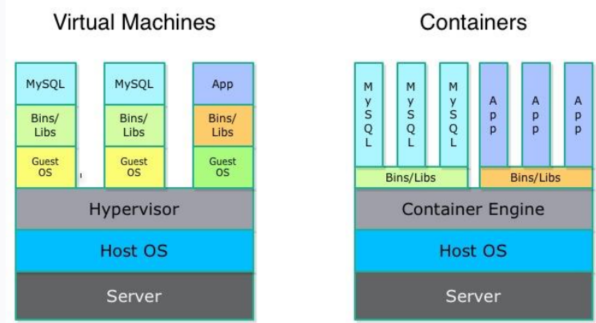
# 1、kubernetes

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）

## 1.1 kubernetes基础

### 1.1 kubernetes使用的容器引擎

**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）



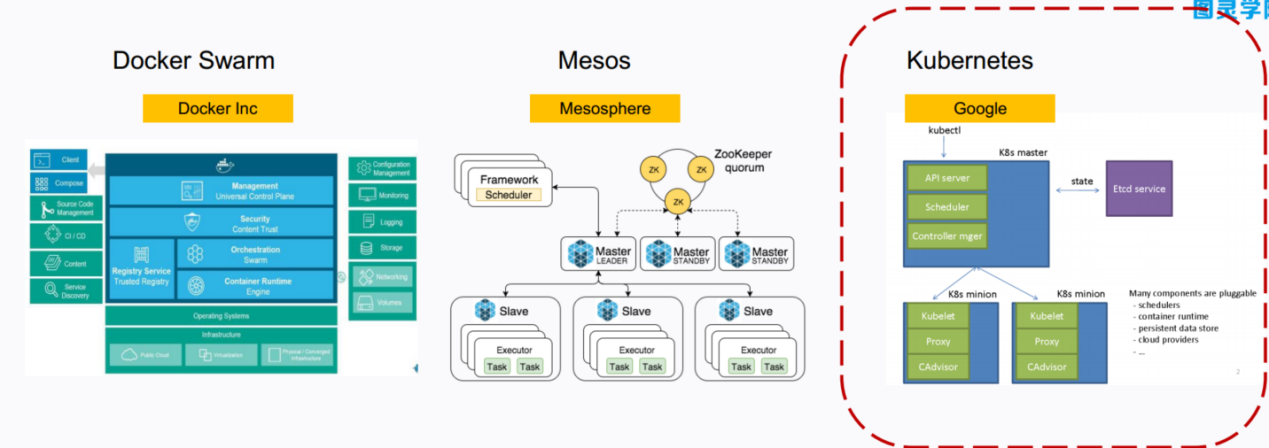
**容器解决了应用打包、部署、运行的问题**  
– 一次构建，随处运行（Build,Ship and Run Any App,Anywhere )

**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…）

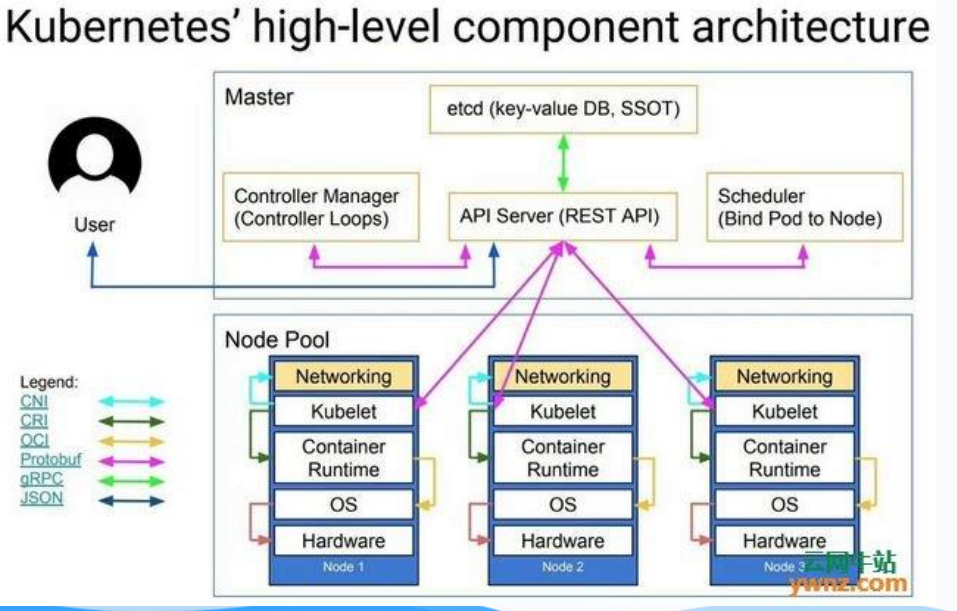
容器编排的几种实现方式



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

### 1.2 kubernetes架构

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



#### 1、API Server

该进程只运行在master主节点

1、提供了集群管理的REST API接口(包括认证授权、数据校验以及集群状态变更)；

2、提供其他模块之间的数据交互和通信的枢纽（其他模块通过API Server查询或修改数据，只有API Server才直接操作etcd）;

3、是资源配额控制的入口；

4、拥有完备的集群安全机制.

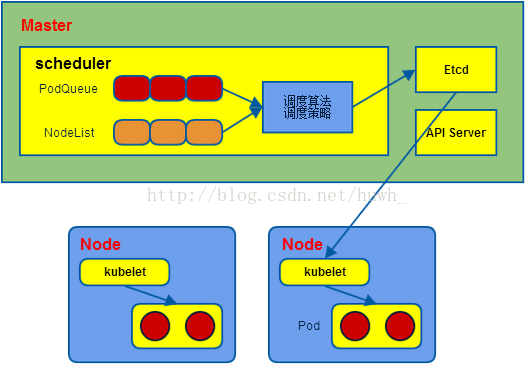
#### 2、Scheduler调度器

负责接收Controller Manager创建的新的Pod，为其选择一个合适的Node。

启下：Node上的kubelet接管Pod的生命周期。

通过调度算法为待调度Pod列表的每个Pod从Node列表中选择一个最适合的Node，并将信息写入etcd中

kubelet通过API Server监听到kubernetes Scheduler产生的Pod绑定信息，然后获取对应的Pod清单，下载Image，并启动容器。



调度流程：

1、预选调度过程，即遍历所有目标Node,筛选出符合要求的候选节点，kubernetes内置了多种预选策略（xxx Predicates）供用户选择

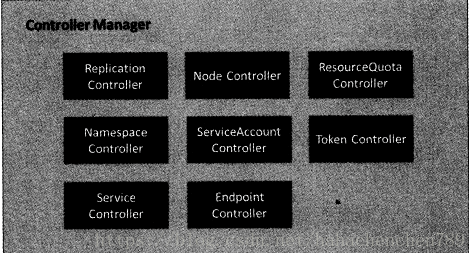
2、确定最优节点，在第一步的基础上采用优选策略（xxx Priority）计算出每个候选节点的积分，取最高积分。

调度流程通过插件式加载的“调度算法提供者”（AlgorithmProvider）具体实现，一个调度算法提供者就是包括一组预选策略与一组优选策略的结构体。

#### 3、controller manager

controller manager作为集群内部的管理控制中心，负责集群内的Node，Pod副本，服务端点（endpoint），命名空间（namespace）等的管理，当某个Node意外宕机，CM会及时发现此故障并执行自动化修复流程，确保集群始终处于预期的工作状态。

CM内部包含了RC，node controller，namespace controller等多个控制器，每种控制器都负责一种具体的控制流程，而CM正是这些controller的核心管理者。



#### 4、Dashboard(addons)

图形化界面操作k8s

#### 5、node具体容器部署环境

Node主要由Kubelet、Kube-Proxy、Container Runtime三部分组成



##### 1、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

##### 2、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

##### 3、Container Runtime

具体容器

– Docker  
– Rkt

#### 6、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（ZAB协议）  
– 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

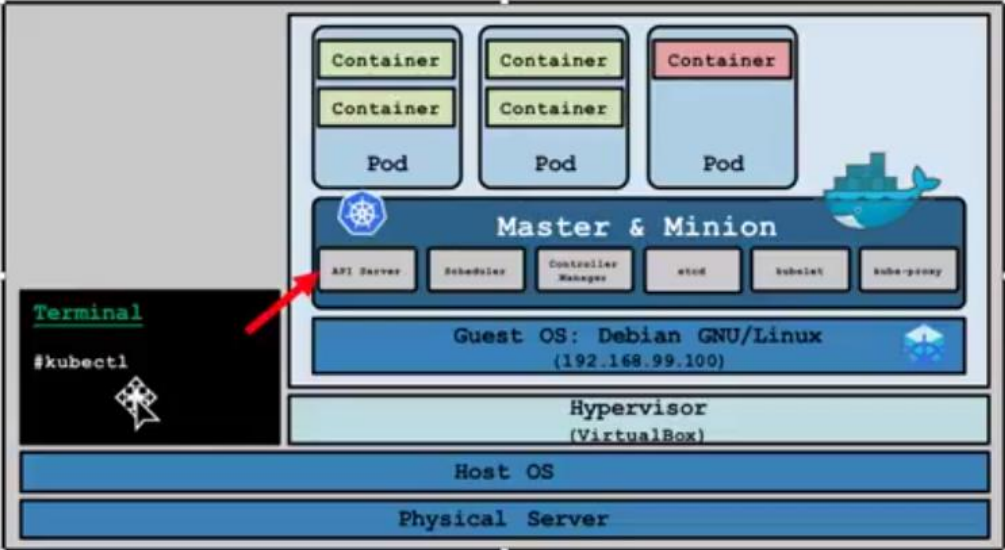
### 1.3 kubernetes安装

• 本地开发环境  
– Minikube  
• https://kubernetes.io/docs/getting-started-guides/minikube/  
– Vagrant + Virtualbox可以在github上找到现成的集群实例，搜索Vagrant即可

• Kubernetes集群  
– Kubeadm (https://kubernetes.io/docs/setup/independent/installkubeadm/)  
– Kops: GCE，阿里，腾讯云等云的服务  
– CoreOS Tectonic  
– 二进制部署（<https://github.com/rootsongjc/kubernetes-vagrantcentos-cluster>）

• 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



## 1.2 kubernetes的基础资源对象

### 2.1 kubectl命令行语法

• 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

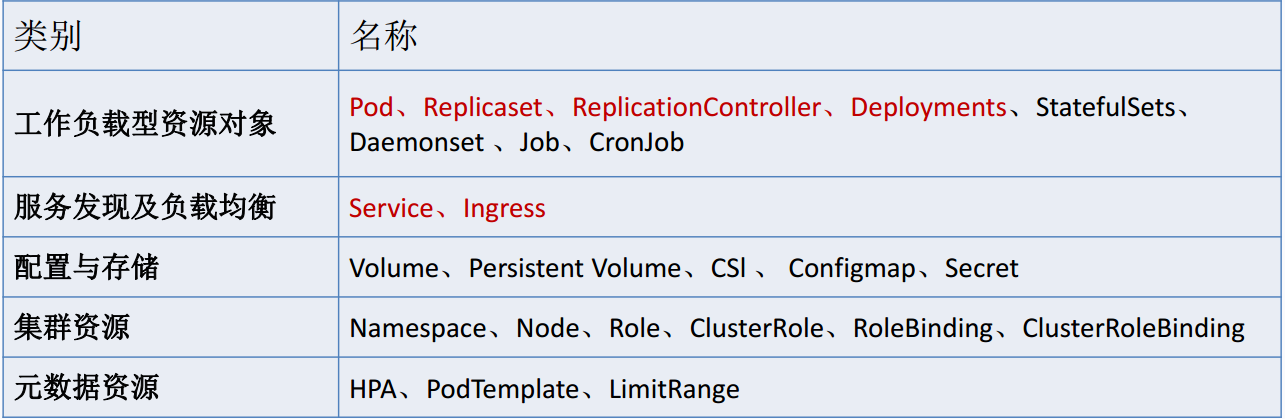




### 2.2 kubernetes基础对象类型

• 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主要对象类型**



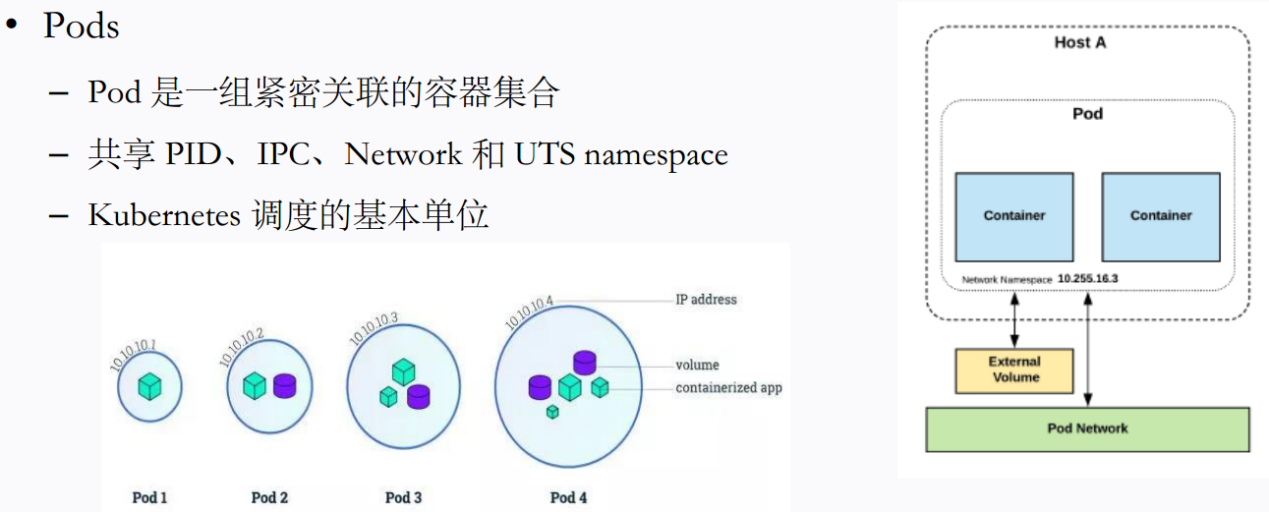
**Kubernetes对象模型**





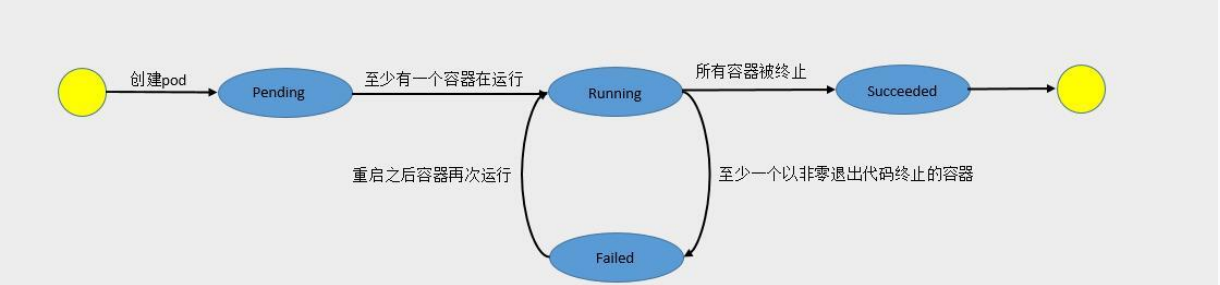
### 2.3 kubernetes具体对象

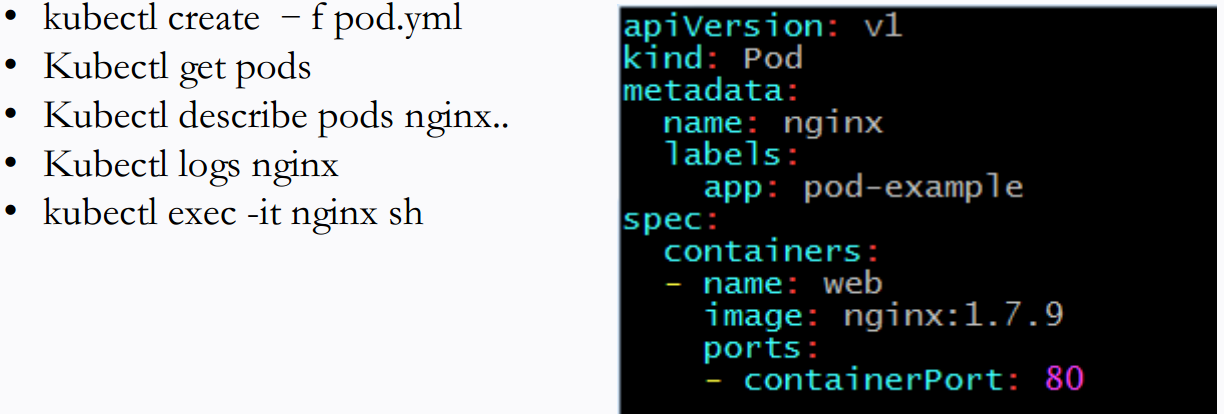
#### 1、pod



Pod以非持久形式存在，当发生异常后会如同垃圾对象被回收，如果自主式的pod异常后彻底消息，如果通过控制器管理则会重新创建一个相同的pod

Pod的生命周期

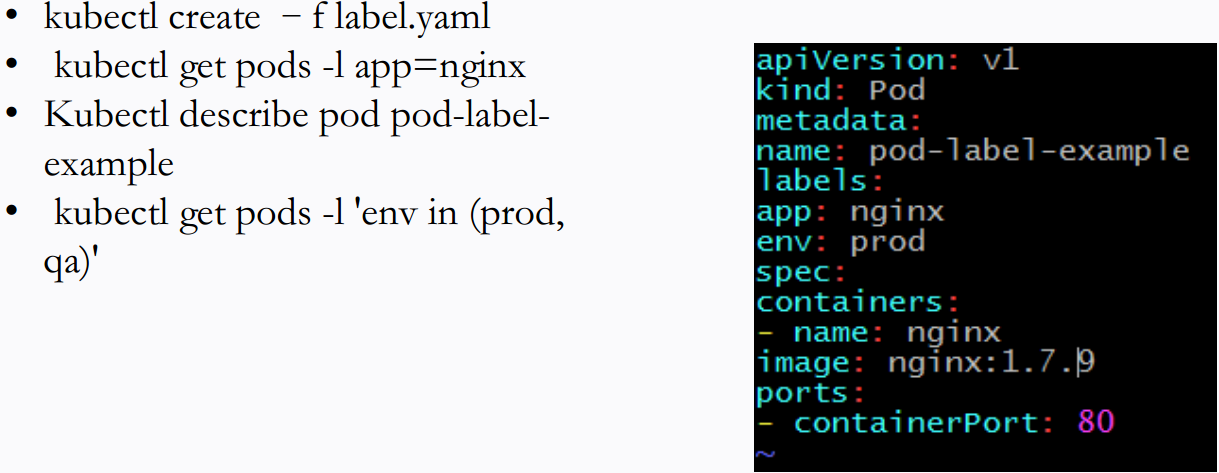




**Selectors与 Labels**

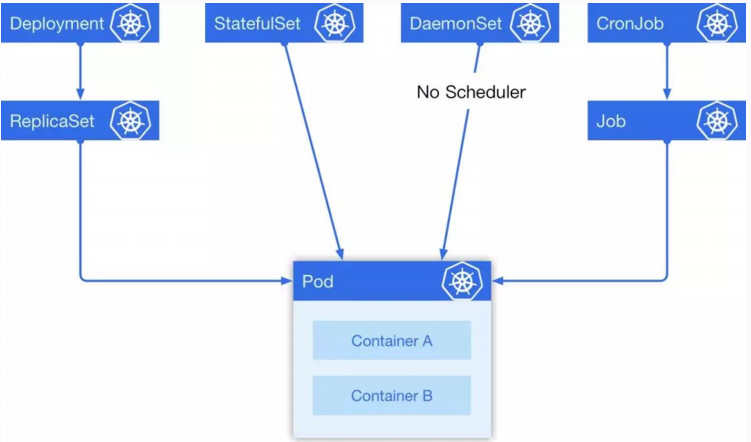
• Labels  
– 可以标记任何对象  
– 具体表示形式为： Key-Value对  
• Selectors  
– 根据label来选择对象  
– 支持两种方式：  
• equality-based  
– = Prod  
– ReleasEnvironmente!= test  
• set-based  
– Environment in (Prod, Dev)  
– Release not in (Test, Canary)

使用：



#### 2、controller对象

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



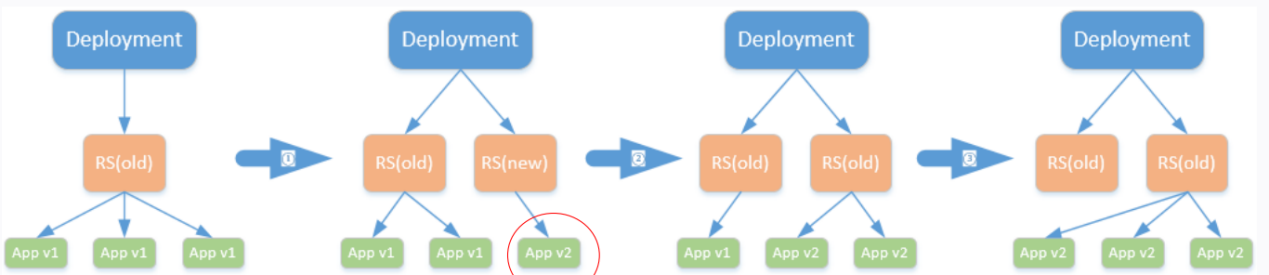
• ReplicationController  
– 确保Pod以指定的副本数运行，即如果有容器异常退出，会自动创建新的 Pod 来替代  
– 仅支持equality-based selector  
• ReplicationSet  
– ReplicationController的升级，支持 set-based selector，其他无本质区别  
– 可单独使用，但是更多通过  
Deployment来管理

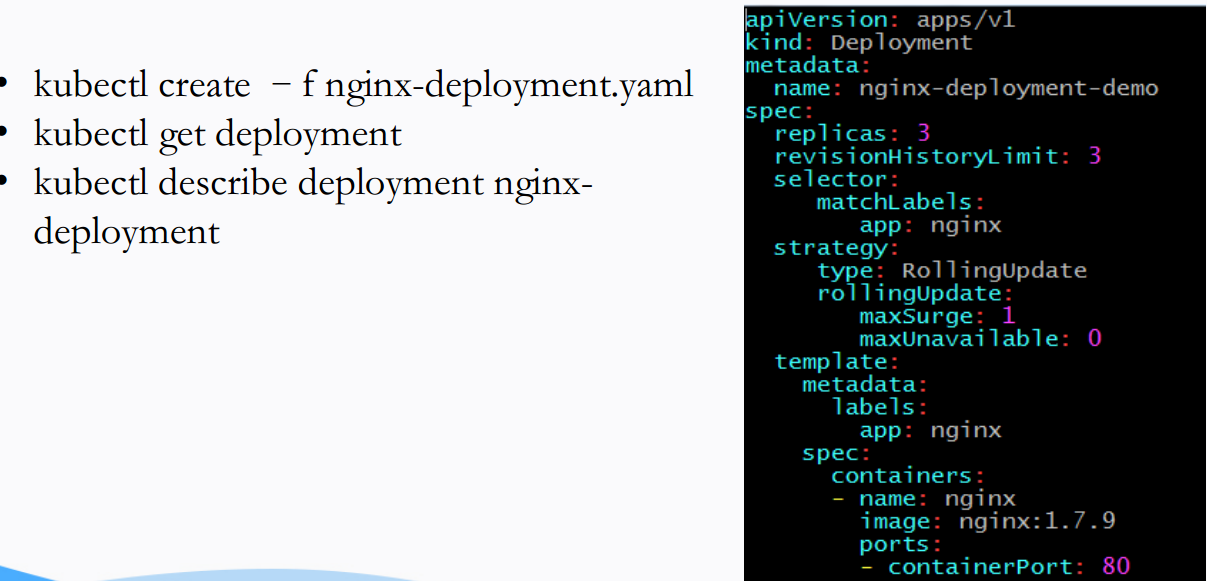
#### 3、Deployment

ReplicaSet  
– 确保Pod副本的数量  
– 支持自动扩容和收缩（scale)  
– 不能支持服务的滚动部署

– 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

• 红绿部署  
• 滚动部署（默认）  
• 灰度部署(canary，金丝雀部署)



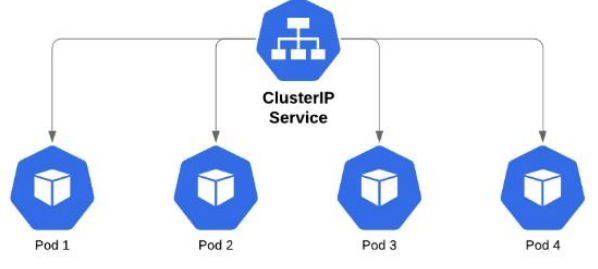


• kubectl apply – f nginx-deployment.yml --record  
• kubectl rollout status deployment/nginxdeployment-demo  
• kubectl get deployment  
• Kubectl get rs  
• kubectl describe deployment nginx-deployment  
• kubectl get pod --show-labels

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

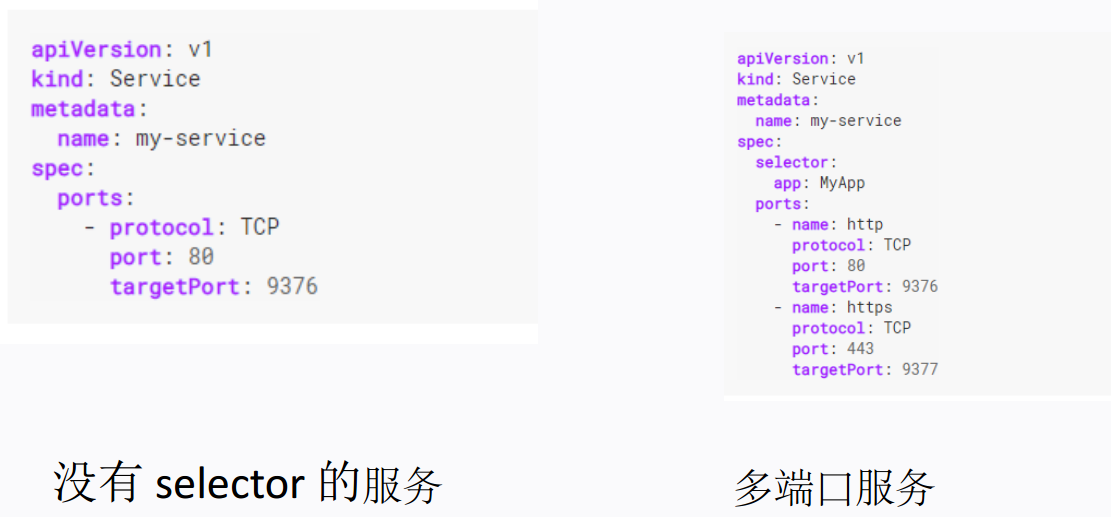
#### 4、Service

– Pod之上的一个抽象层  
– Service 会被分配一个VIP(ClusterIP)，该虚拟id与集群内的pod不在一个网段内，只能用于集群内部使用，集群内部的不同service之间的pod相互访问时使用，并在它销毁之前保持该 VIP 地址保持不变  
– 通过对它的访问，以代理的方式负载到对应的 Pod 上



• kubectl apply –f nginx-service.yml  
• port：虚拟 ip要绑定的 port，每个service 会创建出来一个虚拟 ip，通过访问 vip:port 就能获取服务的内容  
• targetPort：pod 中暴露出来的 port，这是运行的容器中具体暴露出来的端口  
• 默认的 service 类型是 ClusterIP  
– 只能从集群内部访问这个 IP，不能直接从集群外部访问服务





**Service的服务发现**

• 环境变量 - POD  
– Pod 运行在 Node 上， kubelet 会为每个活跃的 Service 添加一组环境变量。{SVCNAME}\_SERVICE\_HOST 和 {SVCNAME}\_SERVICE\_PORT 变量，这里 Service 的名称需大写，横线被转换成下划线。 Nignx-service  
– 服务创建后的新建的Pod里才有

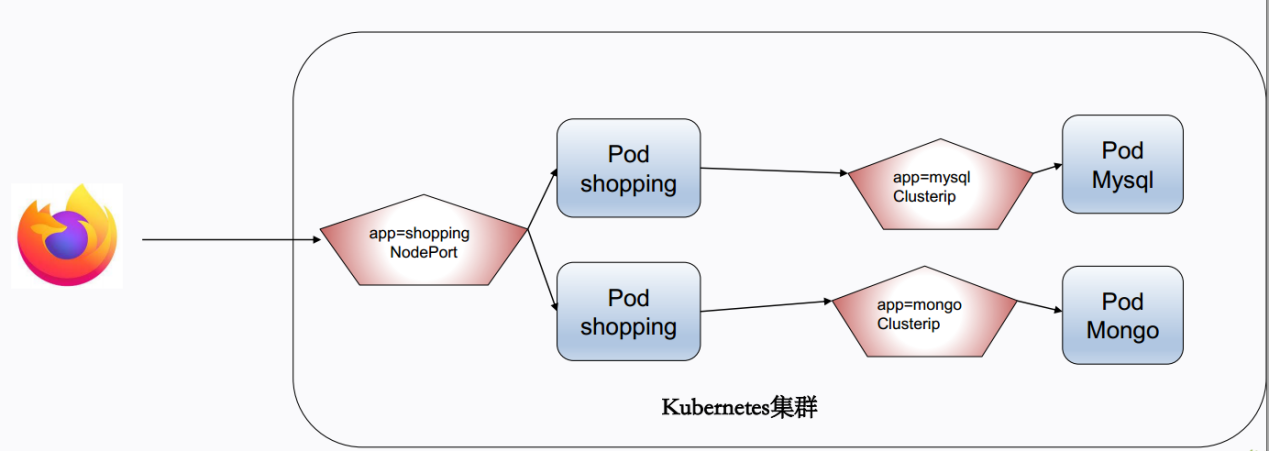
• DNS  
– 支持群集的DNS服务器（例如CoreDNS）监视 Kubernetes API 中的新服务，并为每个服务创建一组 DNS 记录  
– 例如：在 Kubernetes 命名空间 “my-ns” 中有一个名为 “my-service”的服务，则为"my-service.my-ns" 创建 DNS 记录

**Service type服务发布的类型：**

• Service Type 的取值以及行为如下：  
– ClusterIP：（默认）通过集群的内部IP暴露服务，选择该值，服务只能够在集群内部可以访问

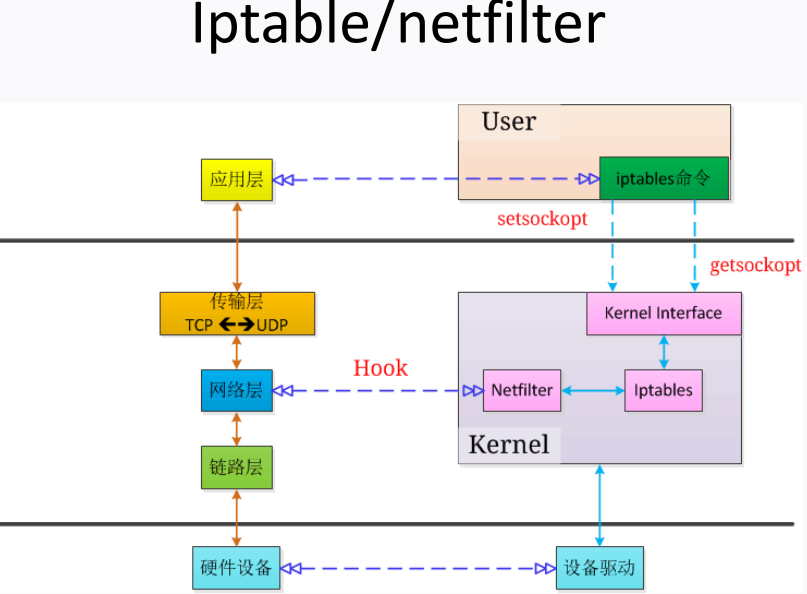
– NodePort：通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。通过请求 <NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务。

– LoadBalancer：使用云提供商的负载局衡器，可以向外部暴露服务。外部的负载均衡器可以路由到 NodePort 服务和 ClusterIP 服务。

– ExternalName：通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容. 没有任何类型代理被创建  


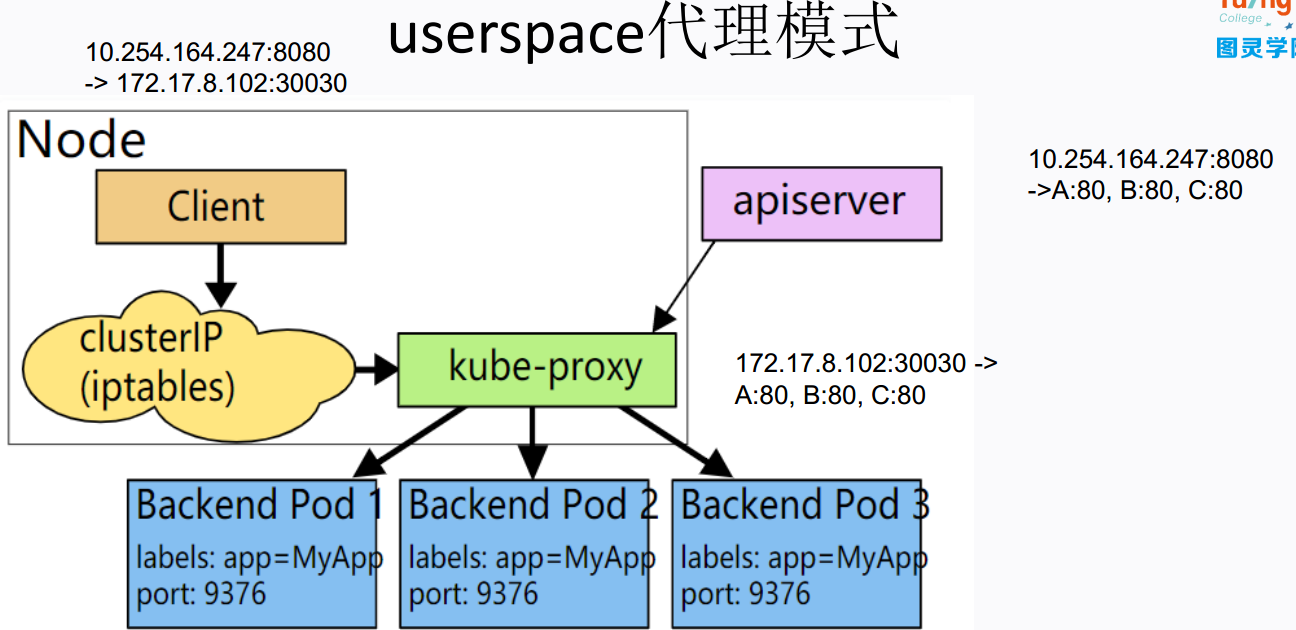
##### 1、Service clusterIP的网络实现

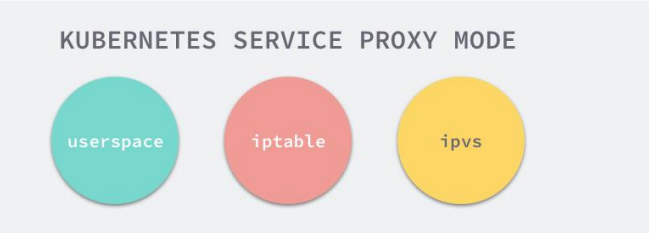
• Service ClusterIP (VIP)并不是真正的IP  
• 关键角色：kube-proxy  
– 每一个节点都运行着一个 kube-proxy 进程  
– 负责监听 Kubernetes 主节点中 Service 的增加和删除事件并修改运行代理的配置  
– 为节点内的客户端提供流量的转发和负载均衡等功能



ClusterIP的三种代理模式：

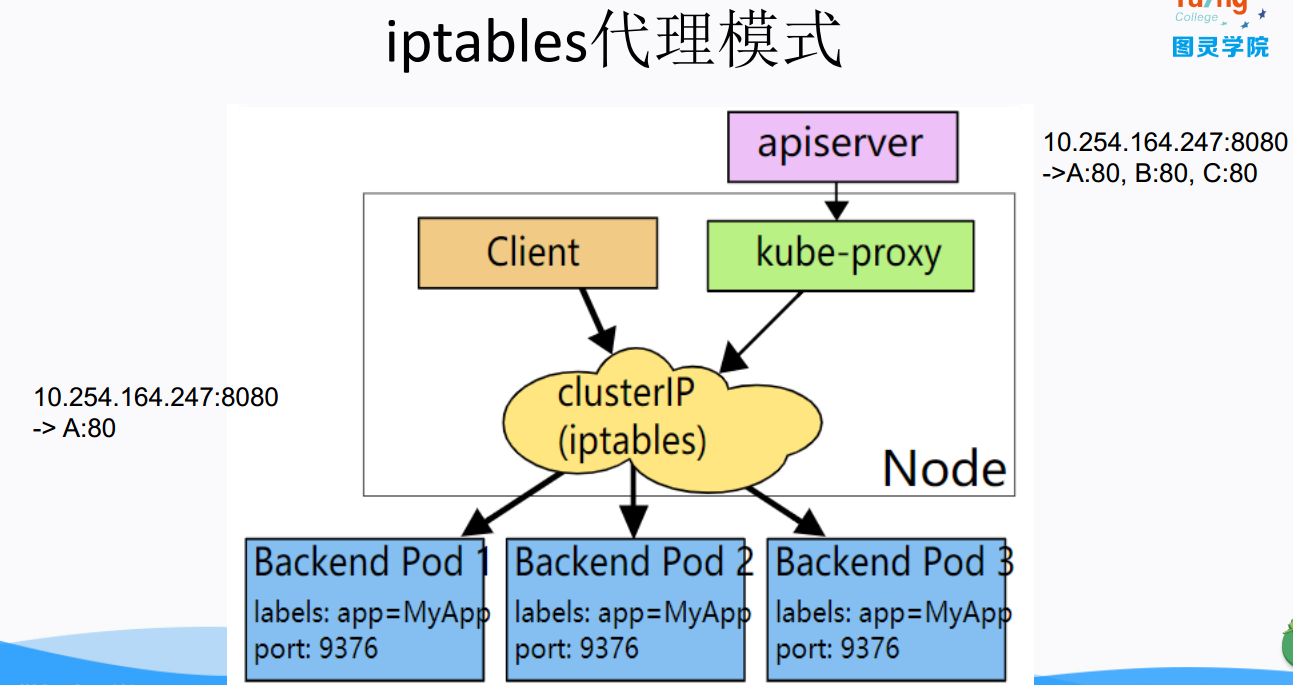
1、userspace代理模式





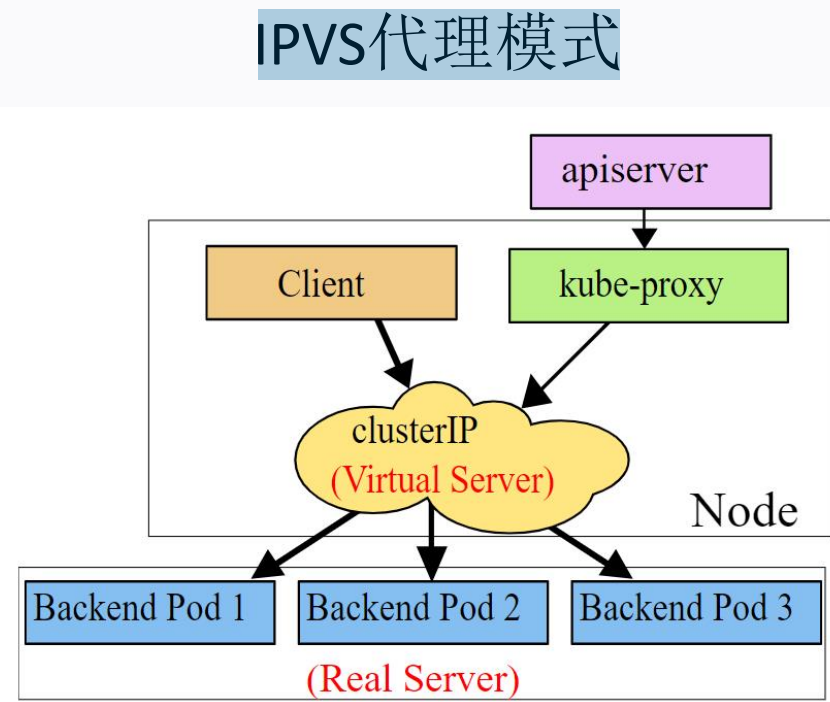
用户代理模式需要有一个用户到内核空间的过程，性能差已经被淘汰，首先从用户空间去获取pod对应的ip需要进入内核，然后从内核返回ip到用户，然后拿着ip继续访问

**2、iptables代理模式**



Iptables代理模式用户提前知道ip的存储位置则直接去拿ip然后访问，没有返回的过程

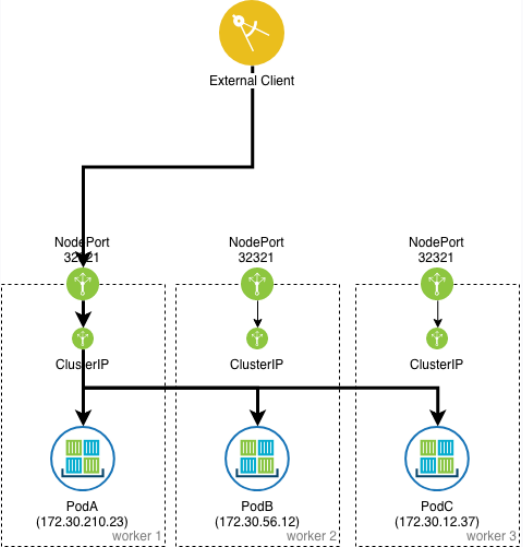
**3、IPVS代理模式**



##### 2、service NodePort

• NodePort示例与原理

-通过外部的负载均衡器将外部请求转发到各个NodeIP：NodePort以实现对外暴露服务端口转发



**• NodePort不足**  
– 每个端口只能提供一个服务  
– 只能使用端口 30000–32767  
– 如果节点 / VM IP 地址更改，则需要处理

– 每个暴露的服务需要使用一个公有云提供的负载均衡器 IP  
– 只提供 4 层负载均衡，不支持 7 层负载均衡功能

#### 5、ingress

K8s集群对外暴露服务的方式目前只有三种：**loadblancer、nodeport、ingress。**

下面详细讲解下ingress这个服务，ingress由两部分组成：ingress controller和ingress服务。

其中ingress controller目前主要有两种：基于nginx服务的ingress controller和基于traefik的ingress controller。

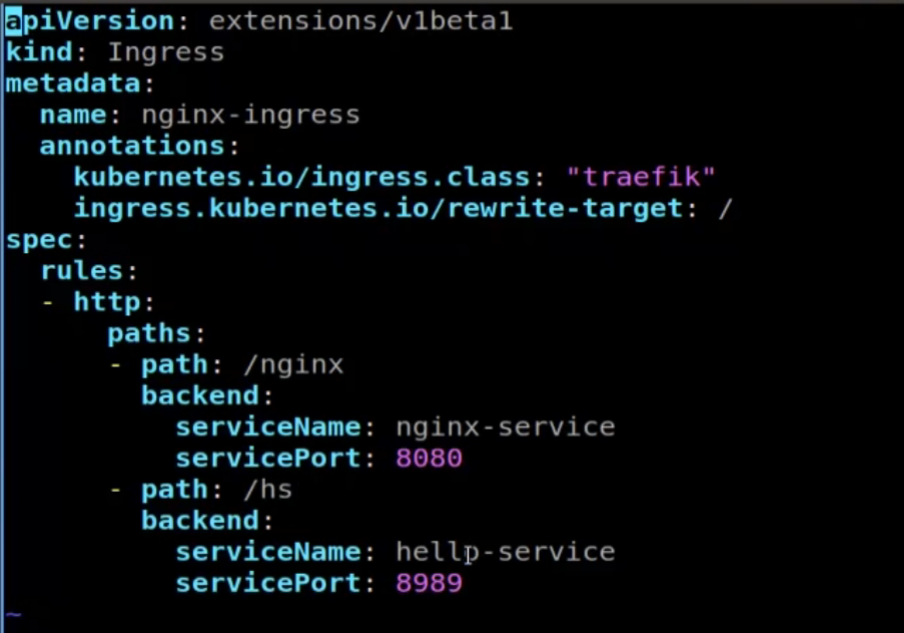
traefik的ingress controller，目前支持http和https协议。由于对nginx比较熟悉，而且需要使用TCP负载，所以在此我们选择的是基于nginx服务的ingress controller。

**ingress具体的工作原理如下:**

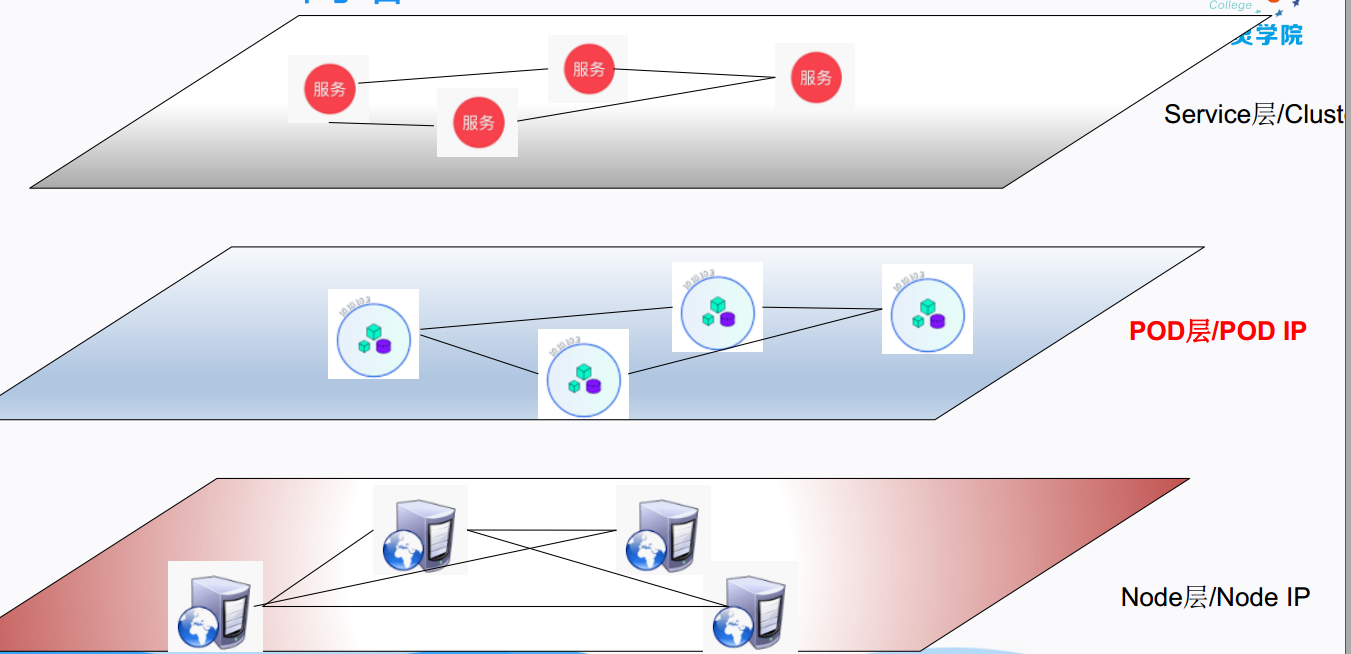
ingress contronler通过与k8s的api进行交互，动态的去感知k8s集群中ingress服务规则的变化，然后读取它，并按照定义的ingress规则，转发到k8s集群中对应的service。

而这个ingress规则写明了哪个域名对应k8s集群中的哪个service，然后再根据ingress-controller中的nginx配置模板，生成一段对应的nginx配置。

然后再把该配置动态的写到ingress-controller的pod里，该ingress-controller的pod里面运行着一个nginx服务，控制器会把生成的nginx配置写入到nginx的配置文件中，然后reload一下，使其配置生效。以此来达到域名分配置及动态更新的效果。



## 1.3 kubernetes内部网络



• Every Pod gets its own IP address.  
• Pod on a node can communicate with all pods on all nodes  
**without NAT**  
• Agents on a node (e.g. system daemons, kubelet) can  
communicate with all pods on that node

• Pod内 Container的通信  
– 共享网络空间  
• POD和POD的通信  
– 同一个NODE  
– 跨不同的NODE  
• Pod和Service