# Kubernetes(K8S)培训

开普云

2021年3月

# 培训目标

- 了解kubernetes的应用场景介绍
- 了解Kubernetes和Docker的 基础知识
- 掌握在K8S上部署容器化应用程序
- 了解弹性部署(扩容缩容)
- 掌握更新容器化应用程序
- 能调试容器化应用程序（日志排错）
- 了解容器仓库（harbor）
- 了解K8S监控（prometheus）

# 培训概要

- 应用治理背景介绍
- Docker和Kubernetes基础概念与常用命令
- 实战环节
  - 搭建集群
  - 发布应用
  - 应用管理
  - 应用调度
  - 扩容缩容
  - 持久化存储
- 容器镜像管理
- 其他相关依赖
- 容器方案

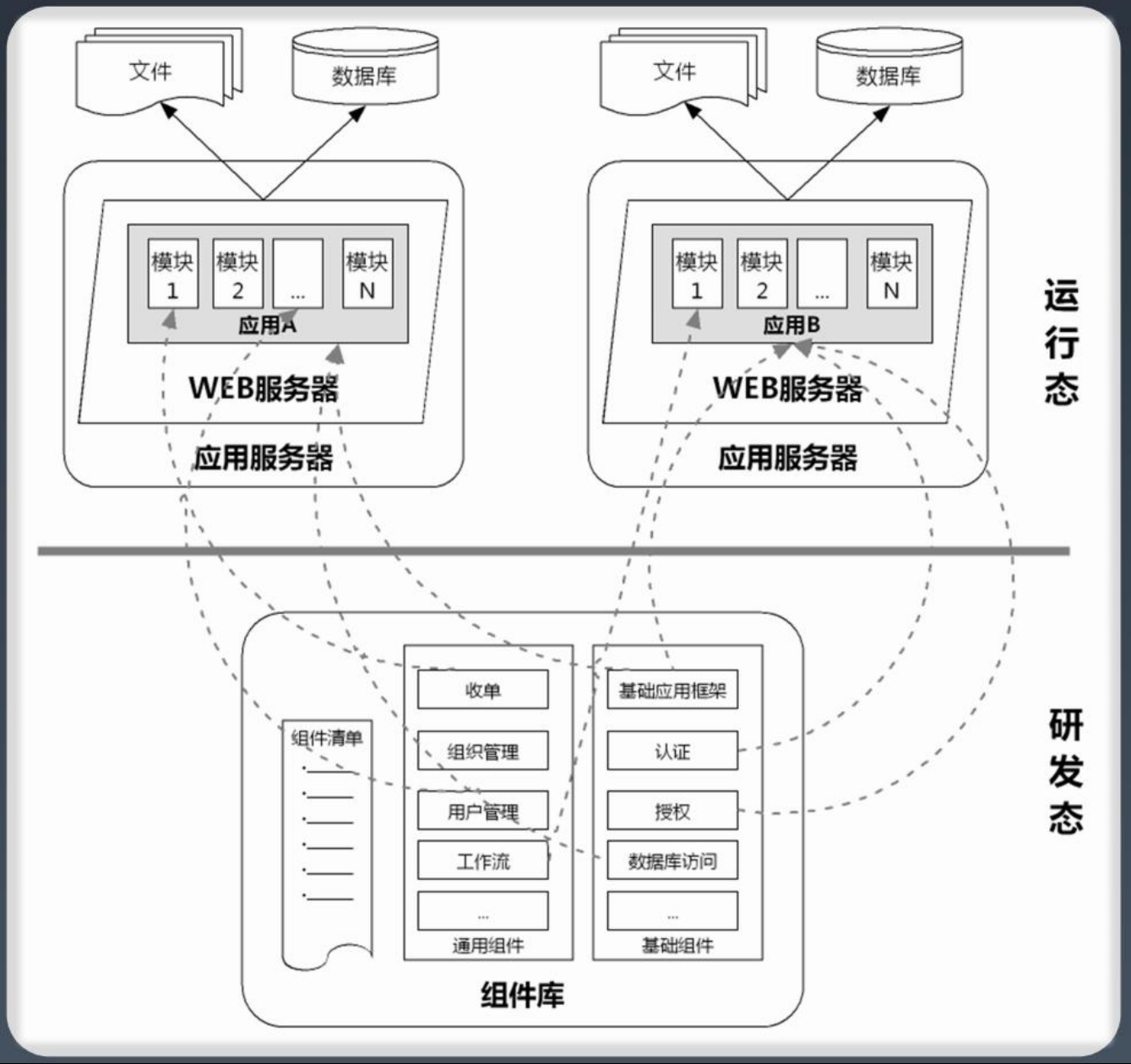
# 背景介绍

- 单体服务到微服务的治理
- 为什么是kubernetes?

# 服务治理发展简史

## 单体应用

**没有服务的概念，所以谈不上“服务治理”，复杂度来自于自身内部组件，由于组件化的管理需要，衍生出“组件治理”的需求。**



# 服务治理发展简史

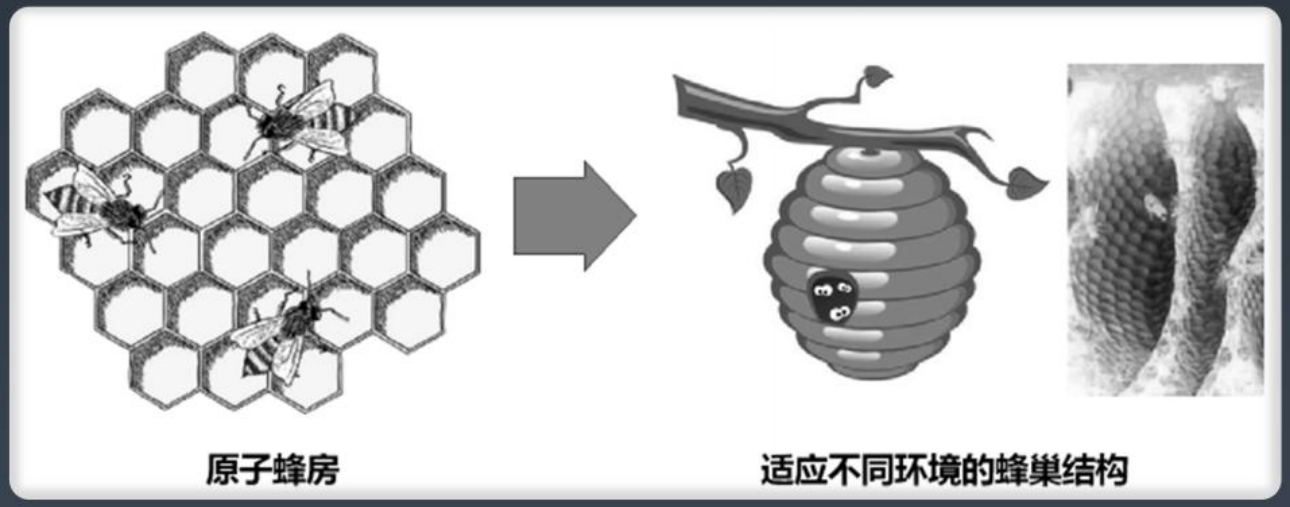
## 微服务

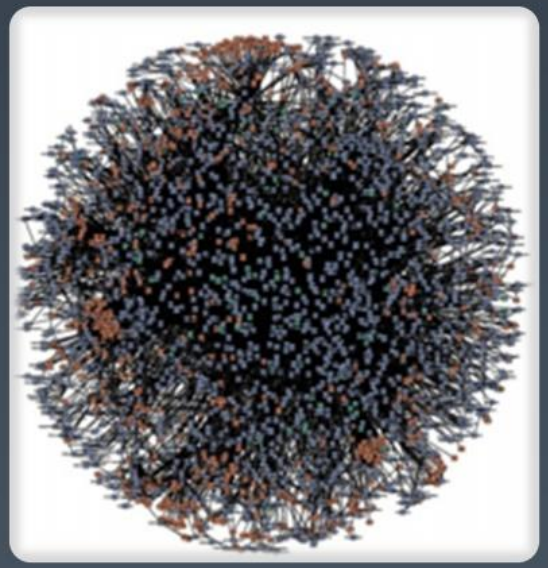
*“任何组织在设计一套系统时，所交付的设计方案在结构上都与该组织的沟通结构保持一致。”*

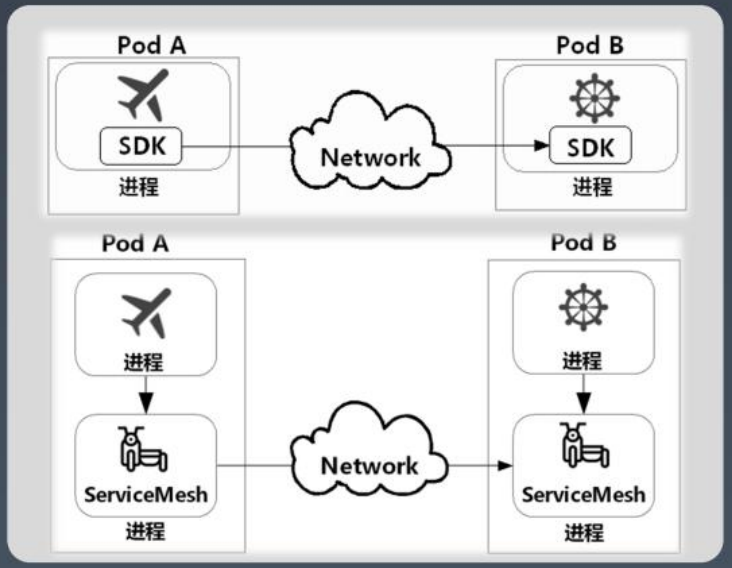
——康威定律

- **大平台、微服务**
- **和容器技术紧密结合**
- **量变导致质变，不仅仅是服务化架构的延伸**

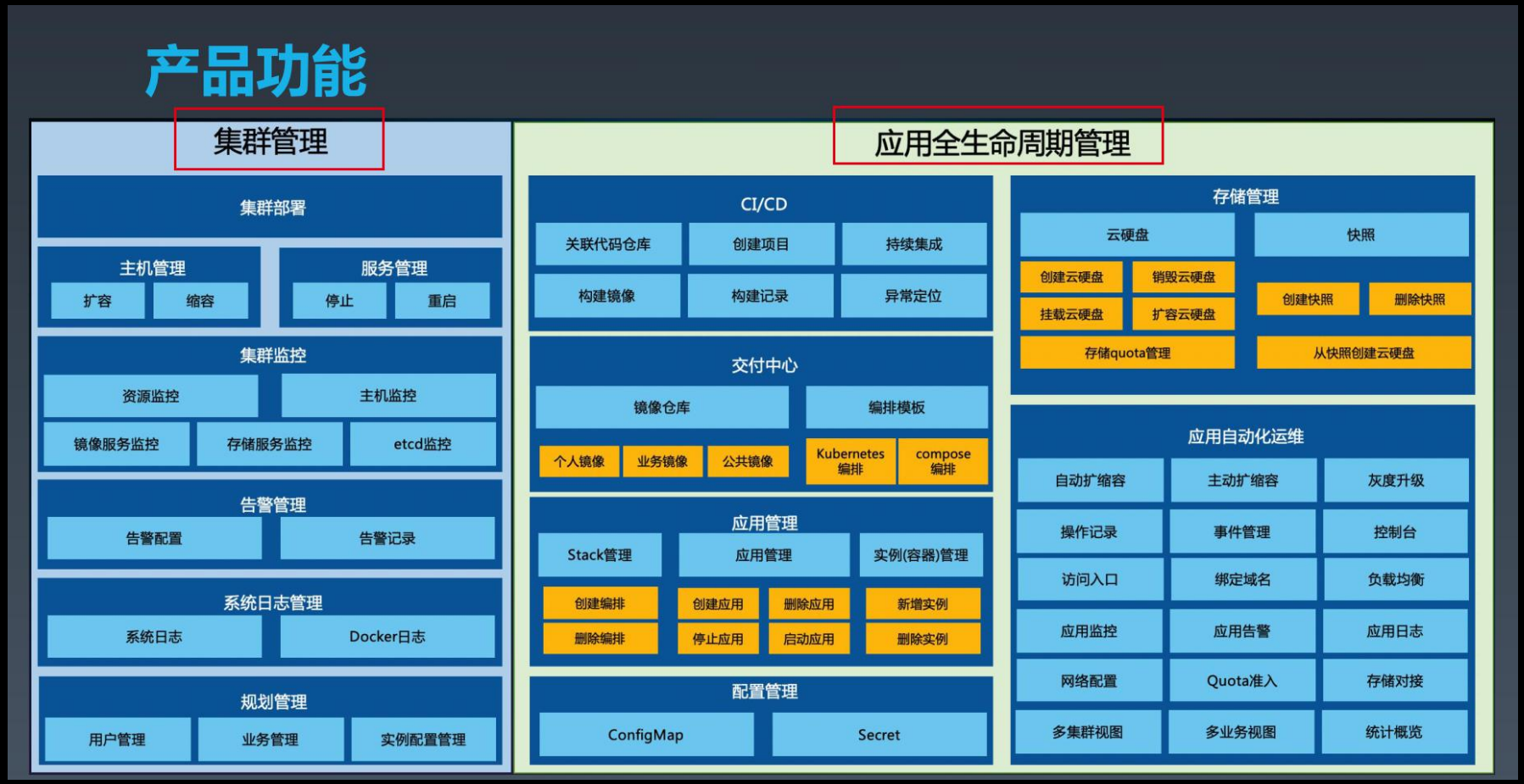
  组织架构、管理策略、研发模式、测试、运维等领域都要做出相应的调整，以为微服务架构的落地创造合适的“土壤”。
- **线上线下一体、全生命周期的立体化治理**
- **强调自动化、智能化**





# 为什么是kubernetes?



# 企业级容器云解决方案

# 可靠

## 平台容灾

- 所有组件无单点；
- 平台本身支持热升级；
- 组件自身HA机制，如docker；
- 多地域多可用区的容灾设计
- 管理机挂掉：对应用无影响
- 计算节点挂掉：跎机迁移

- 举例：1.4升级1.9版本
- Pod Hash发生变化
- Container名称发生变化，点分隔改为了下划线分隔
- 容器标签发生变化

  pause容器的标签io.kubernetes.container.name=POD改为
  io.kubernetes.docker.type=podsandbox
  io.kubernetes.container.restartCount改为
  annotation.io.kubernetes.container.restartCoun
- Cgroup目录结构发生变化，新增Pod层级

## 应用容灾

- 健康探针

① 存活探针
② 就绪探针

- 负载均衡
- 重启机制

① 区分异常原因
② 本地重启/跨机重启

- 黑名单机制

## 数据容灾

- 集群核心数据的备份和恢复

① Etcd
② 核心数据库

- 云盘机制保护应用数据

企业内部各个集群灰度运营。

# 安全

**容器技术**

Linux命名空间 → cgroups → GaiaStack cgroups: netrx/diskio

**镜像**

镜像签名 → 镜像安全扫描 → 镜像分层管理

**构建**

自动构建 → 集成测试 → 代码扫描 → 单元测试 → 系统测试

**部署**

部署准入 → Quota准入 → 严控特权 → 控制hostpath

**编排**

健康探针 → 资源预估 → 资源反馈 → 扩缩容管理 → 亲和性/反亲和性调度

**网络隔离**

租户隔离 → User隔离 → App隔离 → pod隔离 → IP隔离

**存储**

独占存储 → 共享存储 → 本地存储管理 → 日志管理

**平台安全**

自动化部署 → 配置统一管理 → RBAC → 操作审计 → 平台容灾 → 全方位监控/告警

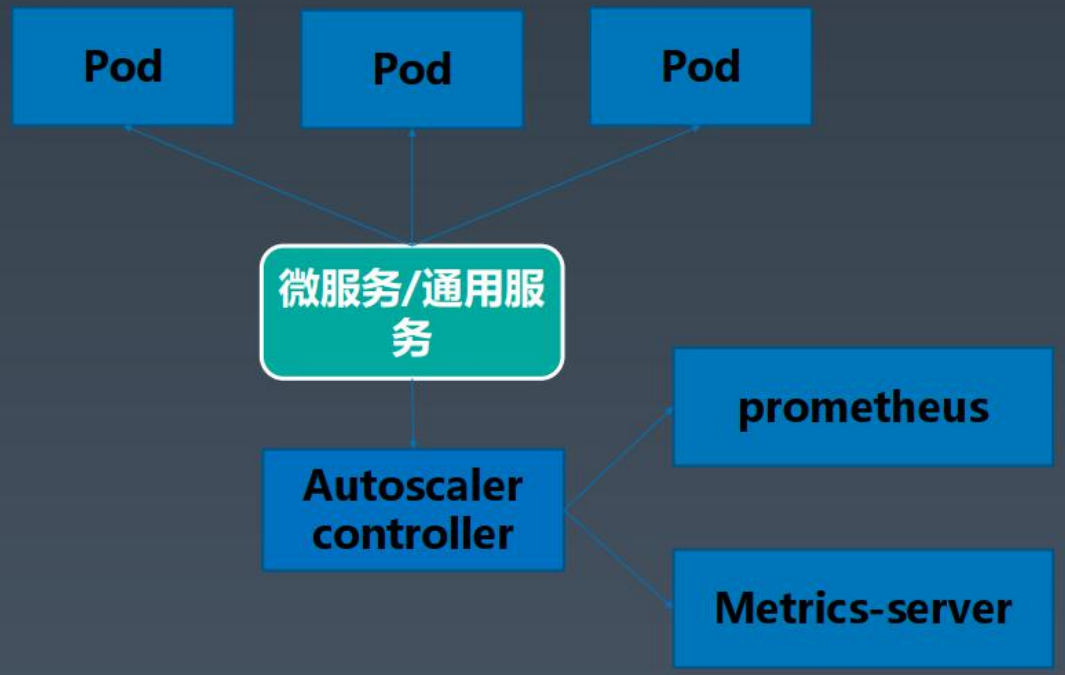
# 能力扩展：弹性伸缩

**APP弹性伸缩：**

- 主动扩缩容
  - 扩容可以指定新版本
  - 缩容可以定点裁撤

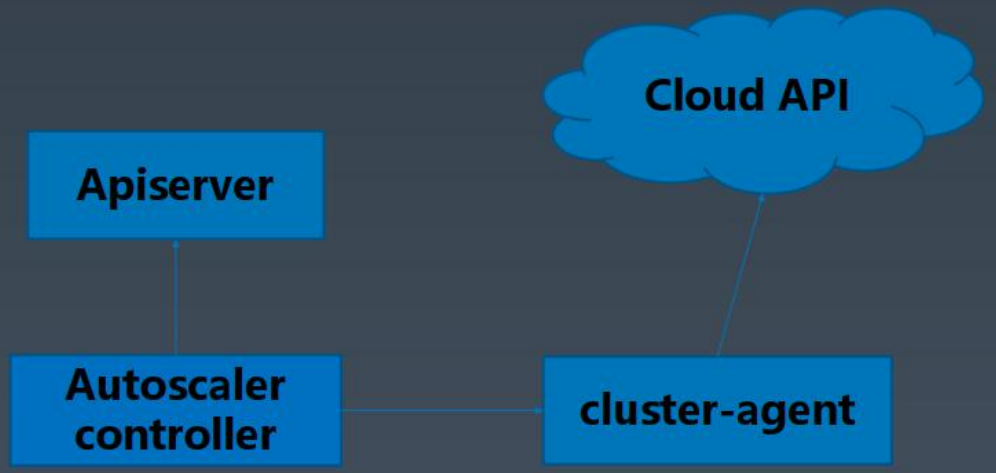
自动扩缩容

- **资源阈值**
- **自定义指标阈值**
- **实例个数范围**
- **周期性自动伸缩**

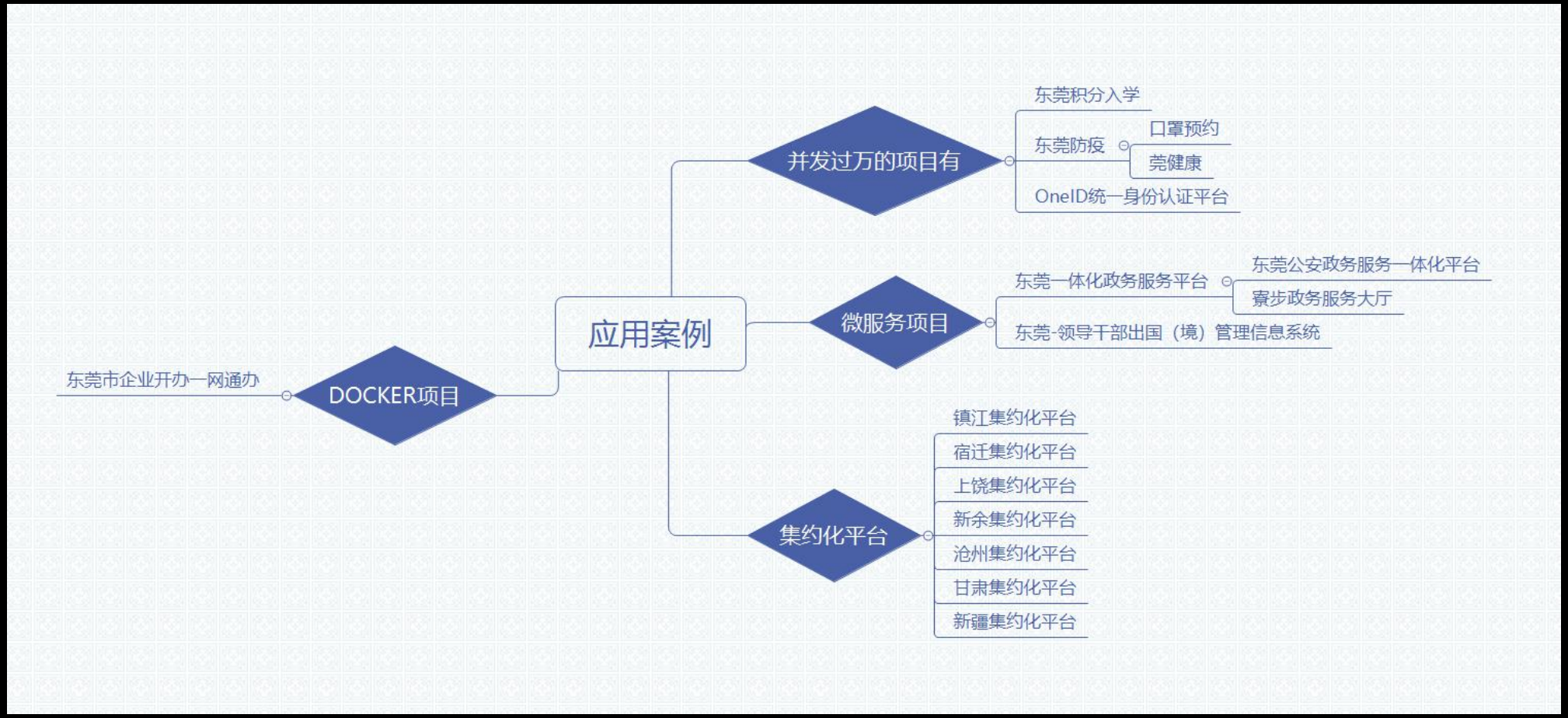



集群弹性伸缩：

- 监控节点资源使用率
- 自动迁移低负载Node上的Pod，完成缩容
- 一定数量Pod因资源不足pending时，自动扩容



# 应用案例


应用案例
并发过万的项目有
东莞积分入学
东莞防疫
口罩预约
莞健康
OneID统一身份认证平台
微服务项目
东莞一体化政务服务平台
东莞公安政务服务一体化平台
寮步政务服务大厅
东莞-领导干部出国（境）管理信息系统
DOCKER项目
东莞市企业开办一网通办
集约化平台
镇江集约化平台
宿迁集约化平台
上饶集约化平台
新余集约化平台
沧州集约化平台
甘肃集约化平台
新疆集约化平台

# 应用案例-口罩预约-举例-1

### 3、运维保障能力

一个地市的口罩预约系统，从上线开始，就注定要承受巨大的访问压力，系统架构设计应能够在大规模并发之下响应及时，并且支持弹性扩容。

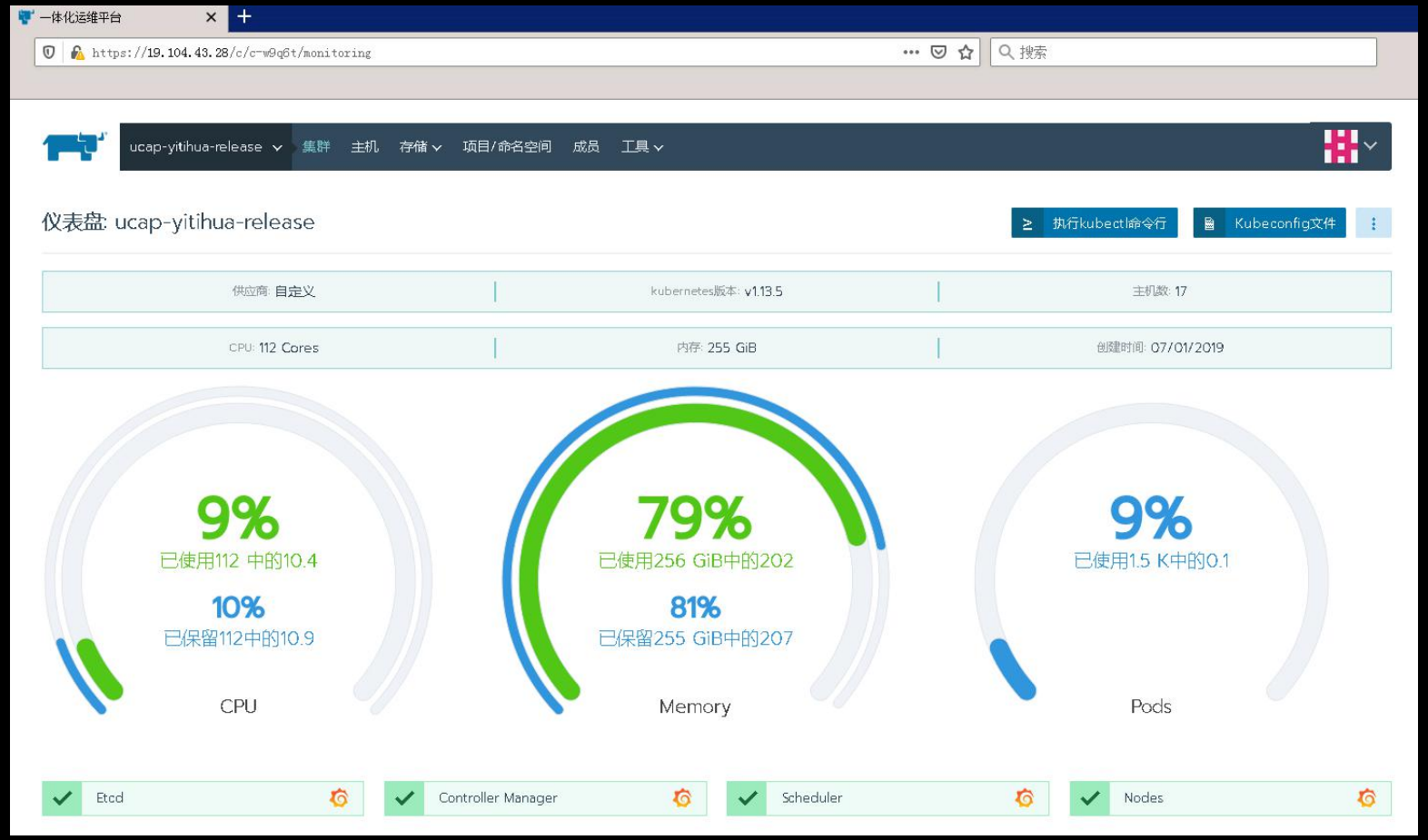
广东某地市口罩预约系统采用Kubernetes技术构建，由15个节点组成集群，包括Kubernetes调度高可用节点，work节点，ingress服务、私有仓库（代码发布），缓存节点，查询节点，持久化数据库节点等，并采取了根据访问量使用HPA进行自动扩容、数据库延时写入、前端分布式缓存等一系列电商级别的部署技术。整套系统使用Prometheus实时监控，运维人员根据监控数据及时预判，提前处理，保证了数百万市民始终能正常使用包括口罩预约在内的应用功能。

并发高峰截图


系统压力截图


集群状态截图

# 应用案例-一体化政务服务平台-举例2



Welcome - Nexus Repository | 一体化运维平台 | Grafana - Cluster

https://19.104.43.28/k8s/clusters/c-w9q6t/api/v1/namespaces/cattle-prometheus/services/http:access-grafana:80/proxy/d/

# Cluster

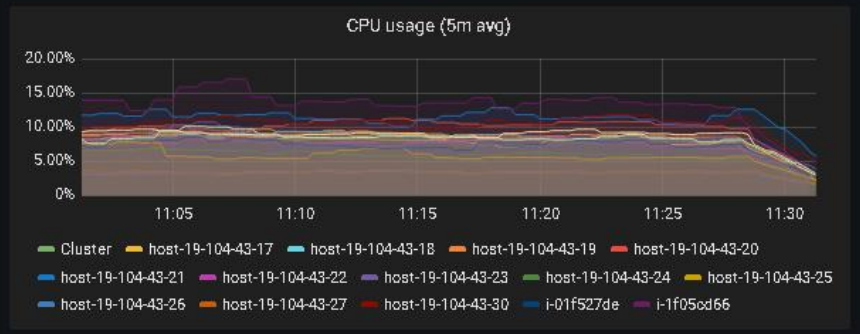
Last 30 minutes Refresh every 1m

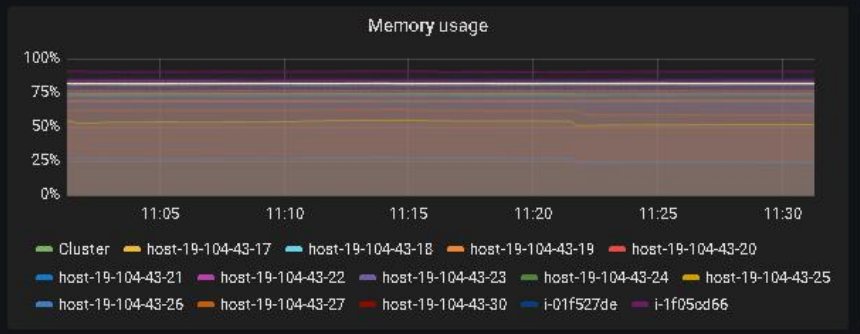
Node All

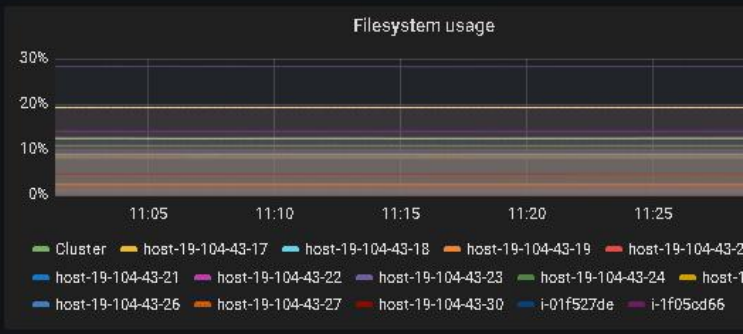
## Total usage

| CPU usage (5m avg) | Memory usage | Filesystem usage |
|---|---|---|
| 1.74% | 74% | 8.47% |

| Used | Total | Used | Total | Used | Total |
|---|---|---|---|---|---|
| 2.36 cores | 136.00 cores | 224.23 GiB | 303.03 GiB | 727.14 GiB | 8.39 TiB |




## Node *(7 panels)*

## Deployments *(4 panels)*

Cluster

Last 30 minutes Refresh every 1m

Pods Running

155

Pods Pending

3

Pods Failed

1

Pods Succeeded

0

Pods Unknown

0

## Containers

Containers Running

192

Containers Waiting

0

Containers Terminated

0

Containers Restarts (Last 30 Minutes)

0

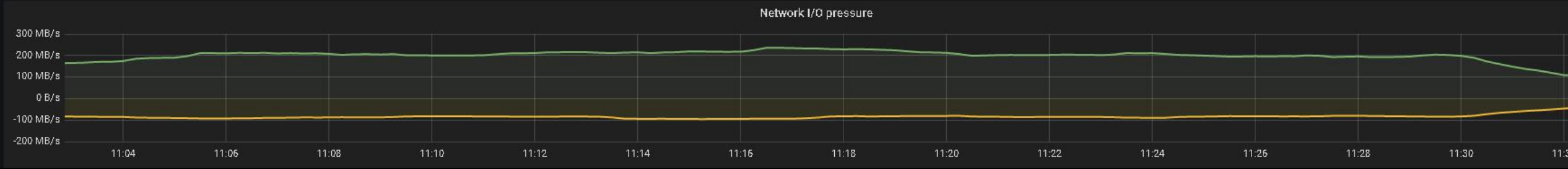
CPU Cores Requested by Containers

12

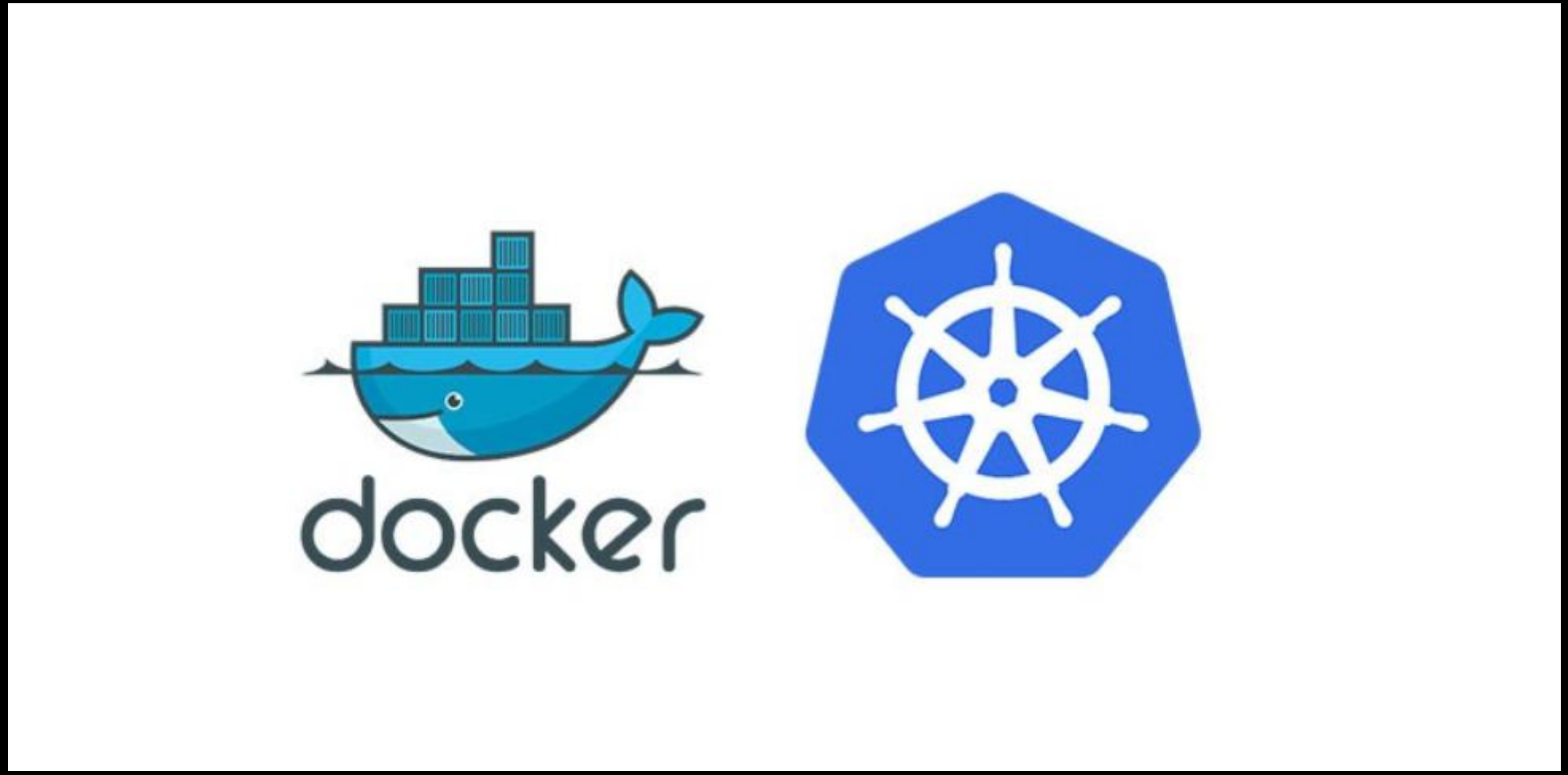
Memory Requested By Containers

225 GB

## Network I/O pressure



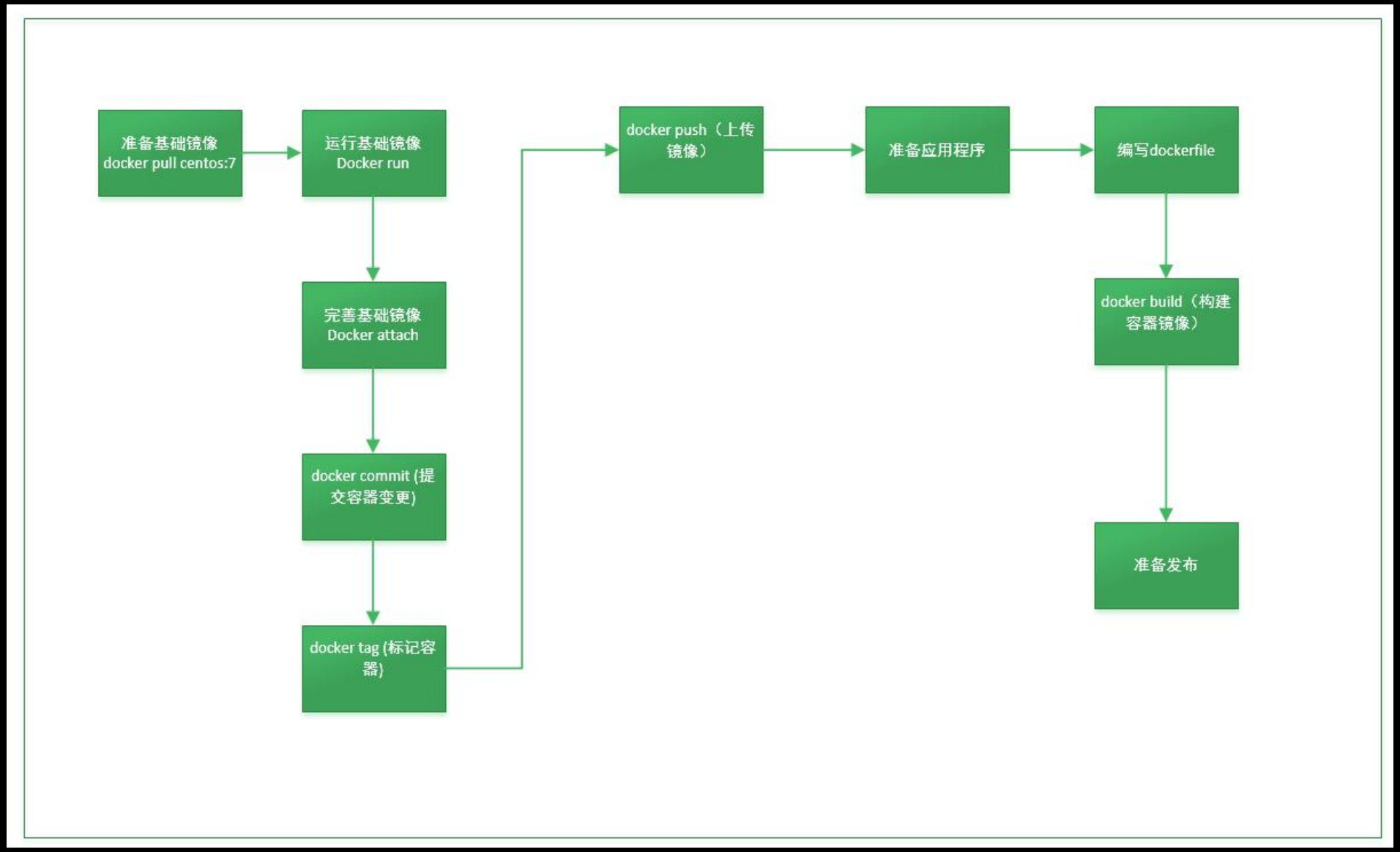
# Docker和Kubernetes基础



# 构建容器化程序相关基础和流程

- docker pull(拉取镜像)
- docker push（上传镜像）
- docker logs（查看容器日志）
- docker attach（进入容器）CTRL+P+Q(保存)
- docker commit (提交容器变更)
- docker tag (标记容器)
- docker build（构建容器）
- docker run （运行容器）
- docker save -o (导出容器)
- docker load (导入容器)

# Docker构建镜像示意图



# Dockerfile模板样例

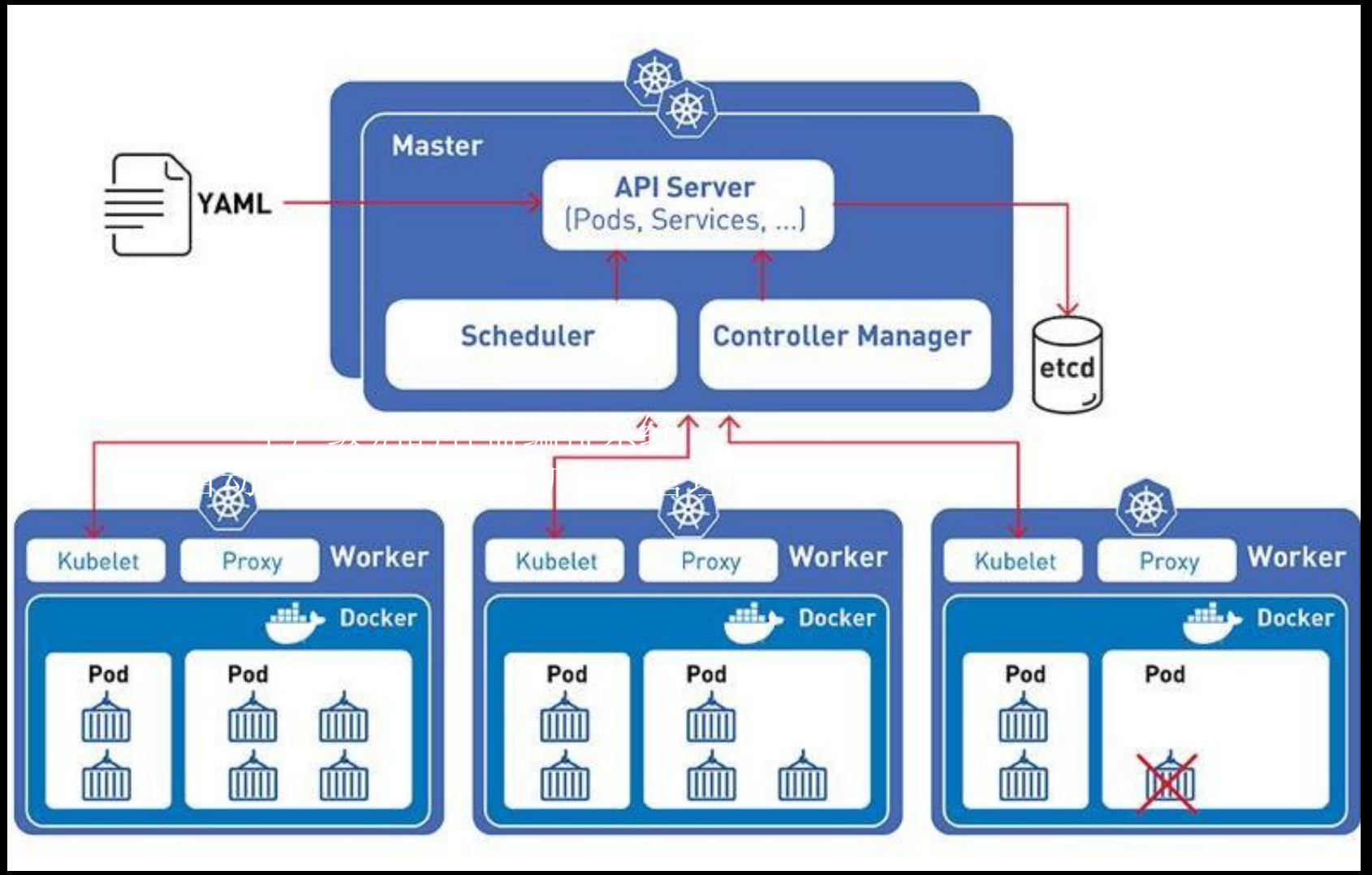
```
FROM ucapirr.com:5000/base/tomcat8:v1
ENV TZ=Asia/Shanghai
ENV LANG=zh_CN.UTF-8
RUN localedef -c -f UTF-8 -i zh_CN zh_CN.utf8
ENV LC_ALL zh_CN.UTF-8
WORKDIR /usr/local/tomcat
RUN mkdir /data/attachment
ADD ./apps /usr/local/tomcat/webapps/apps
ENTRYPOINT ["/usr/local/tomcat/bin/catalina.sh","run"]
```

# K8S架构
# K8S是什么

生产级别的容器编排系统

自动化的容器部署

扩展和管理



# K8S服务器角色说明

Master

Master:集群控制节点，负责整个集群的管理和控制。
API Server:提供接口，资源增删改查入口。
Controller Manager:所有资源对象的自动化控制中心。
Scheduler:负责资源调度。
Etcd:master的持续状态存储。

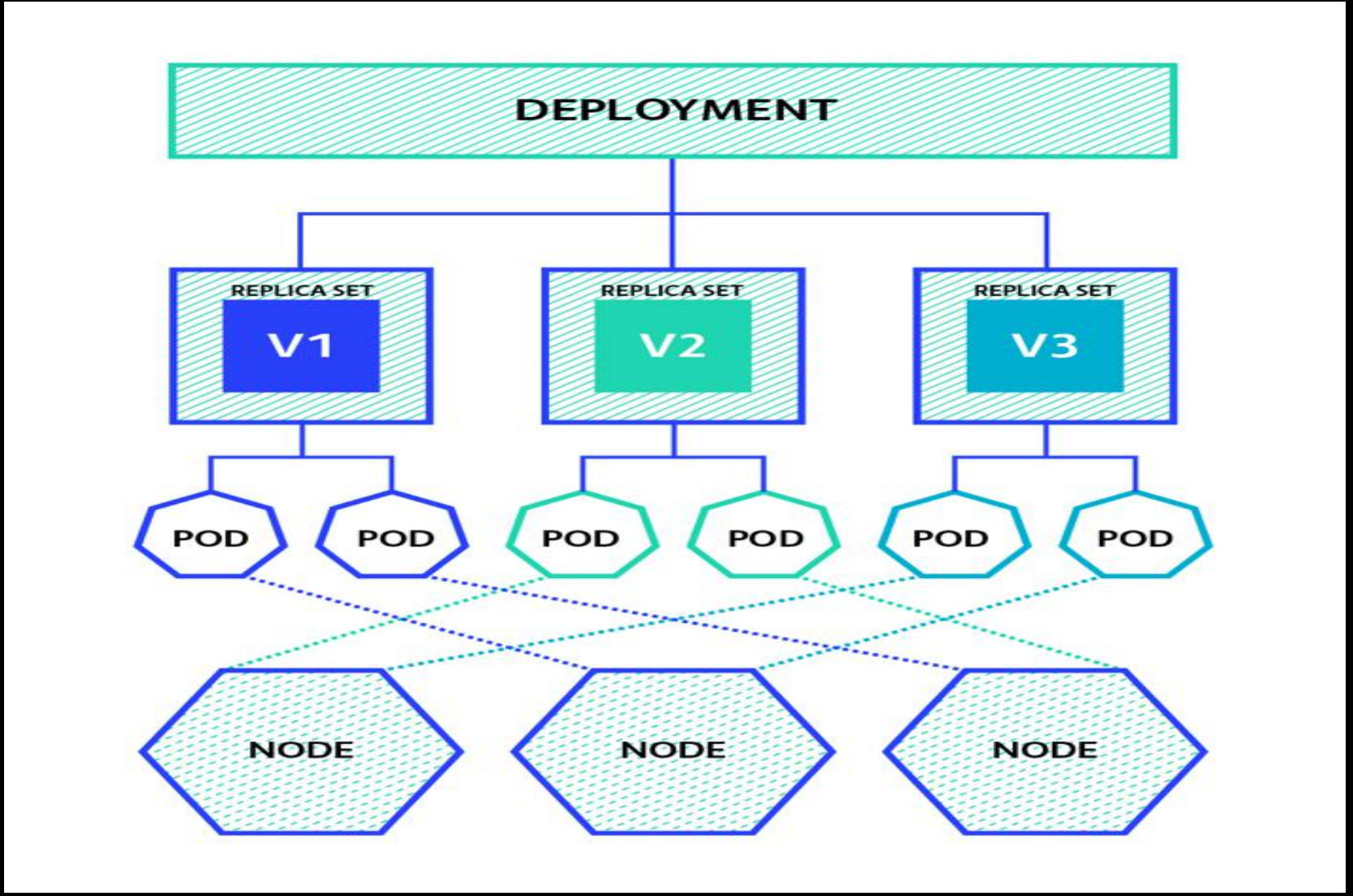
Node
（worker）

Node：工作节点，听从master的工作分配。
Kubelet：Pod容器创建、启停、集群管理等任务。
Kube-proxy:实现service的通信与负载均衡组件。
Docker：docker引擎，负责本机容器创建和管理工作。

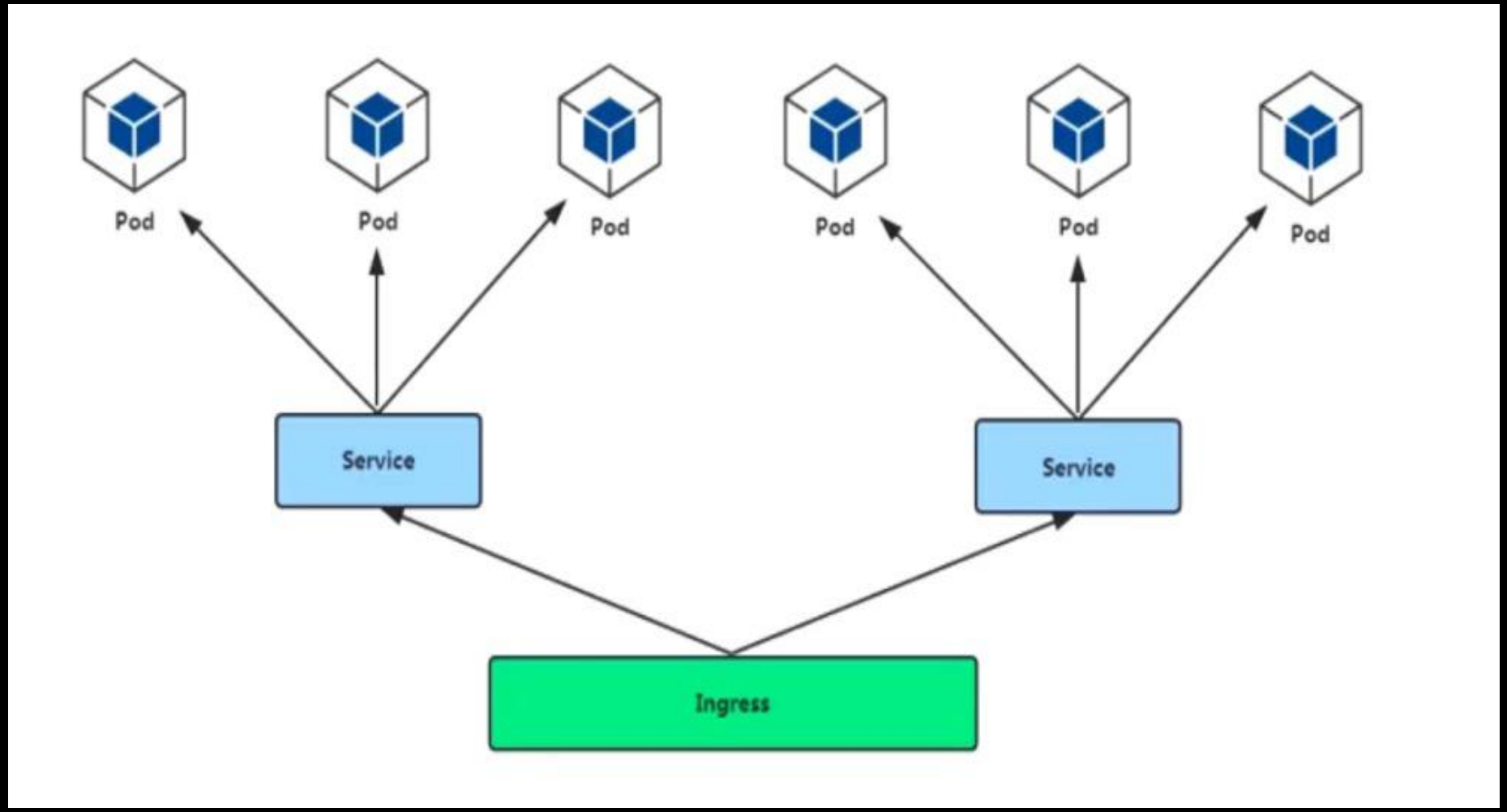
# K8S常用资源对象

| 类别 | 名称 |
| --- | --- |
| 资源对象 | Pod、ReplicaSet、ReplicationController、Deployment、StatefulSet、DaemonSet、Job、CronJob、HorizontalPodAutoscaler |
| 配置对象 | Node、Namespace、Service、Secret、ConfigMap、Ingress、Label、CustomResourceDefinition、ServiceAccount |
| 存储对象 | Volume、Persistent Volume |
| 策略对象 | SecurityContext、ResourceQuota、LimitRange |

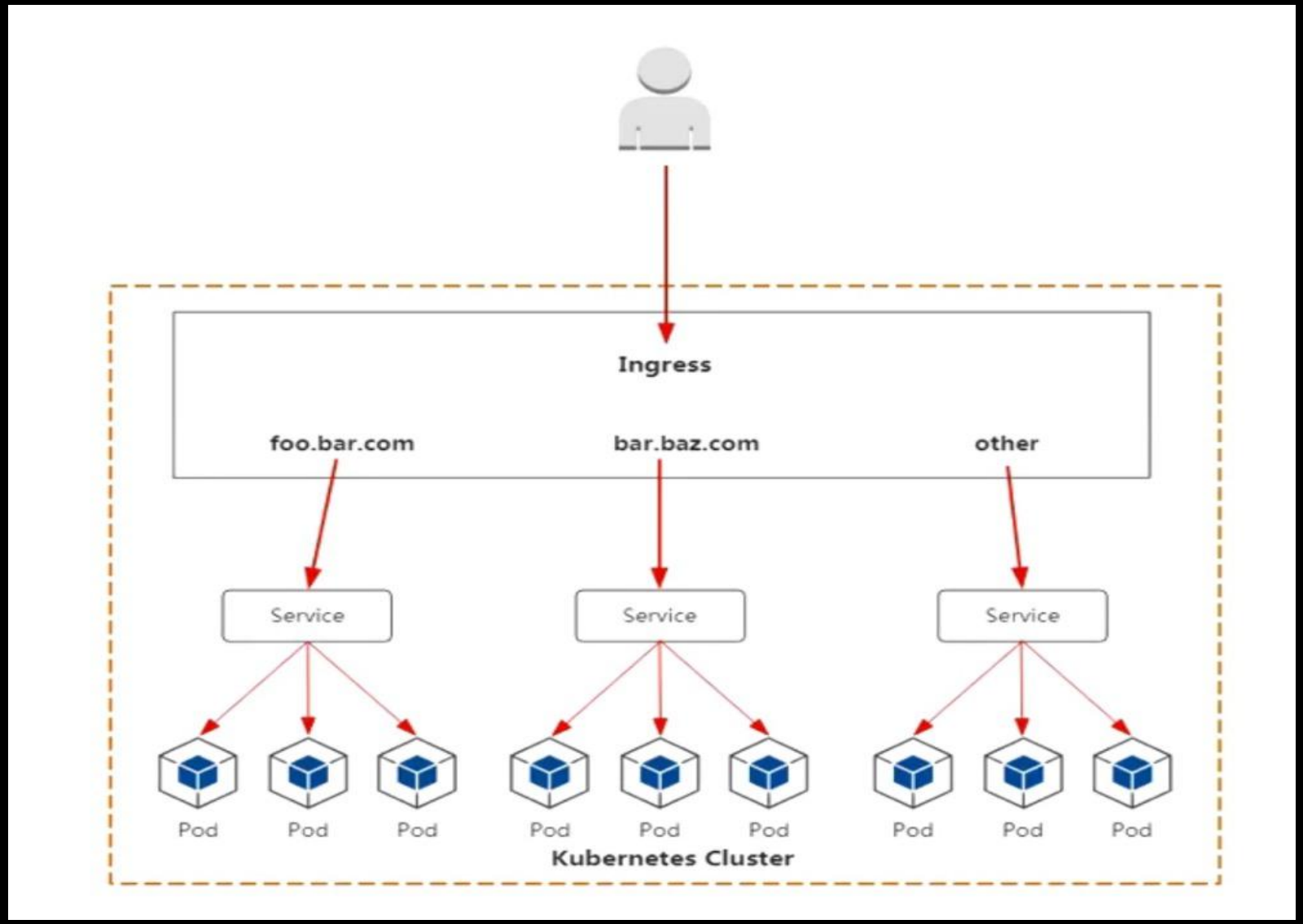
# Pod与deployment的关系



# Pod与Ingress的关系



# Ingress Controller



# K8S常用命令

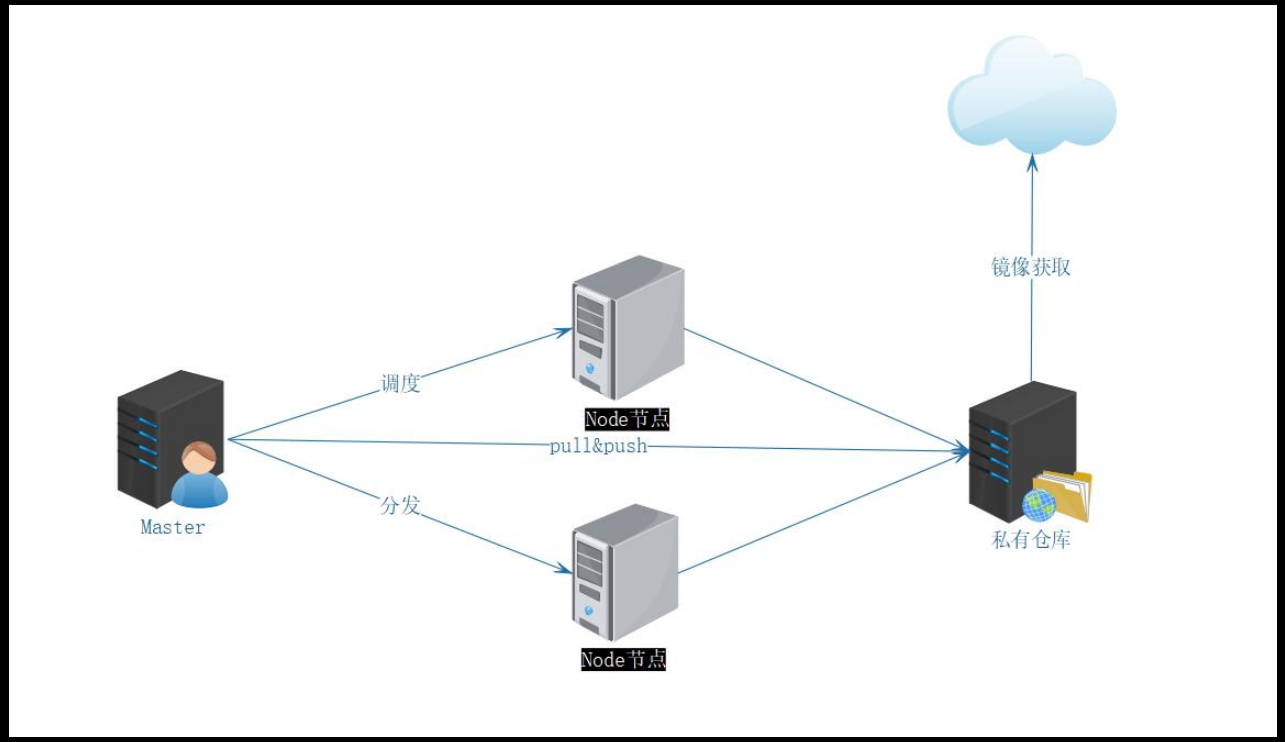
- kubectl

# K8S常用命令

- # kubectl get pods  #查看pod状态
- # kubectl get pods -o wide  #查看pod详细状态
- # kubectl get svc #查看服务状态端口信息
- #kubectl get ep #查看负载均衡节点
- # kubectl get deploy  #查看deployment资源
- #kubectl logs containerID
- #kubectl get ns #查看命名空间

# 实验环节

- 环境要求：
- 配置要求：3台主机，2核 4G内存 50G磁盘
- 系统要求：CentOS7或者Ubuntu18.4



# 准备工作

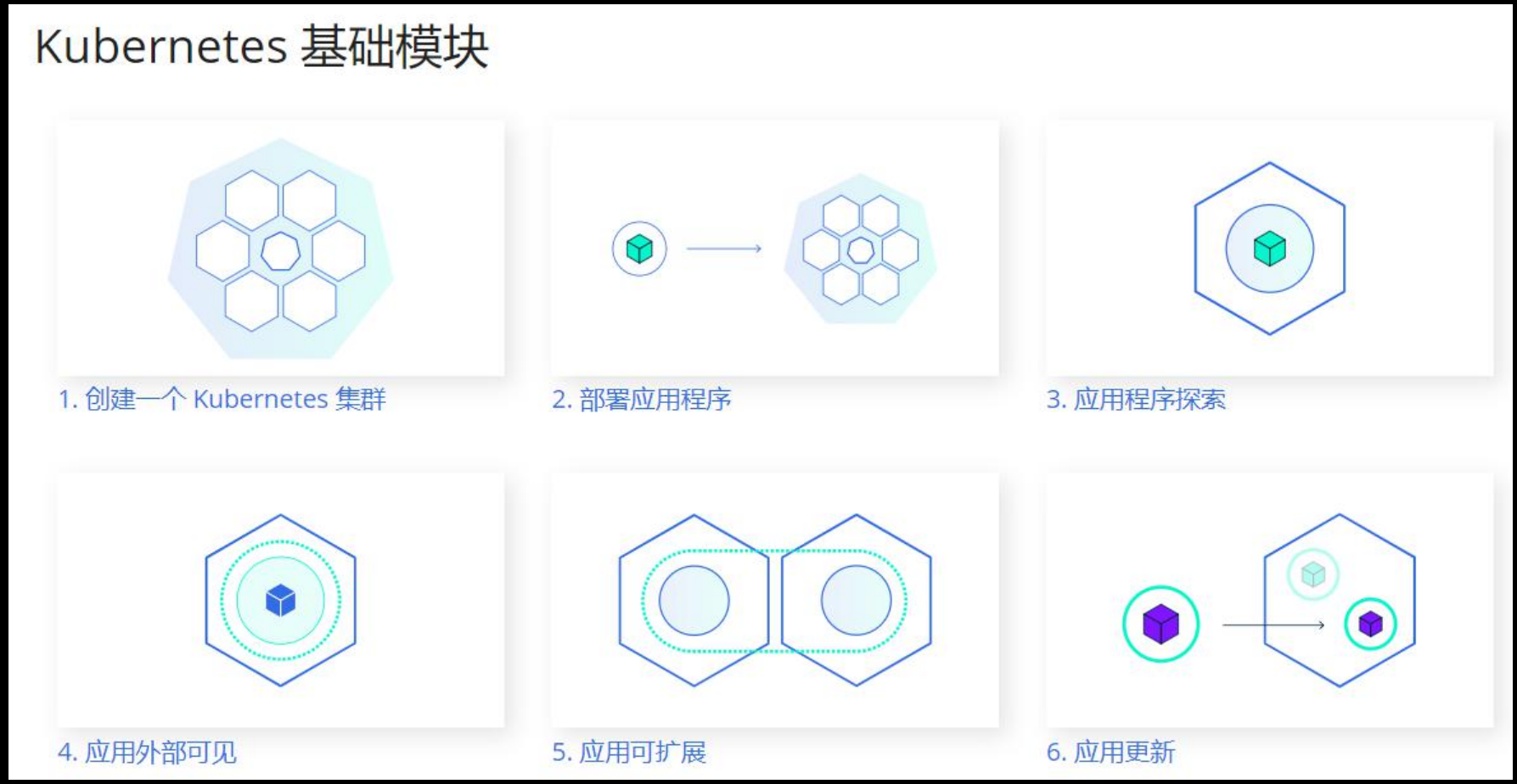
- 准备私有镜像仓库
- 基于harbor搭建

  必备：docker-compose

# 搭建K8S集群

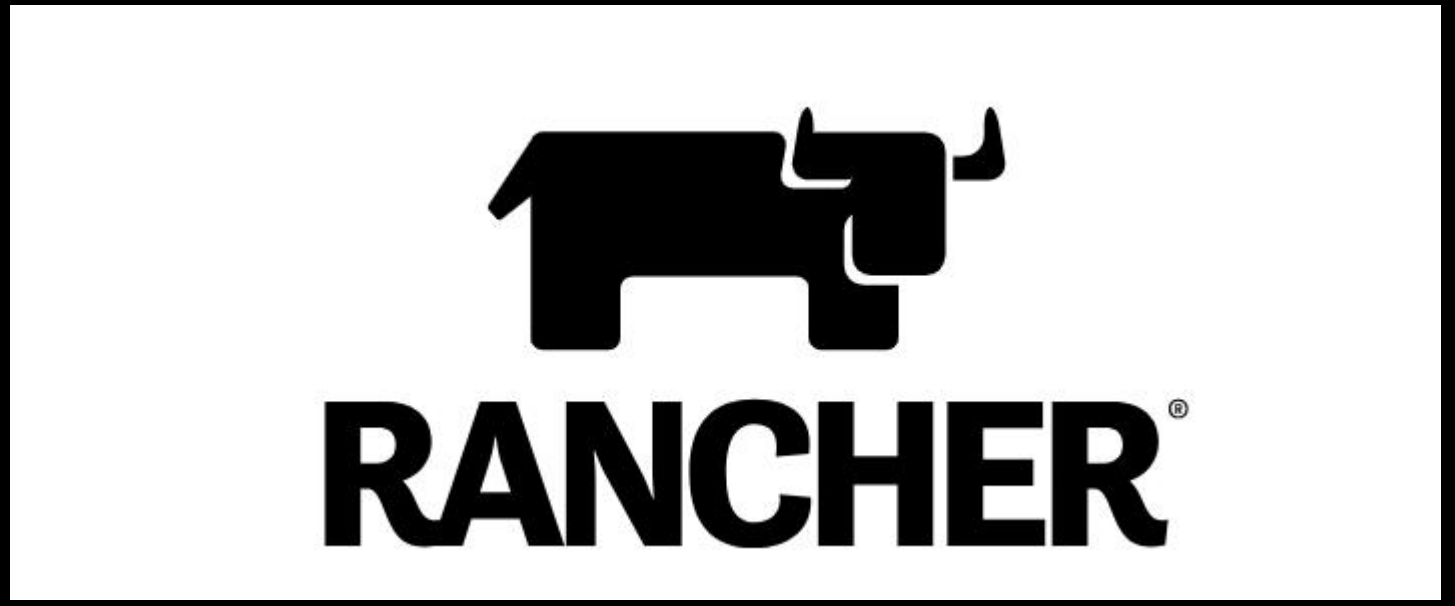
- docker run -d --restart=unless-stopped \
- -p 80:80 -p 443:443 \
- -v /data/rancher:/var/lib/rancher \
- rancher/rancher:v2.4.9

# 实验流程


Kubernetes 基础模块
1. 创建一个 Kubernetes 集群
2. 部署应用程序
3. 应用程序探索
4. 应用外部可见
5. 应用可扩展
6. 应用更新

# Rancher（图形界面的K8S）

- Rancher 是为使用容器的公司打造的容器管理平台。
- Rancher 简化了使用 Kubernetes 的流程，开发者可以随处运行 Kubernetes（Run Kubernetes Everywhere），满足 IT 需求规范，赋能 DevOps 团队。



# 容器镜像管理（仓库）

- 开源企业级容器镜像仓库
- 由Vmware中国团队设计和开发
- 通过图形界面可以轻松浏览和搜索镜像仓库，能够方便的管理项目和权限。
- 通过项目组织权限。
- 支持接入企业AD/LDAP
- 所有操作都会被追踪记录用于审计
- 支持镜像安全扫描

# K8S监控-Prometheus

# 培训结束!

留个疑问，
是不是掌握了kubernetes就实现了对微服务的应用治理
呢？

# Service Mesh—后 Kubernetes 时代的微服务

- Kubernetes 的本质是通过声明式配置对应用进行生命周期管理。
- 而 Service Mesh 的本质是提供应用间的流量和安全性管理以及可观察性。