Kubernetes技术和相关命令简介

zhiliao_19EVG 2019-10-31 发表

组网及说明

H3Cloud CloudOS3.0 E3106

配置步骤

1. Kubernetes简介

Kubernetes是Google开源的一个容器编排引擎，它支持自动化部署、大规模可伸缩、应用容器化管理。在生产环境中部署一个应用程序时，通常要部署该应用的多个实例以便对应用请求进行负载均衡。kubernetes，简称K8s，是用8代替8个字符“ubernete”而成的缩写。

在Kubernetes中，我们可以创建多个容器，每个容器里面运行一个应用实例，然后通过内置的负载均衡策略，实现对这一组应用实例的管理、发现、访问，而这些细节都不需要运维人员去进行复杂的手工配置和处理。

2. Kubernetes功能模块

2.1 Kubernetes Pod

Pod是Kubernetes中能够创建和部署的最小单元，是Kubernetes集群中的一个应用实例，总是部署在同一个节点Node上。Pod中包含了一个或多个容器，还包括了存储、网络等各个容器共享的资源。Pod支持多种容器环境，Docker则是最流行的容器环境。Pod内的容器会一起启动、停止，每个Pod会有自己独立的内部动态IP，在Pod新建或重启时会重新分配新的IP。Pod会有自己的Label用来标示Pod的服务内容。Service会根据服务的Label来绑定Service与Pod之间的管理。Pod自身不具有高可用等特性，Pod一般不会直接使用，而是通过RC等方式进行调度使用。如图2-1所示的Pod1中有2个容器，其IP地址为10.10.10.1，Pod2中有3个容器，其IP地址为10.10.10.2，Pod3中有3个容器，其IP地址为10.10.10.3，Pod4中有4个容器，其IP地址为10.10.10.4。

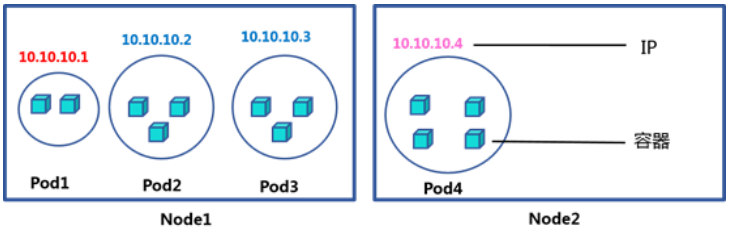

图 2-1 Pod

2.2 Kubernetes Label

Label是Kubernetes系统中另外一个核心概念，一个Label是一个key=value的键值对，其中key与value由用户自己指定。Label可以附加到各种资源对象上，例如Node、Pod、Service、RC等，一个资源对象可以定义任意数量的Label，同一个Label也可以被添加到任意数量的资源对象上去，Label通常在资源对象定义时确定，也可以在对象创建后动态添加或者删除。如图2-2所示的Pod1的Label=service1，Pod2的Label=service2，Pod3的Label=service2，Pod4的Label=service3。

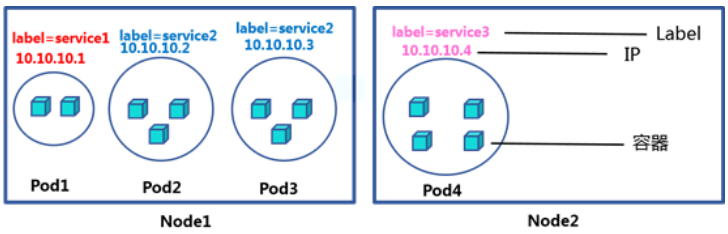

图 2-2 Label

2.3 Kubernetes Service

Kubernetes中一个应用服务会有一个或多个实例（Pod），每个实例（Pod）的IP地址由网络插件动态随机分配。为屏蔽这些后端实例的动态变化和对多实例的负载均衡，引入了Service这个资源对象，Service与其后端Pod副本集群之间则是通过Label Selector来实现"无缝对接"。用户访问Pod的服务均需要通过Service进行。每个Service会分配一个独立的ClusterIP，并通过Selector的Label标示来选择相应的Pod。如果有多个相同Label的Pod，Service服务会自动在Pod之间Round-Robin。（负载均衡算法），ClusterIP随着Service的生命周期产生销毁，期间不会发生变化。如图2-3所示，Service1对应的pod为pod1，其Cluster IP地址为10.1.0.10，端口号为1000，Service2对应的pod为pod2和pod3，其Cluster IP地址为10.1.0.11，端口号为4321，Service3对应的pod为pod4，其Cluster IP地址为10.1.0.12，端口号为1234。

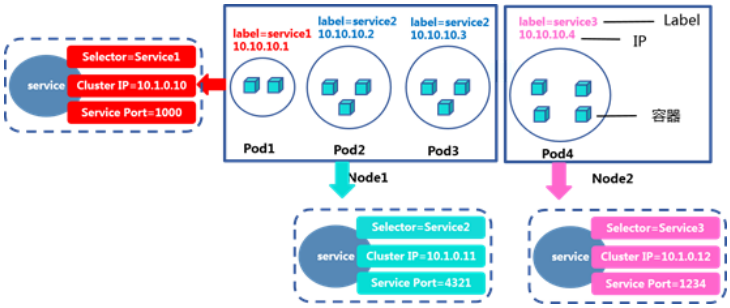

图 2-3 Service

2.4 Kubernetes RC

Kubernetes RC是Pod的复制、管理、监控工具，Pod自身不具有高可用的特性，而RC则提供了一系列的高可用特性。例如设定RC的replication数量为2，那么相同的Pod会被创建2次，例如Label=Service2的Pod，如果Pod2出现问题而失效（例如物理机器down），那么RC会发现replication的数量变成了1，则会自动的再创建一个Label=Service2的Pod，保证服务的可用性。RC是Kubernetes使用POD推荐的方法，即使只建立一个Pod，也要使用RC来创建，从而保证服务的可用性。如图2-4所示，label=service2的pod有两个副本。

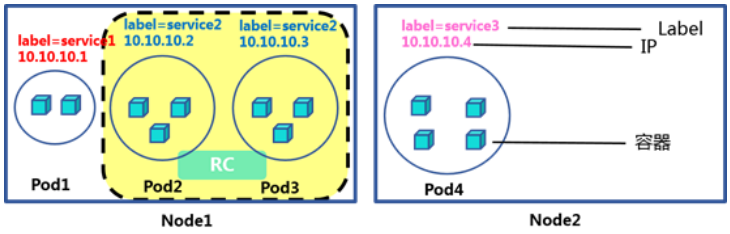

图 2-4 Replication Controller

2.5 etcd

etcd是一个分布式的Key/Value存储系统，数据写入节点中后会自动的同步到其他的节点之上。etcd通过raft算法自主进行master选举，当master失效时，会自动重新选择新的master节点，从而保证etcd集群的高可用，如图2-5所示，当一个节点的数据更新时数据会同步到其他节点上。

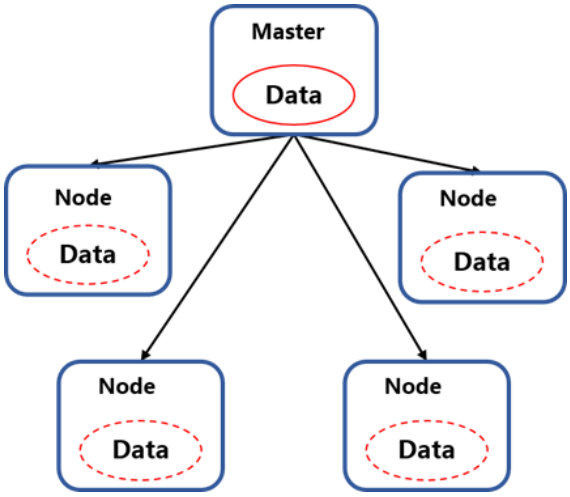

图 2-5 etcd

3. Kubernetes组件及架构

一个K8S系统，通常称为一个K8S集群，这个集群主要包括两个部分：一个Master节点（主节点）：负责管理和控制，一群Node节点（计算节点）：工作负载节点，里面是具体的容器，如图3-1所示是一个典型的K8S架构。

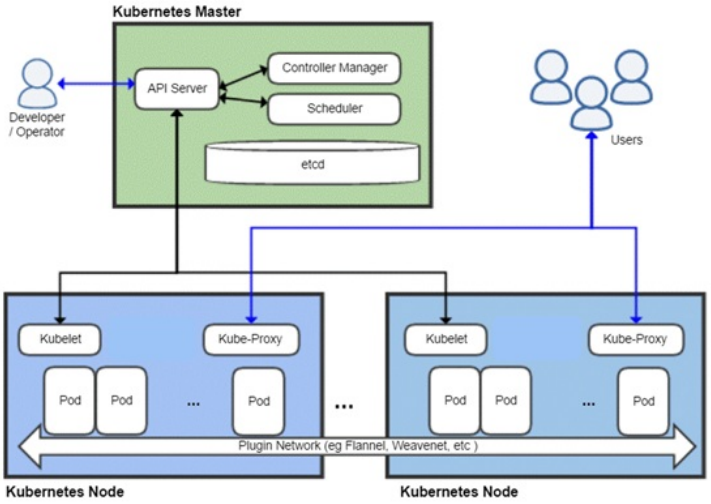

图 3-1 K8S架构

3.1 Master节点

Master节点包括API Server、Scheduler、Controller manager、etcd等服务。Scheduler负责对集群内部的Pod进行调度，相当于“调度室”。Controller manager负责集群的管理，相当于“大总管”。API Server是整个系统的对外接口，供客户端和其它组件互相通信，相当于“营业厅”。etcd负责集群的数据同步，相当于存放数据的“仓库”，如图3-2所示就是一个Master节点及其所包含的服务。

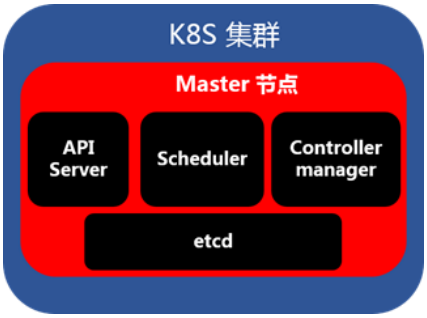

图 3-2 Master节点

3.1.1Controller manager

Controller Manager是各种controller的管理者,是集群内部的管理控制中心，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）、服务账号（ServiceAccount）、资源定额（ResourceQuota）的管理，当某个Node意外宕机时，Controller Manager会及时发现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。

3.1.2Scheduler

Scheduler只负责Pod调度，通过算法来计算pod和Node节点的对应关系。在整个系统中起“承上启下”作用，承上：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。如图3-3所示，Pod1、Pod2和Pod3对应在Node1上，Pod3和Pod4对应在Node2上。

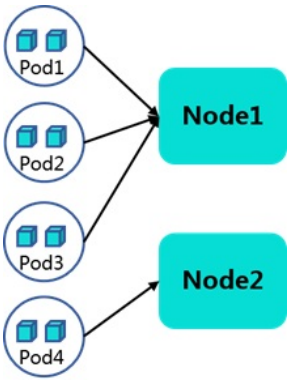

图 3-3 Scheduler

3.1.3API Server

API server作为集群的核心，负责各个功能模块之间的通信。集群中各个模块通过API server将信息存入etcd，当需要获取和操作这些数据时，则通过API server提供的REST接口来实现，从而实现各模块之间的信息交互。集群内部各个模块之间通信的枢纽：所有模块之前并不会之间互相调用，而是通过和 API Server 打交道来完成自己那部分的工作，集群之间各个组件的通信关系如图3-4所示。

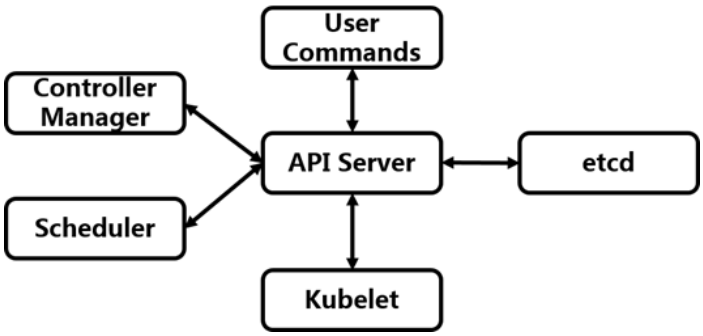

图 3-4 API Server

3.2Node节点

Node是工作负载节点，上面承载着容器，Node节点包括Pod、kubelet、kube-proxy。Pod是Kubernetes最基本的操作单元。一个Pod代表着集群中运行的一个进程，它内部封装了一个或多个紧密相关的容器。Kubelet主要负责监视指派到它所在Node上的Pod，包括创建、修改、监控、删除等。Kube-proxy：对Node提供网络代理和LB功能，配合Service提供网络服务。如图3-5所示为两个Node节点，Node1上面有Pod1、Pod2和Pod3，Node2上面有Pod4。

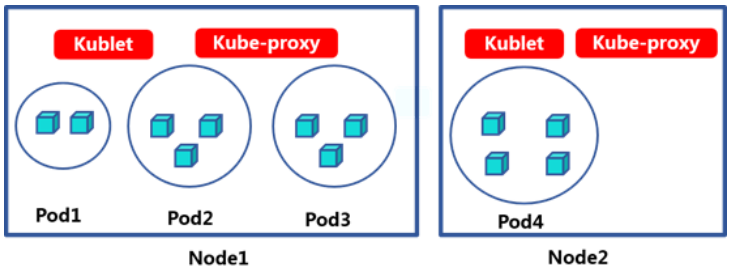

图 3-5 Node节点

3.2.3Kubelet

kubelet 是运行在每个节点上的主要的“节点代理”，每个节点都会启动 kubelet进程，用来处理 Master 节点下发到本节点的任务，管理Pod 和其中的容器。其功能主要为：1）Pod管理，获取Pod的状态，运行的容器数量，种类，副本数量，网络配置等。2）容器监控：定时汇报当前节点的资源使用情况给API Server，让Master节点知道整个集群所有节点的资源情况，以供调度时使用。3）容器健康状态检查：如果容器运行出错，就要根据设置的重启策略进行处理。4）镜像和容器的清理工作：保证节点上镜像不会占满磁盘空间，退出的容器不会占用太多资源。如下图所示，Node1和Node2上的Kubelet获取了Pod状态后，通 过API Server将Pod状态告知Master节点。如图3-6所示为Kubelet服务。

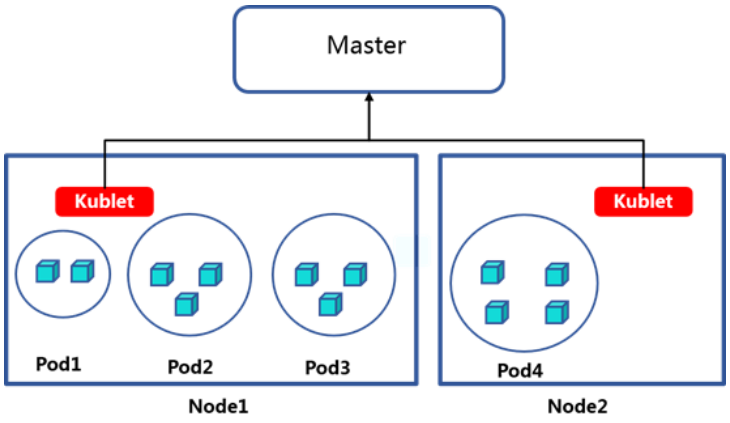

图 3-6 Kubelet

3.2.2Kube-proxy

kube-proxy的这个组件运行在每个node节点上。kube-proxy进程其实就是一个智能的软件负载均衡器，它会负责把对Service的请求转发到后端的某个Pod实例上并在内部实现服务的负载均衡与会话保持机制。它监听API server中service和endpoint的变化情况，并通过iptables等来为服务配置负载均衡，是让我们的服务在集群外可以被访问到的重要方式。Kube-proxy与service在集群中的工作原理如图3-7所示。

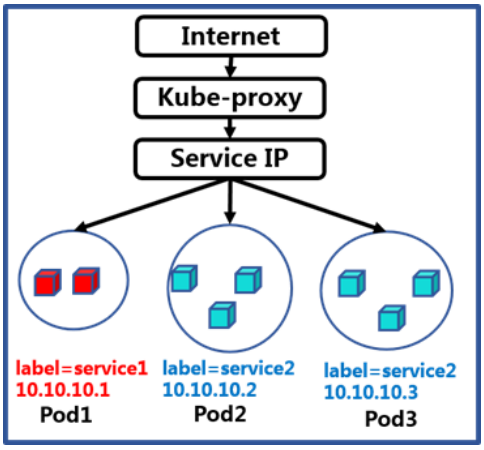

图3-7 Kube-proxy工作原理

配置关键点

4. 实验

根据上述所介绍的Kubernetes架构，我们直接通过在H3Cloud OS 3.0的实际环境来帮助我们更好的理解。

4.1获取集群节点信息

首先，我们搭建好了Kubernetes的集群环境，输入命令检查H3Cloud OS系统运行状态，查看各节点状态信息，命令为“/opt/bin/kubectl –s 127.0.0.1:8888 get node”，如图4-1所示各节点都处于Ready状态。

图 4-1 节点状态信息

4.2 获取集群Pod信息

H3Cloud OS采用容器化架构，Pod的运行状态反应了提供服务的容器的状态，Pod状态运行正常即表示相关服务正常。使用root用户登录Master节点，执行以下命令查看服务组件所在节点，通过输入命令：“/opt/bin/kubectl –s 127.0.0.1:8888 get pode -o wide”可以看到如图4-2所示的Pod信息。NAME列显示了各个Pod的名称，READY列显示了Pod的运行个数和设定的副本数，1/1中的前一个1表示当前运行了一个Pod，后一个1表示此Pod的设定副本数为1，STATUS列显示了Pod的运行状态，可以观察到pod都是处于running状态的，AGE列显示了Pod的运行时间，可以看到大部分的Pod的运行时长都在20小时以上，IP列显示了Pod的IP地址，NODE列显示了该服务组件所在节点的IP地址。

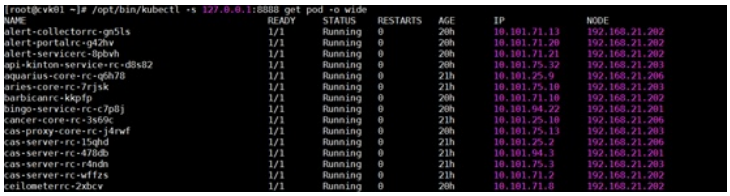

图4-2 Pod信息

4.3 获取集群Docker信息

查看H3Cloud OS云平台使用容器进程的运行状态：输入命令docker ps ，如图4-3所示，输出显示的第一列为容器的UUID信息，第二列为容器使用的镜像名称，第三列为启动容器时运行的命令，第四列为容器的创建时间，显示格式为**时间之前创建，第五列为容器的运行状态，第六列为容器的端口信息和使用的连接类型（tcp\udp）名称，第七列为容器的名称。

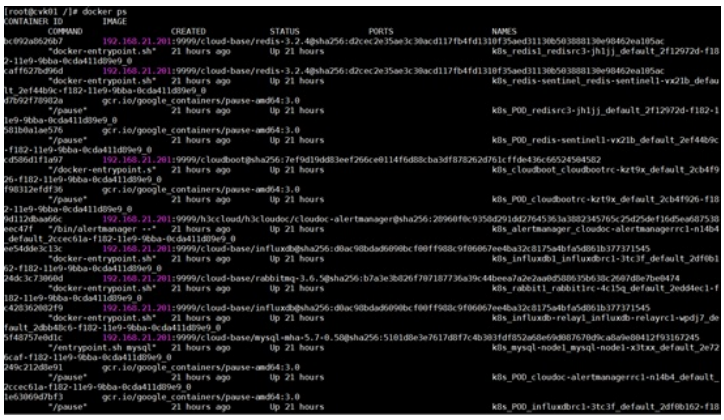

图 4-3 Docker信息

通过使用命令：pod | grep glance可以只查看包含glance名称的容器实例，如图4-4所示。

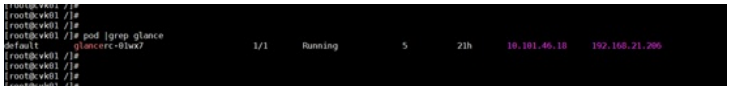

图 4-4 包含glance名称的容器实例

进入Glance容器实例中，使用命令：kubectl exec –it 名称> bash；H3Cloud OS业务均有不同的容器提供，容器内服务状态异常会导致相关的功能异常。容器内执行命令：systemctl status <服务名称>，结果如图4-5所示。

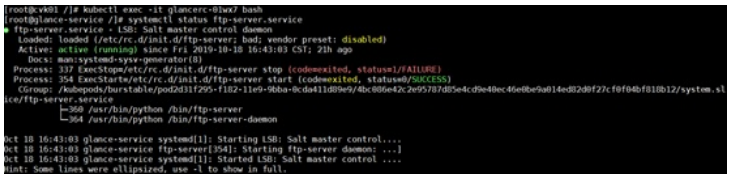

图 4-5 容器服务状态

附件下载：Kubernetes技术和相关命令简介-王瑜21360.zip