# **Kubernetes简介 课程简介**

K8s概念和架构

从零搭建K8s集群

* + 基于客户端工具kubeadm搭建（简单，最多半小时）
  + 基于二进制包方式（能看到内部的架构）

K8s核心概念

* + Pod：K8s管理的最小单位级，是所有业务类型的基础
  + Controller：控制器，有状态，无状态，一次任务，定时任务，守护进程
  + Service Ingress：对外暴露端口
  + RBAC：安全机制，权限模型
  + Helm：下载机制
  + 持久化存储

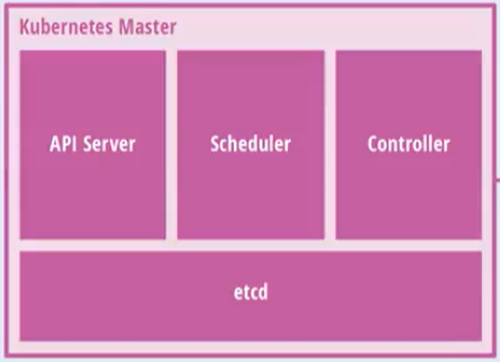
搭建集群监控平台系统

从零搭建高可用K8s集群

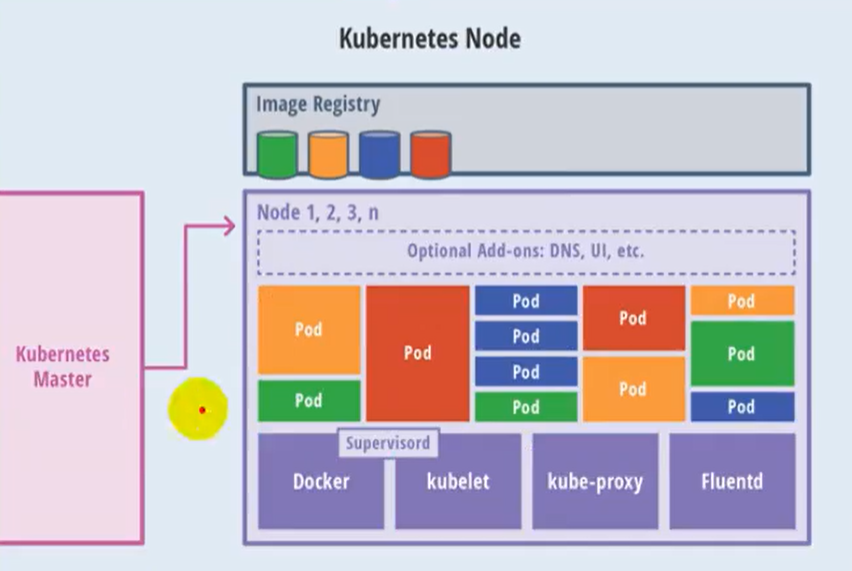
在集群环境部署项目

**架构细节**

master节点架构图



Node节点架构图



k8s 集群控制节点，对集群进行调度管理，接受集群外用户去集群操作请求；

****master****：主控节点

* + API Server：集群统一入口，以restful风格进行操作，同时交给etcd存储
    - 提供认证、授权、访问控制、API注册和发现等机制
  + scheduler：节点的调度，选择node节点应用部署
  + controller-manager：处理集群中常规后台任务，一个资源对应一个控制器
  + etcd：存储系统，用于保存集群中的相关数据

****Work node****：工作节点

* + Kubelet：master派到node节点代表，管理本机容器
    - 一个集群中每个节点上运行的代理，它保证容器都运行在Pod中
    - 负责维护容器的生命周期，同时也负责Volume(CSI) 和 网络(CNI)的管理
  + kube-proxy：提供网络代理，负载均衡等操作

容器运行环境【****Container Runtime****】

* + 容器运行环境是负责运行容器的软件
  + Kubernetes支持多个容器运行环境：Docker、containerd、cri-o、rktlet以及任何实现Kubernetes CRI (容器运行环境接口) 的软件。

fluentd：是一个守护进程，它有助于提升 集群层面日志

## **K8S核心概念**

### **Pod**

* Pod是K8s中最小的单元
* 一组容器的集合
* 共享网络【一个Pod中的所有容器共享同一网络】
* 生命周期是短暂的（服务器重启后，就找不到了）

### **Volume**

* 声明在Pod容器中可访问的文件目录
* 可以被挂载到Pod中一个或多个容器指定路径下
* 支持多种后端存储抽象【本地存储、分布式存储、云存储】

### **Controller**

* 确保预期的pod副本数量【ReplicaSet】
* 无状态应用部署【Deployment】
  + 无状态就是指，不需要依赖于网络或者ip
* 有状态应用部署【StatefulSet】
  + 有状态需要特定的条件
* 确保所有的node运行同一个pod 【DaemonSet】
* 一次性任务和定时任务【Job和CronJob】

### **Deployment**

* 定义一组Pod副本数目，版本等
* 通过控制器【Controller】维持Pod数目【自动回复失败的Pod】
* 通过控制器以指定的策略控制版本【滚动升级、回滚等】

**Service**

* 定义一组pod的访问规则
* Pod的负载均衡，提供一个或多个Pod的稳定访问地址
* 支持多种方式【ClusterIP、NodePort、LoadBalancer】

### **Label**

label：标签，用于对象资源查询，筛选



### **Namespace**

命名空间，逻辑隔离

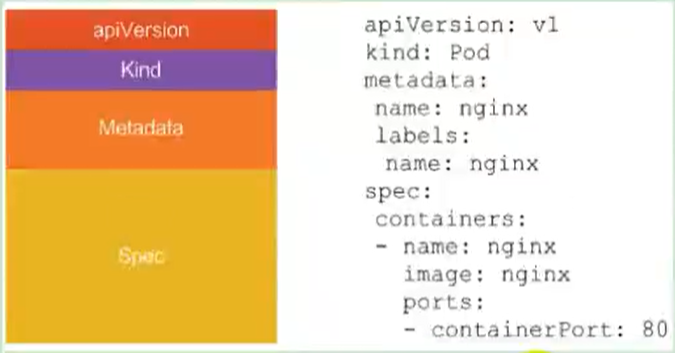
* 一个集群内部的逻辑隔离机制【鉴权、资源】
* 每个资源都属于一个namespace
* 同一个namespace所有资源不能重复
* 不同namespace可以资源名重复

### **API**

我们通过Kubernetes的API来操作整个集群

同时我们可以通过 kubectl 、ui、curl 最终发送 http + json/yaml 方式的请求给API Server，然后控制整个K8S集群，K8S中所有的资源对象都可以采用 yaml 或 json 格式的文件定义或描述

如下：使用yaml部署一个nginx的pod



## **完整流程**



* 通过Kubectl提交一个创建RC（Replication Controller）的请求，该请求通过APlserver写入etcd
* 此时Controller Manager通过API Server的监听资源变化的接口监听到此RC事件
* 分析之后，发现当前集群中还没有它所对应的Pod实例
* 于是根据RC里的Pod模板定义一个生成Pod对象，通过APIServer写入etcd
* 此事件被Scheduler发现，它立即执行执行一个复杂的调度流程，为这个新的Pod选定一个落户的Node，然后通过API Server讲这一结果写入etcd中
* 目标Node上运行的Kubelet进程通过APiserver监测到这个"新生的Pod.并按照它的定义，启动该Pod并任劳任怨地负责它的下半生，直到Pod的生命结束
* 随后，我们通过Kubectl提交一个新的映射到该Pod的Service的创建请求
* ControllerManager通过Label标签查询到关联的Pod实例，然后生成Service的Endpoints信息，并通过APIServer写入到etod中，
* 接下来，所有Node上运行的Proxy进程通过APIServer查询并监听Service对象与其对应的Endponts信息，建立一个软件方式的负载均衡器来实现Service访问到后端Pod的流量转发功能