernetes

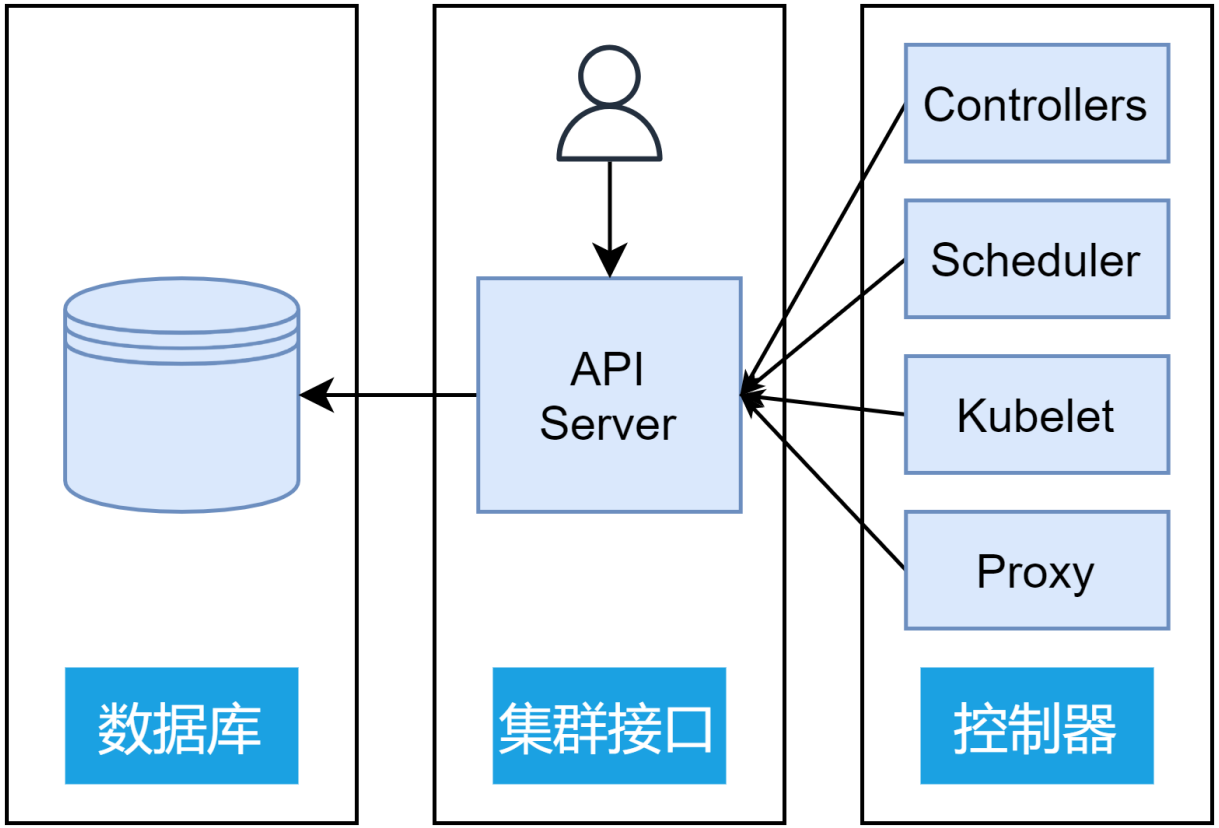
我们以一个应用部署为例：



容器编排——Kubernetes



03 容器编排——Kubernetes



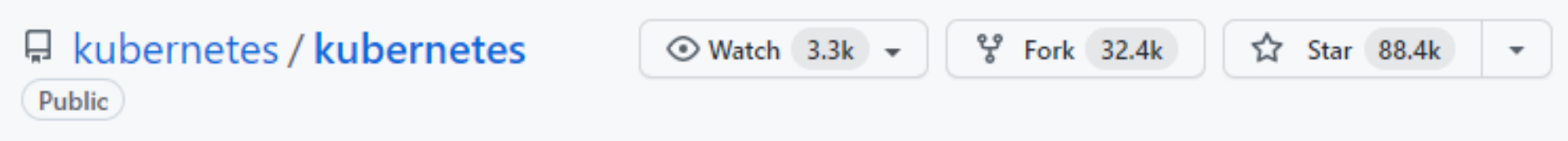
控制器模式。每个组件以控制循环的模式来运行，即：它们不断比较 etcd 中记录的**当前状态**和用户提交的 YAML 文件的**期望状态**，如果没达到期望状态，则进行相应的动作，称为调谐（Reconcile）。

03 容器编排——Kubernetes

以 **微服务架构** 和 **控制器模式** 构建核心组件无疑是非常成功的。

一方面，使整个系统变得清晰而统一，易于开发和拓展。

另一方面，这使得众多开发者能参与到如此庞大的一个项目中来，构建出非常有活力的生态。



目录

CONTENTS

04 总结

云原生是一种**构建**和**运行**应用程序的方法。

各个组件作为 Container 在 Kubernetes 中运行。

04 总结

产品	解决的问题	解决技术
Linux Container	对进程进行隔离和限制	Namespace + cGroups
Docker	容器镜像的打包	Docker image 分层
Borg	调度和运行容器	由 BorgMaste 管理
Kubernetes	自动化管理容器应用以及管理容器相关的一切	微服务架构 + 控制器模式