Turing College 图灵学院

Kubernetes基础-架构与核心组件详解

目录

Kubernetes背景

什么是Kubernetes?

Kubernetes产生的背景

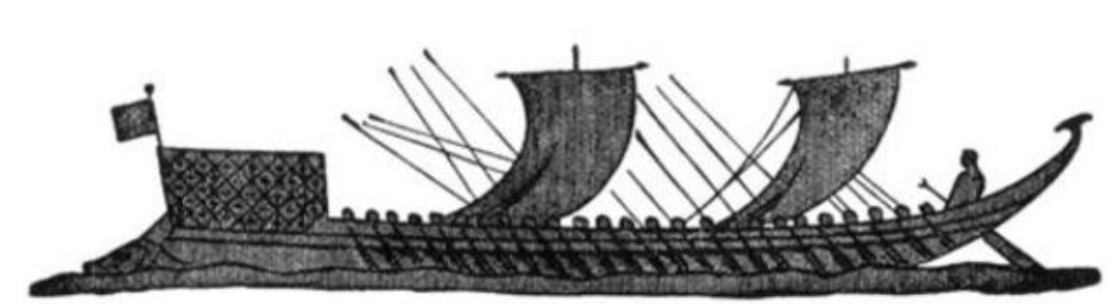
Kubernetes的发展历程和应用现状

什么是Kubernetes?

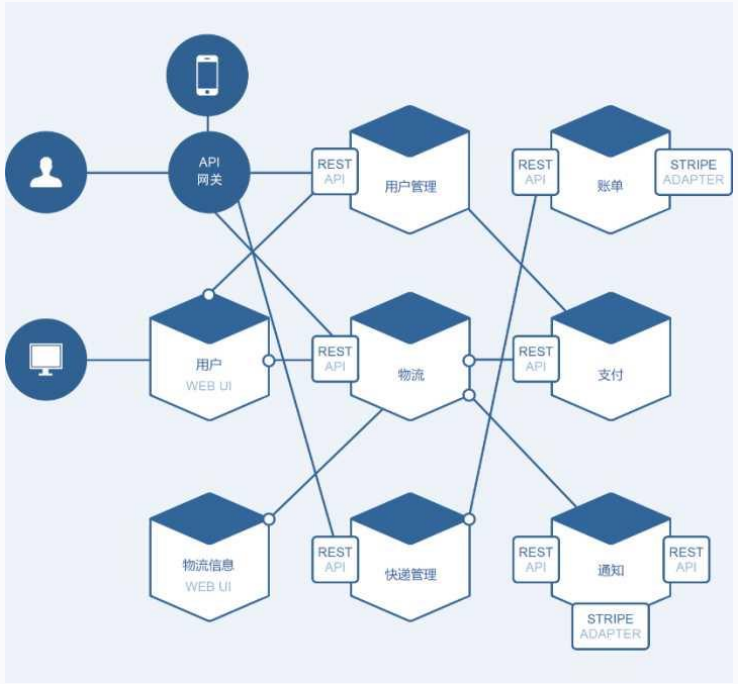
- 生产级别的**容器编排**系统
  - 自动化的容器部署、扩展和管理

- 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
  - 借鉴Google内部的集群管理系统“Borg”（2015 EuroSys)和”Omega”（15年的生产环境应用经验）
  - Google于2014年开源，捐献给云原生计算基金会（CNCF， Cloud Native Computing Foundation）

- Kubernetes意思
  - 希腊语
  - 驾驶员（Pivlot)或舵手(Helmsman)
  - 一般简称k8s(Kubernetes)



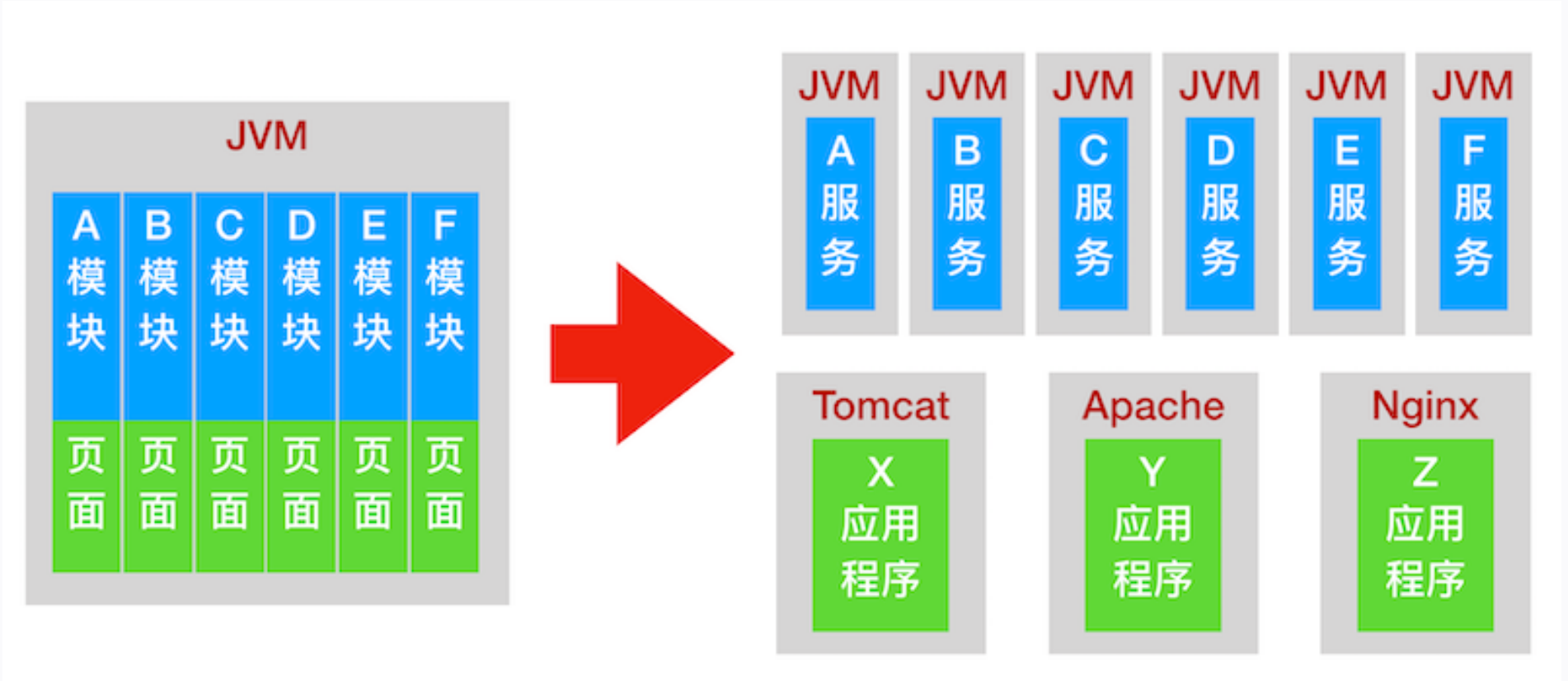
Kubernetes产生背景



微服务

容器

微服务

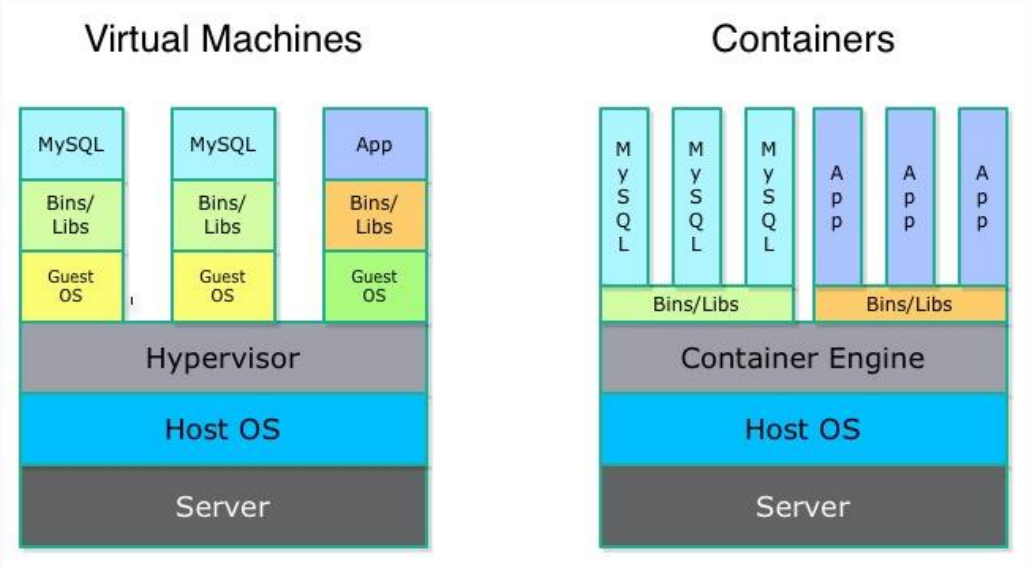

JVM
A模块
B模块
C模块
D模块
E模块
F模块
页面
页面
页面
页面
页面
页面
JVM
A服务
JVM
B服务
JVM
C服务
JVM
D服务
JVM
E服务
JVM
F服务
Tomcat
X应用程序
Apache
Y应用程序
Nginx
Z应用程序

容器

- 什么是容器?
  - 一系列隔离运行的进程，提供了一种轻量操作系统层面的虚拟化技术
  - 每个容器拥有自己的PID, User, UTS, Network栈命名空间等
  - 与传统VM比具有启动快、性能损耗小、更轻量等优点

- Docker是目前使用最广，最成熟的容器技术

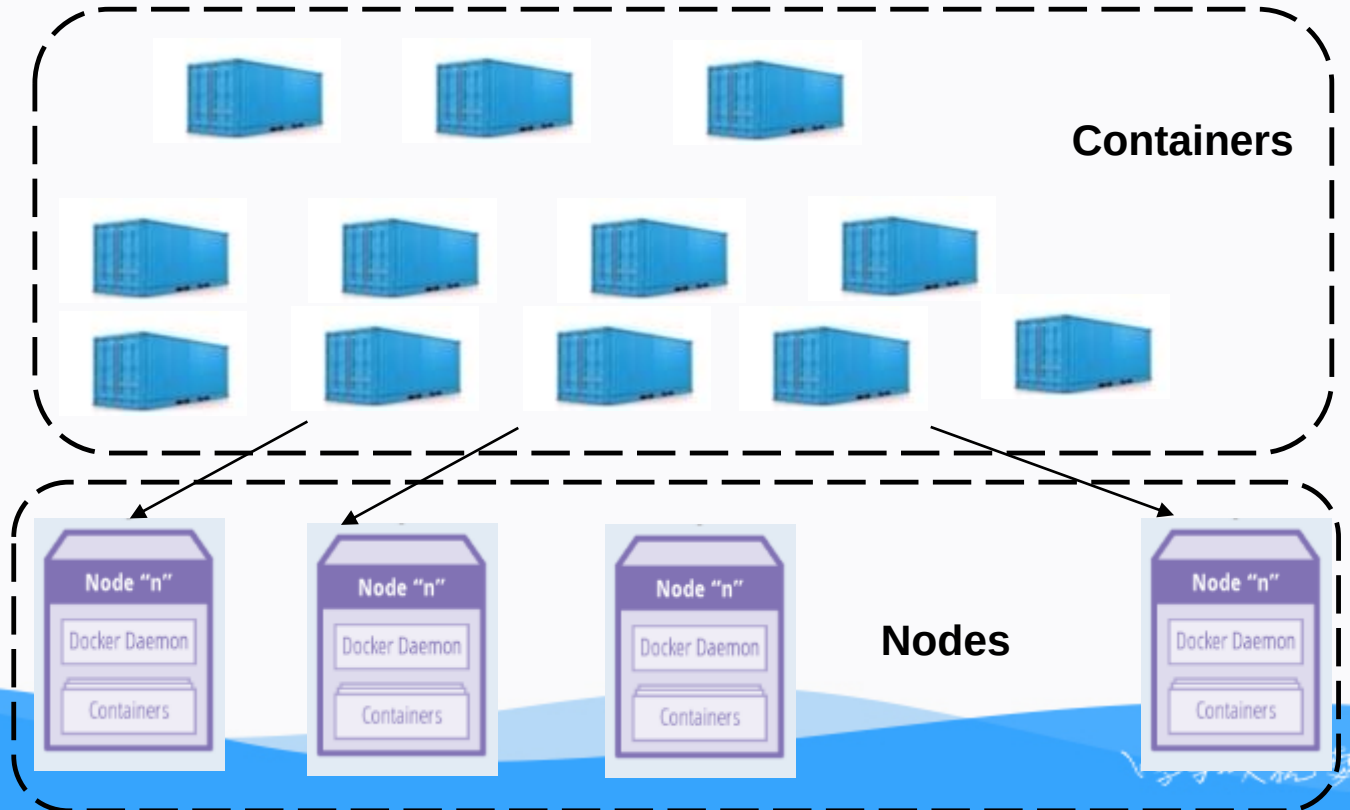
- K8S默认使用Docker引擎
  - 也可使用Rkt(coreos)，或其他遵循CRI(continer runtime interface)的容器引擎，例如Containerd等)



容器化系统面临的挑战

- 容器解决了应用打包、部署、运行的问题
  - 一次构建，随处运行（Build, Ship and Run Any App, Anywhere ）

- 容器的挑战
  - 跨机器部署
  - 资源调度
  - 负载均衡
  - 自动伸缩
  - 容错处理
  - 服务发现



容器编排

- 容器编排（Container Orchestration）
  - 以容器为基本对象进行管理
  - 协同容器共同实现应用功能

- 容器编排系统主要功能
  - 容器调度（Placement, health checking..）
  - 资源管理 (CPU、GPU、Memory…)
  - 服务管理 (Service Discovery、Load Balance…)

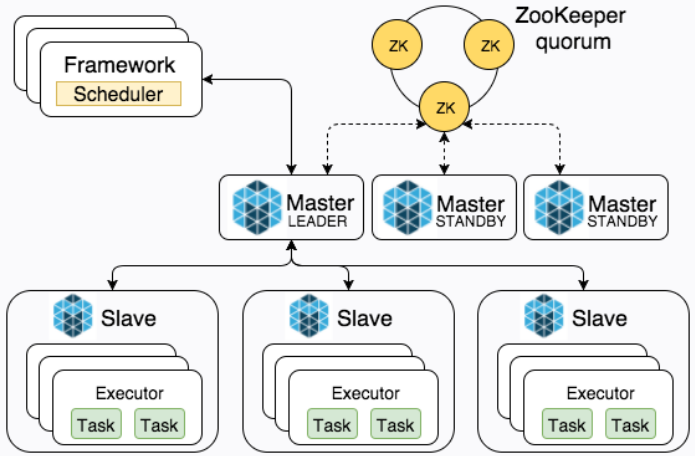
容器编排大战

Docker Swarm

Docker Inc

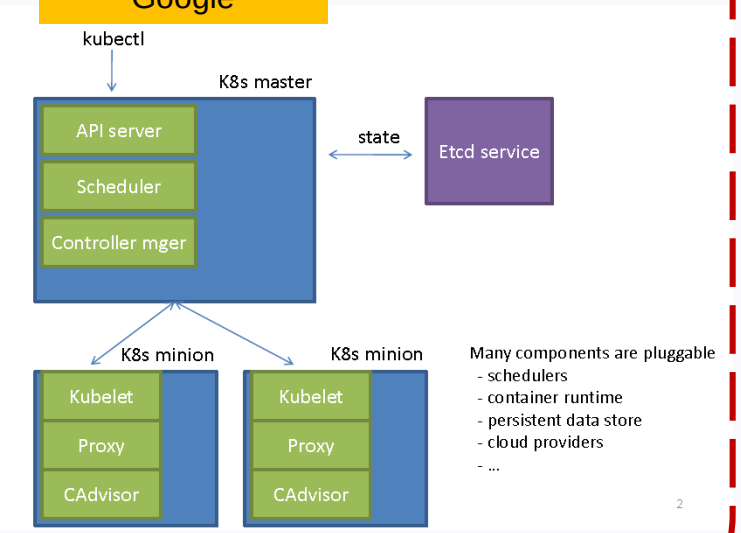
Mesos

Mesosphere



Kubernetes

Google



Kubernetes发展历史

- 基于Google‘s 内部的Borg” 和“ Omega”系统
  - 2014年10月由Google正式开源。
  - 2015年7月22日发布1.0版本，在OSCON（开源大会）上发布了1.0版本。
  - 目前最新版为1.18版
- 不但开源，还由开放组织（CNCF）负责管理
- 受到众多厂商的积极支持，社区十分活跃
  - **Google, RedHat, CoreOS, Microsoft (Deis), IBM, Huawei**

Kubernets背景总结

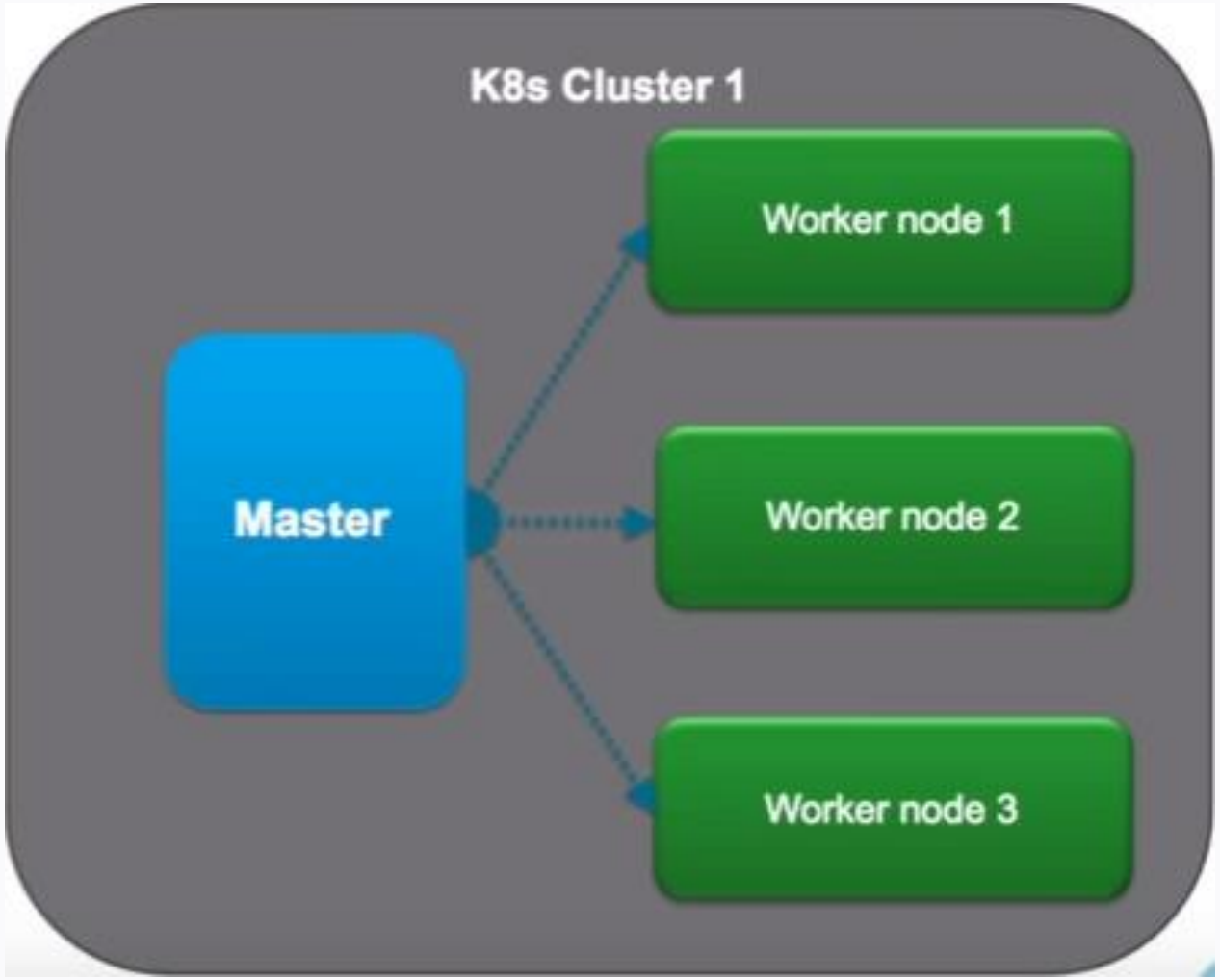
- Kubernetes 是Google开源的生产级容器编排系统，是 Google 多年大规模容器管理技术 Borg 的开源版本
  - 基于容器的应用部署、维护和滚动升级
  - 负载均衡和服务发现
  - 跨机器和跨地区的集群调度
  - 自动伸缩
  - 无状态服务和有状态服务
  - 广泛的 Volume 支持
  - 插件机制保证扩展性

- Kubernetes 发展非常迅速，已经成为容器编排领域的领导者

目录

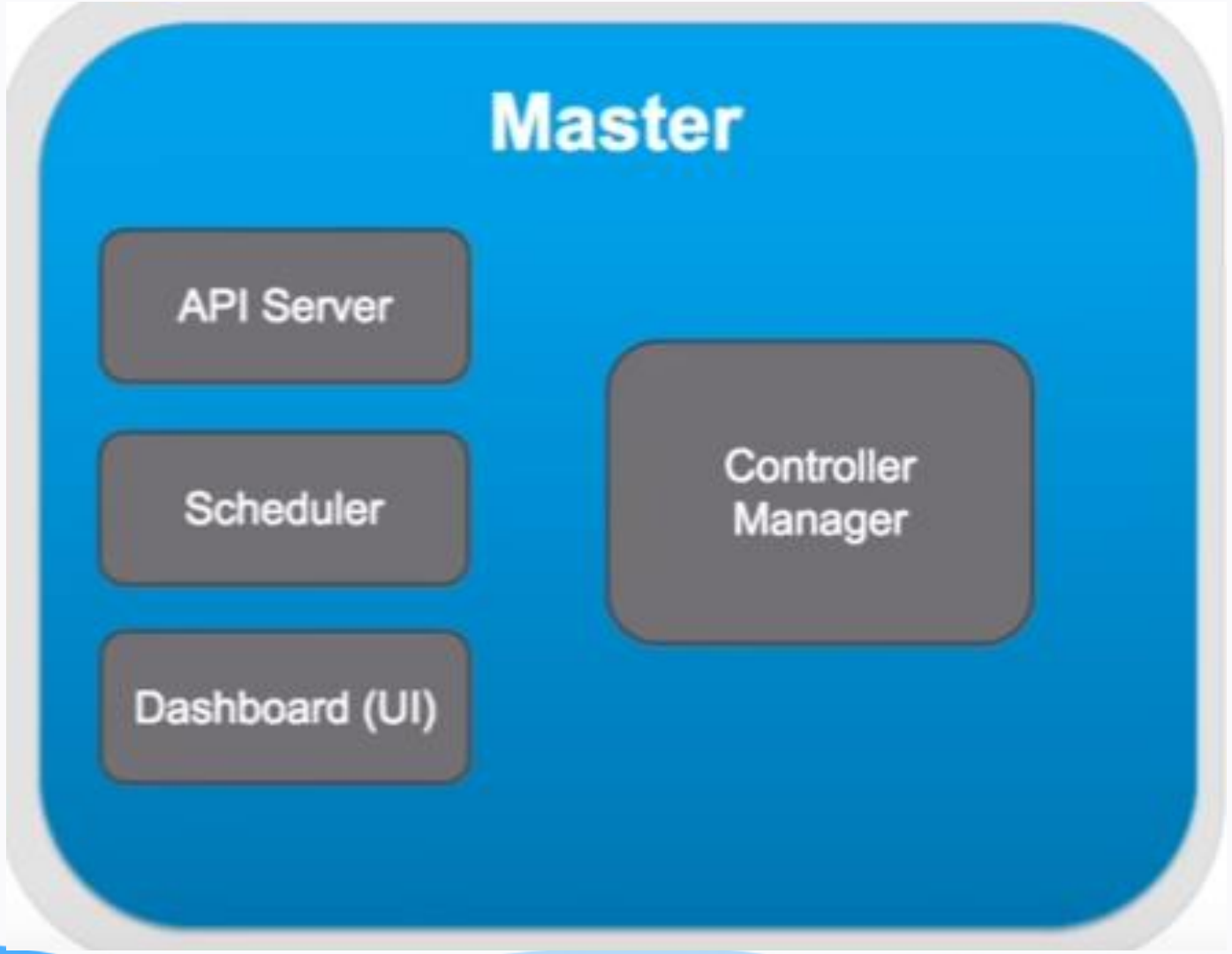
Kubernetes架构

- K8s Cluster
  - Master
  - Worker Node(Minion)



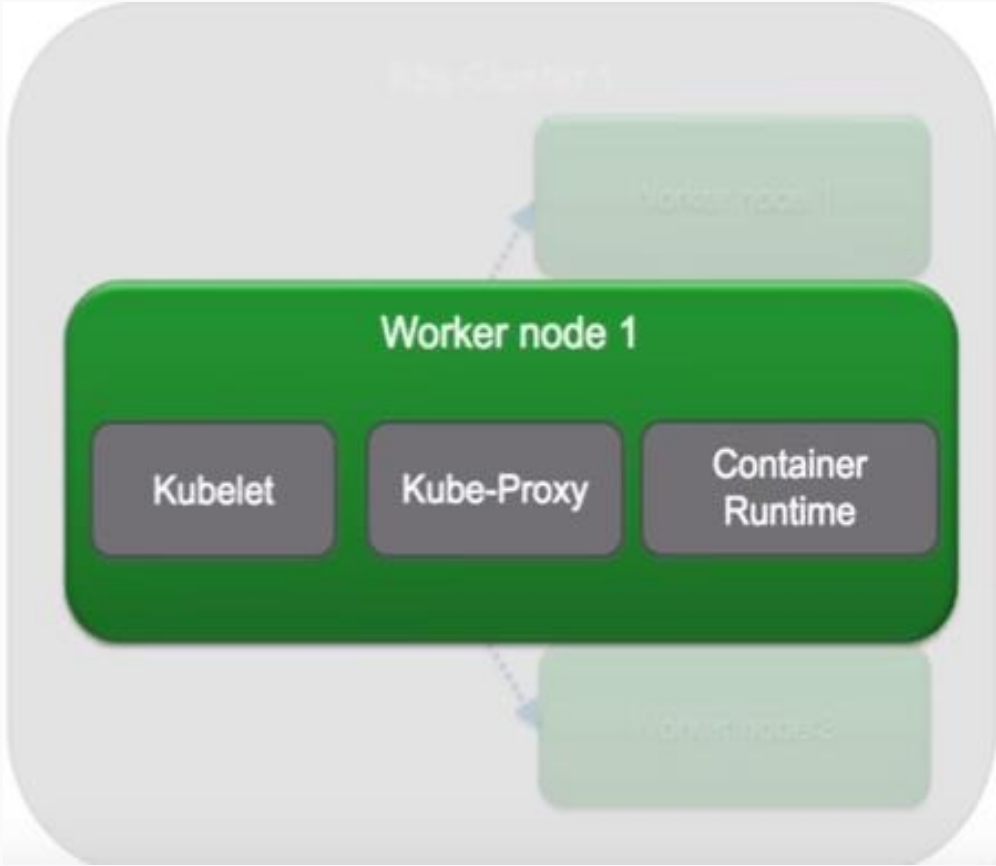
Kubernetes架构 - Master

- K8s Master
  - API Server
  - Scheduler
  - Controller Manager
  - Dashboard(addons)



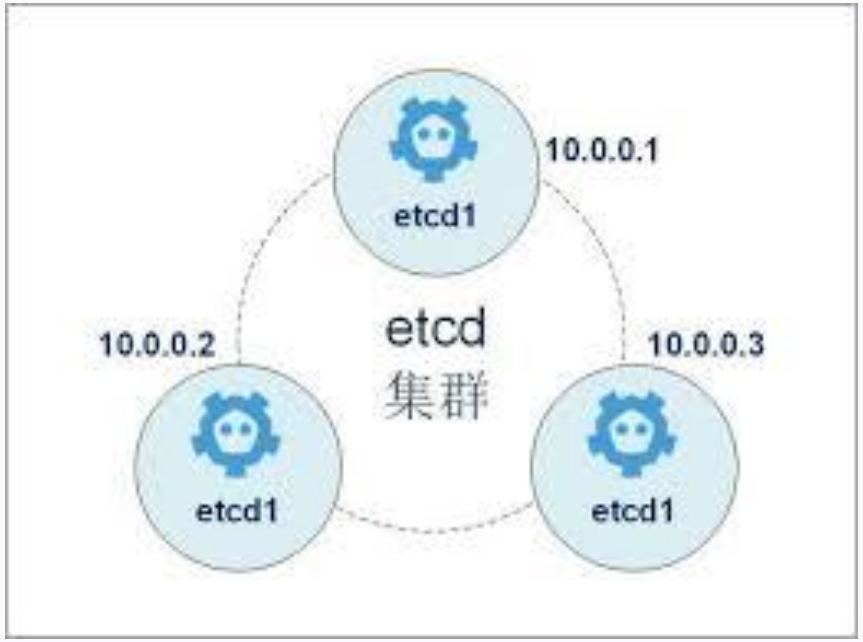
Kubernetes架构 - Node

- Kubelet
  - 运行在Node节点上的Agent
  - 处理Master节点下发到本节点的任务，管理Pod和其中的容器
  - 定期向Master汇报节点资源使用情况
- Kube-Proxy
  - 运行在Node节点上的Agent
  - 实现Service的抽象，为一组Pod抽象的服务（Service）提供统一接口并提供负载均衡功能
- Container Runtime
  - Docker
  - Rkt

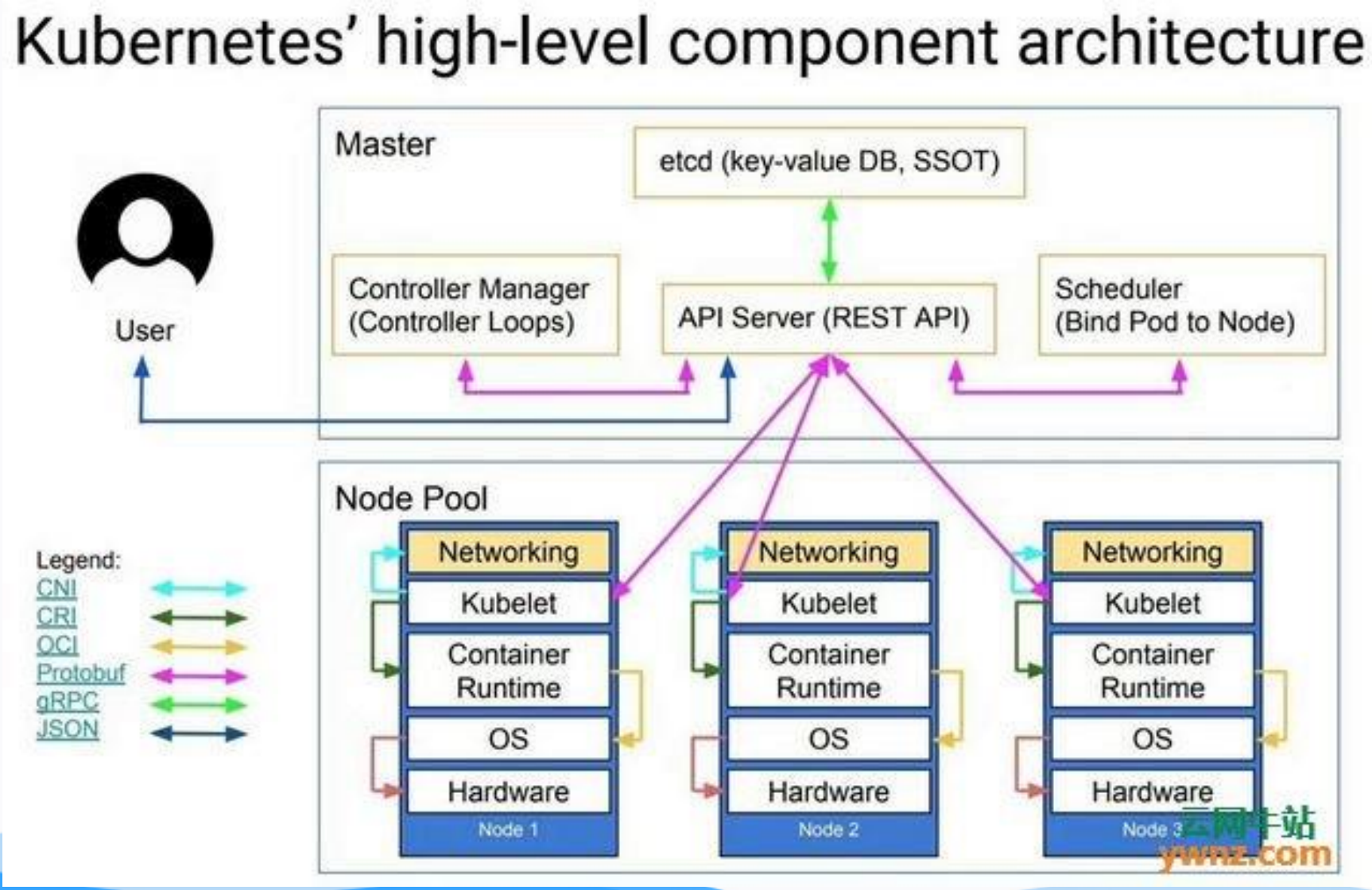


Kubernetes架构 - Etcd

- Etcd
  - CoreOS开发并开源，基于Raft协议的分布式的一致性KV存储
  - 类似于Zookeeper
  - 在K8s中用作分布式KV存储系统
  - 用于保存集群所有的网络配置和对象的状态信息

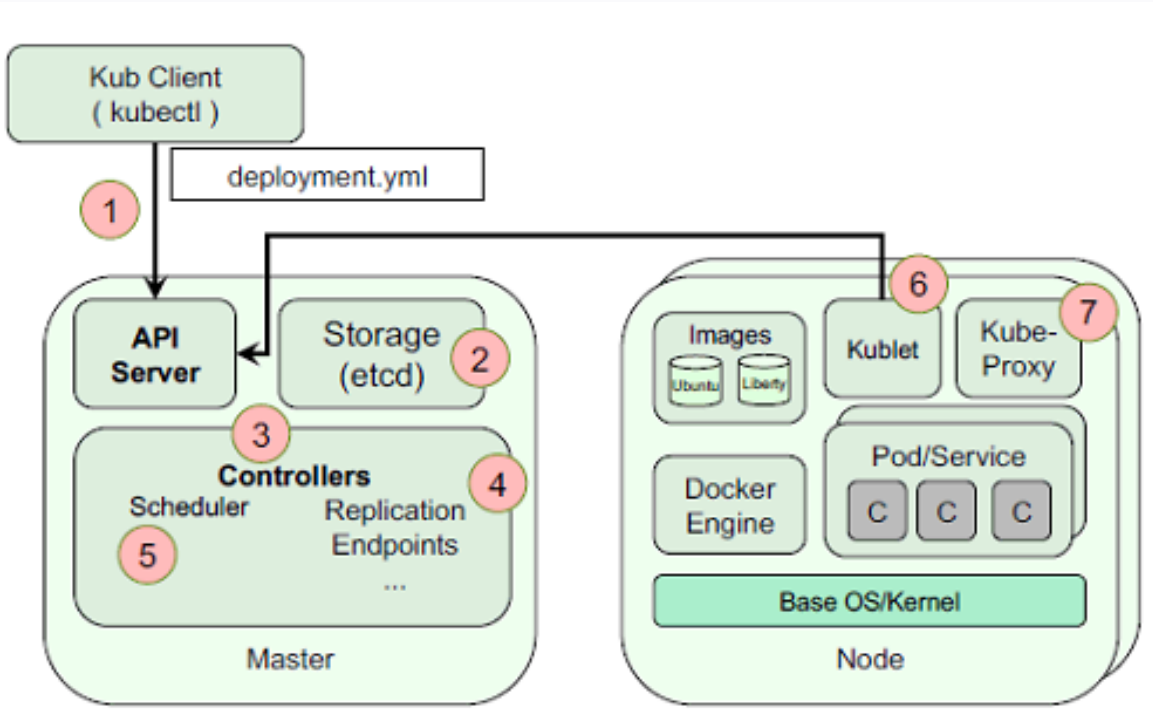


Kubernetes架构



Kubernetes调用流程

1. 用户通过”kubectl”来进行操作，例如部署新的应用
2. API Server接收到请求，并将其存储到Etcd
3. Watcher和Controllers检测到资源状态的变化，并进行操作
4. ReplicaSet watcher/controller检测到新的app，创建新的pod达到期望的实例个数
5. Scheduler将新的Pod分配到Kubelet
6. Kubelet检测到Pods，并通过容器运行时部署它们
7. Kubeporxy管理Pod的网络，包括服务发现、负载均衡



目录

Kubernetes安装部署

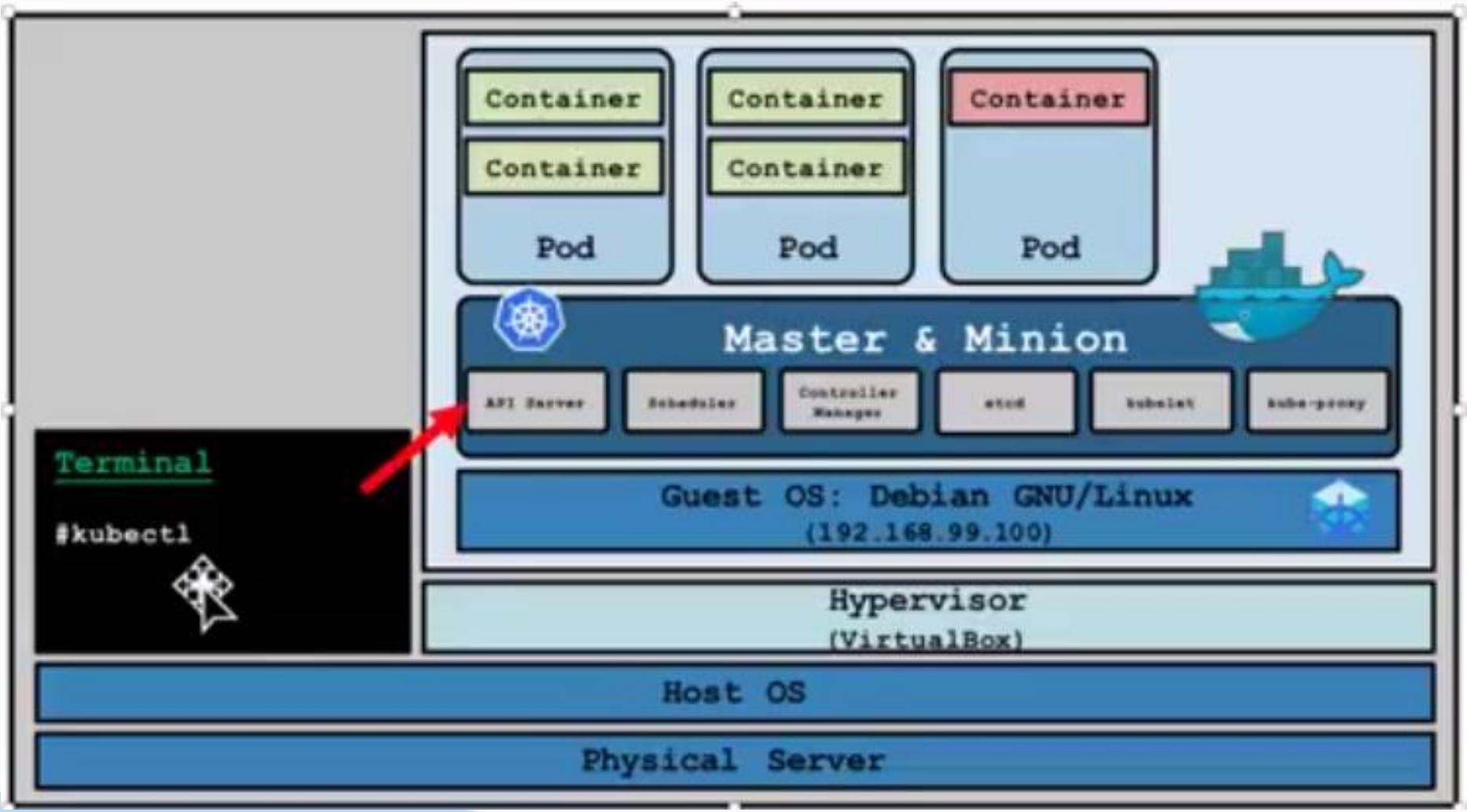
- 本地开发环境
  - Minikube
    - https://kubernetes.io/docs/getting-started-guides/minikube/
  - Vagrant + Virtualbox

- Kubernetes集群
  - Kubeadm (https://kubernetes.io/docs/setup/independent/install-kubeadm/)
  - Kops: GCE，阿里，腾讯云等云的服务
  - CoreOS Tectonic
  - 二进制部署（https://github.com/rootsongjc/kubernetes-vagrant-centos-cluster）

MiniKube介绍

- 为了方便大家开发和体验Kubernetes， Kubernetes社区提供了可以在本地部署的Minikube
- Kubernetes 本地安装 – Minikube 为例
  - 1. 安装虚拟机(Virtualbox,xhyve, VMWare)
  - 2. 安装kubectl命令行工具
  - 3. 安装minikube命令行工具
  - 4. 运行minikube start命令
  - 5. 完成

MiniKube架构


Container
Container
Pod
Container
Container
Pod
Container
Pod
Master & Minion
API Server
Scheduler
Controller Manager
etcd
kubelet
kube-proxy
Guest OS: Debian GNU/Linux
(192.168.99.100)
Terminal
#kubectl
Hypervisor
(VirtualBox)
Host OS
Physical Server

Kubernetes命令行

- Kubectl用于运行Kubernetes集群命令的管理工具
- kubectl命令行语法
- kubectl [command] [TYPE]　[NAME]　[flags]
  - Command: 操作 create , get , describe , delete
  - TYPE: 指定操作的资源类型
  - NAME: 指定资源名称,如忽略则默认命名空间下所有同类资源
  - flags: 命令行选型,如覆盖默认服务器地址,端口,输出样式等

Kubernetes命令行 – 集群状态

命令	解释
kubectl cluster-info	查看集群信息
kubectl version	显示kubectl命令行及kube服务端的版本
kubectl api-version	显示支持的API版本集合
kubectl config view	显示当前kubectl配置
kubectl get node	查看集群中节点

目录

Kubernetes基础– 资源对象

- Kubernetes中所有的资源实体都可以表示为资源对象
  - 资源对象通过Yaml描述
  - 资源实例化后称为对象
  - 通过API或kubectl来管理Kubenetes资源对象

- Kubernetes中资源通过Yaml格式文件来描述资源，称为资源清单（Manifest）

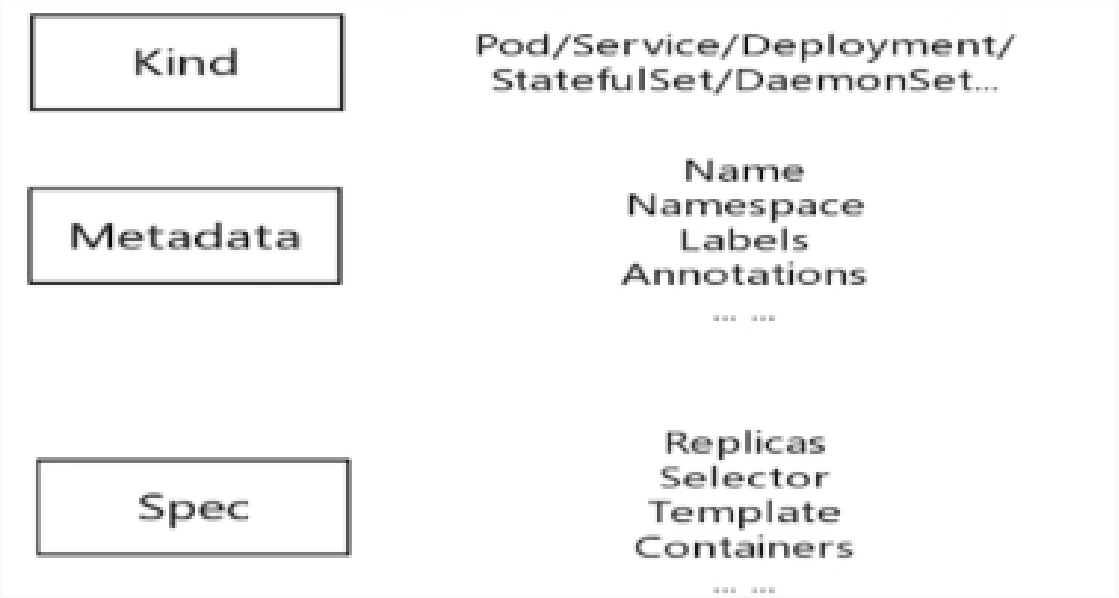
Kubernetes基础– 资源类型

- Kubernetes中主要的资源类型

类别	名称
工作负载型资源对象	Pod、Replicaset、ReplicationController、Deployments、StatefulSets、Daemonset 、Job、CronJob
服务发现及负载均衡	Service、Ingress
配置与存储	Volume、Persistent Volume、CSl 、 Configmap、Secret
集群资源	Namespace、Node、Role、ClusterRole、RoleBinding、ClusterRoleBinding
元数据资源	HPA、PodTemplate、LimitRange

Kubernetes基础– 资源对象

- Kubernetes对象模型：




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

Pod.yaml

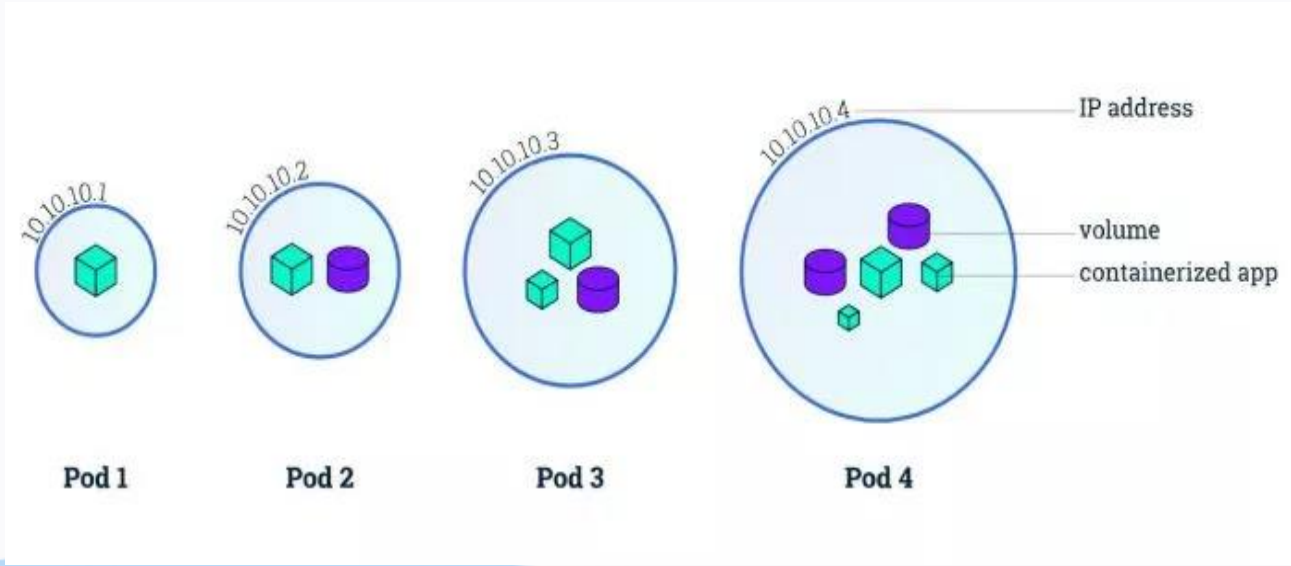
Kubernetes基础– 资源对象

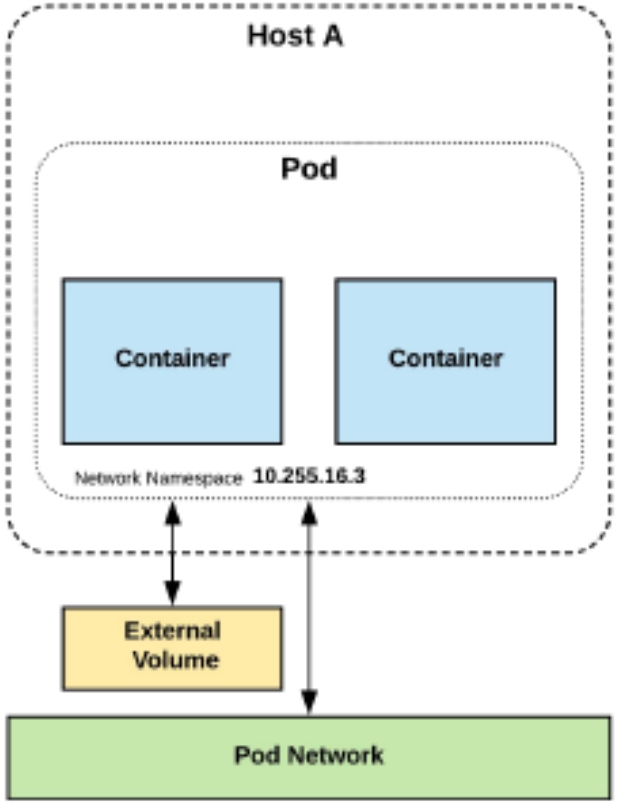
命令式与声明式

命令	解释	
kubectl create -f <res.yaml>	按照yaml文件创建资源对象	命令式
kubectl apply -f <res.yaml>	按照yaml文件创建资源对象	声明式
kubectl get <type> <name>	查看某种类型资源	
kubectl describe <type> <name>	检查某特定资源实例	
kubectl explain <type> <name>	显示资源定义信息	

Kubernetes核心概念 - Pods

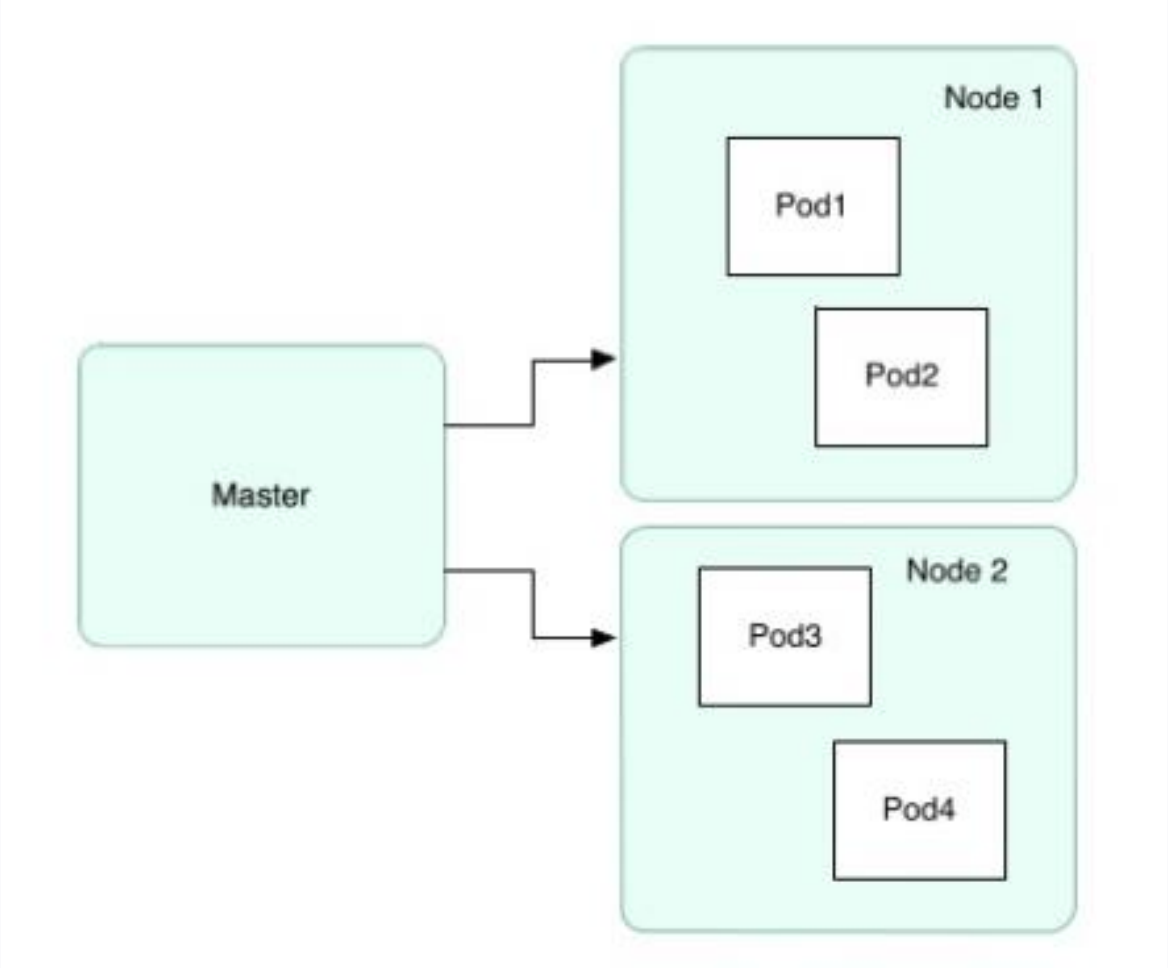
- Pods
  - Pod 是一组紧密关联的容器集合
  - 共享 PID、IPC、Network 和 UTS namespace
  - Kubernetes 调度的基本单位





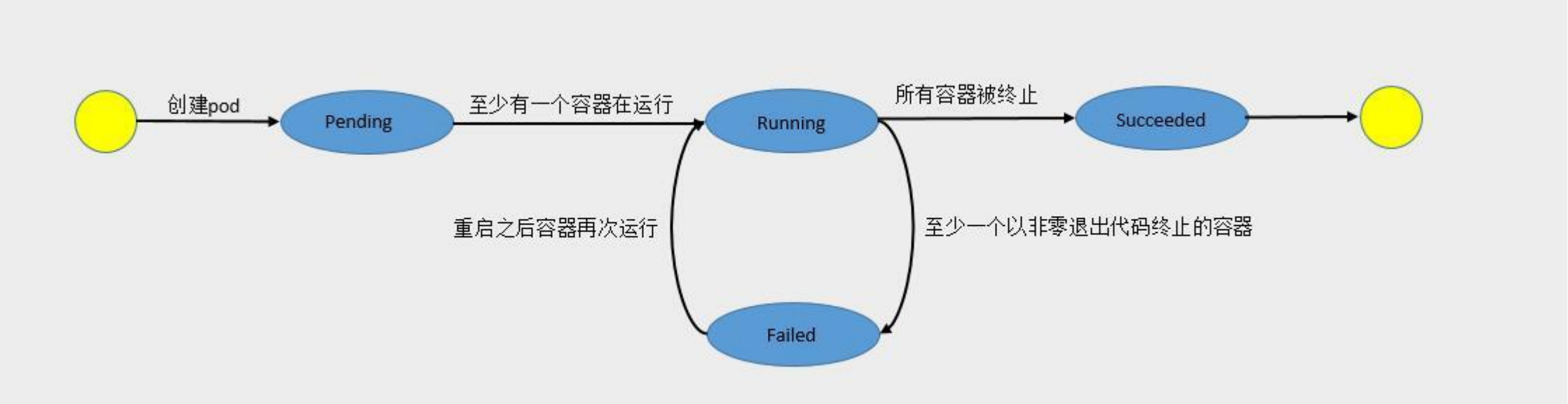
Kubernetes核心概念 - Pods

- Pods
  - 将被调度到Nodes上
  - 一个非持久的实体（自主式 vs 控制器管理的）



Kubernetes核心概念 - Pods

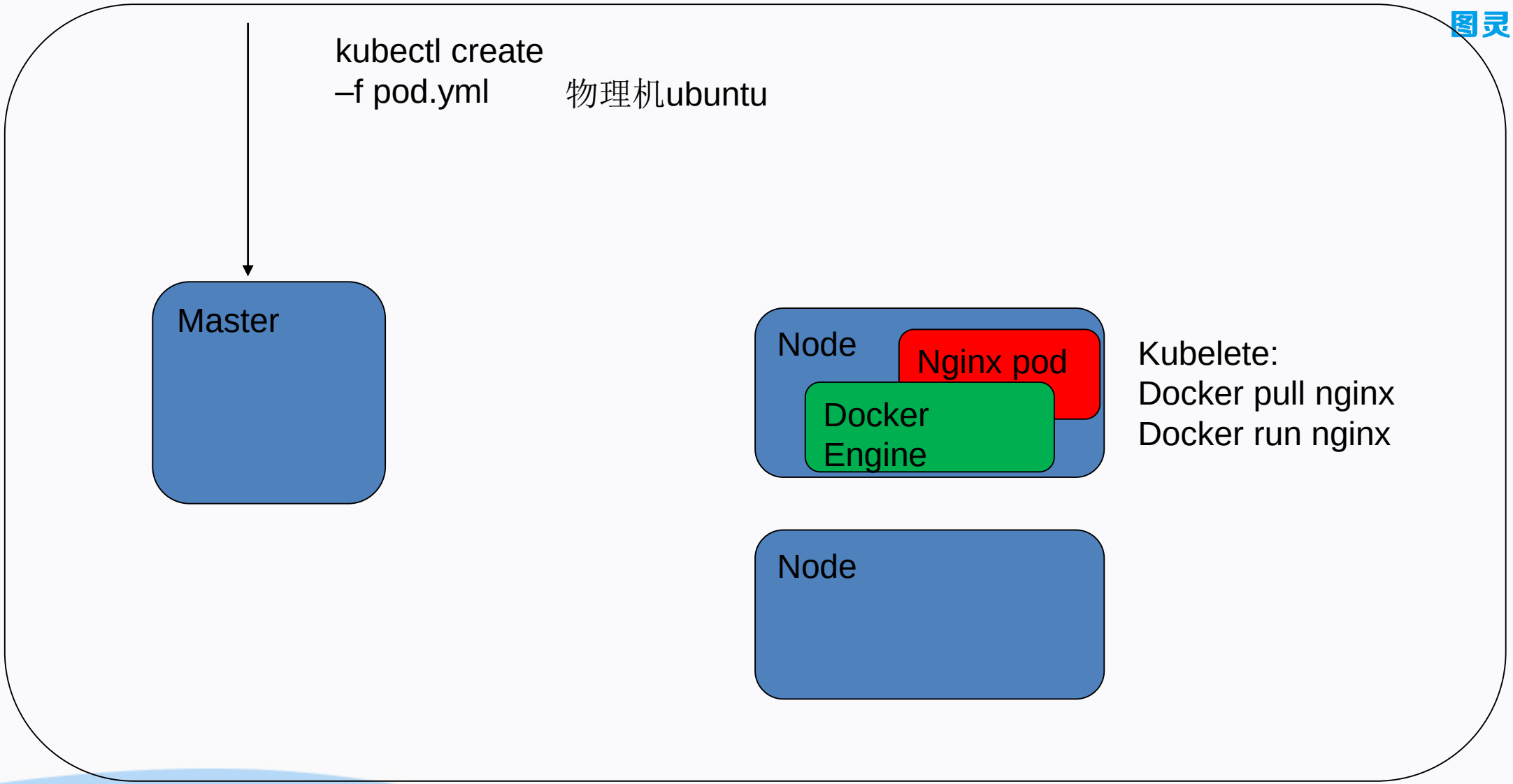
- Pods – 生命周期

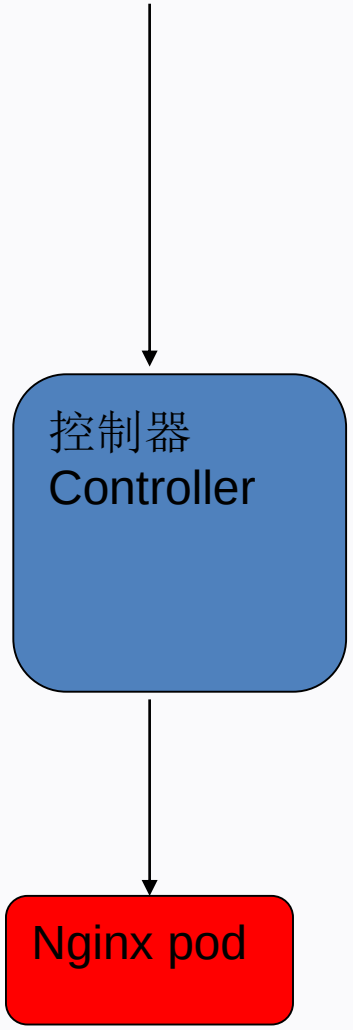


Kubectl – Pods Demo

- kubectl create – f pod.yml
- Kubectl get pods
- Kubectl describe pods nginx..
- Kubectl logs nginx
- kubectl exec -it nginx sh

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: pod-example
spec:
  containers:
  - name: web
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```





ReplicaSet: 保证
这个pod副本数量

Kubernetes核心概念 – Labels和Selectors

- Labels
  - 可以标记任何对象
  - 具体表示形式为：Key-Value对

- Selectors
  - 根据label来选择对象
  - 支持两种方式：
    - equality-based
      - = Prod
      - ReleasEnvironmente!= test
    - set-based
      - Environment in (Prod, Dev)
      - Release not in (Test, Canary)

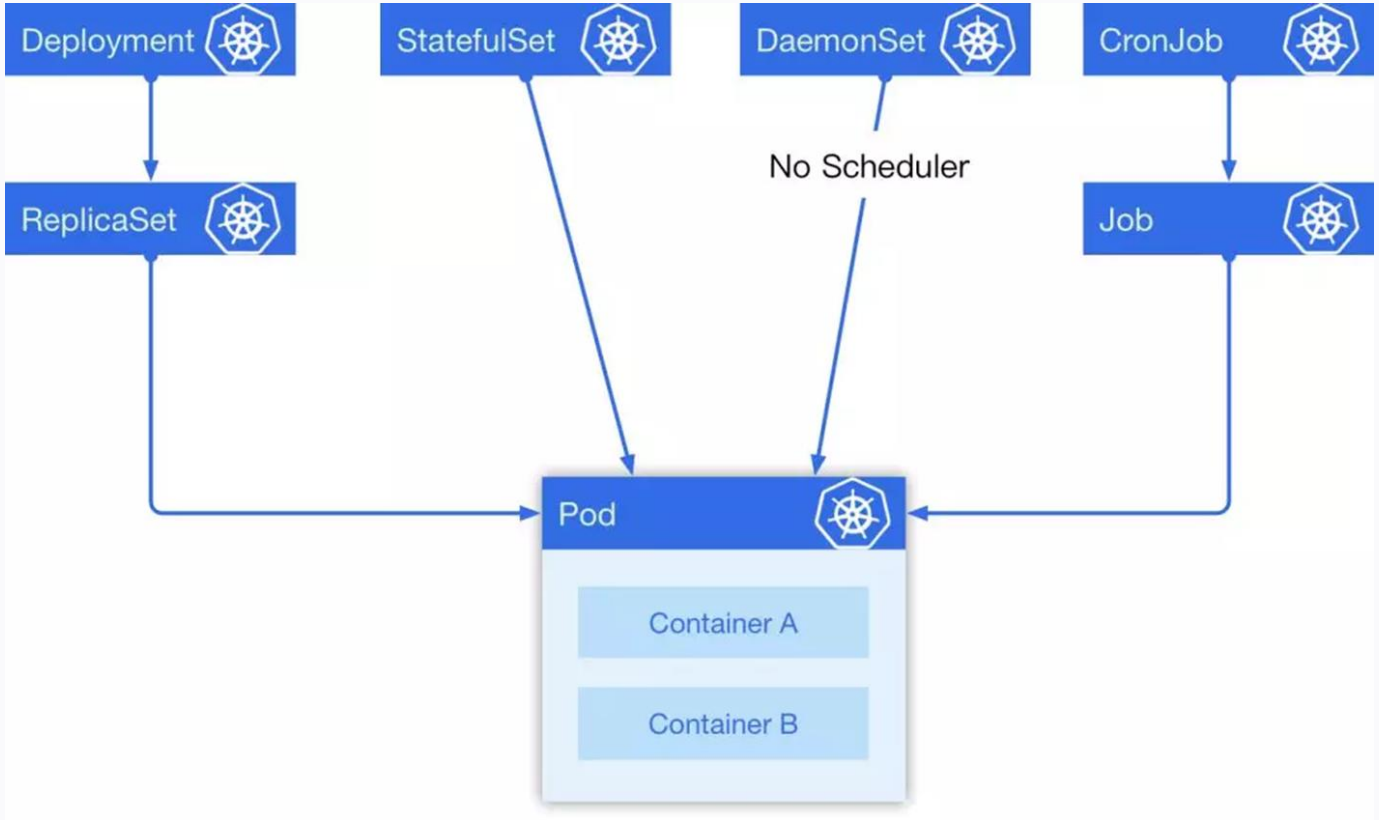
Kubernetes核心概念 – Labels和Selectors

- kubectl create – f label.yaml
- kubectl get pods -l app=nginx
- Kubectl describe pod pod-label-example
- kubectl get pods -l 'env in (prod, qa)'

```
apiVersion: v1
kind: Pod
metadata:
name: pod-label-example
labels:
app: nginx
env: prod
spec:
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
~
```

Kubernetes核心概念 - Controller

- Controller对象
  - 使用Pod模板来创建实际需要的pod，并保证其按照某种期望状态运行（如副本数量等）
  - 可以创建和管理多个Pod，提供副本管理、滚动升级和集群级别的自愈能力
  - 内置了多种Controller，如右图所示



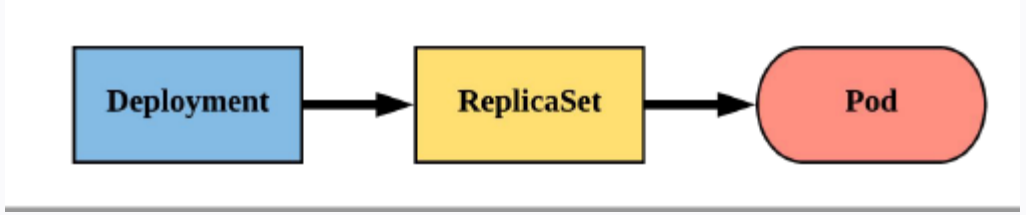
Kubernetes核心概念 – ReplicaSet

- ReplicationController
  - 确保Pod以指定的副本数运行，即如果有容器异常退出，会自动创建新的 Pod 来替代
  - 仅支持equality-based selector

- ReplicationSet
  - ReplicationController的升级，支持 set-based selector，其他无本质区别
  - 可单独使用，但是更多通过Deployment来管理

Kubernetes核心概念 – Deployments

- Deployment
  - 提供了一种声明式的方法来通过ReplicationSet管理Pods
  - 支持Pod的RollingUpdate，并自动管理其背后的ReplicationSet
  - 支持roll back到之前的revision



Kubernetes核心概念 – Deployments Demo

- kubectl create – f nginx-deployment.yaml
- kubectl get deployment
- kubectl describe deployment nginx-deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-demo
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
     matchLabels:
        app: nginx
  strategy:
     type: RollingUpdate
     rollingUpdate:
        maxSurge: 1
        maxUnavailable: 0
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Kubernetes核心概念 – Deployments Demo

- kubectl apply – f nginx-deployment-v2.yaml
- Kubectl get deployment
- Kubectl describe deployment nginx-deploymen
- kubectl rollout status deployment/nginx-deployment-demo
- kubectl rollout **history** deployment nginx-deployment-demo
- kubectl rollout **history** deployment nginx-deployment-demo --revision=3
- kubectl rollout undo deployment nginx-deployment – demo --to-revision=1
- kubectl scale deployment nginx-deployment-demo --replicas=10

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-demo
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.8
        ports:
        - containerPort: 80
~
```

谢谢！