针对H3Cloud OS中Docker引擎的探究

zhiliao_VIWw1 2019-10-31 发表

组网及说明

该探究是基于正常的H3Cloud OS技术环境之下。

配置步骤

1. Docker基本概念

为了区别于CAS虚拟化平台中虚拟机，在学习H3Cloud OS技术前首先要掌握一定的Docker引擎原理与应用。Docker是PaaS提供商dotCloud开源的一个基于LXC的高级容器引擎，采用Go语言编写并遵从Apache2.0协议开源。其可以打包自身的应用程序以及依赖包到一个可移植镜像中，然后发布到任何一个流行的Linux或Windows机器上，也可以实现虚拟化。

1.1 容器与虚拟机

在理解Docker的同时往往首先去区分清楚两个概念，即容器和虚拟机。如图1.1所示，虚拟机类似常用的VMware与VisualBox一样，它们需要模拟整台机器，其中包括CPU、内存、磁盘、网卡等全套硬件系统，在虚拟机被开启时，预分配给它的资源将全部被占用而无法实现资源的动态分配。并且在每一台虚拟机中包括有应用，必要的二进制和库，以及一个完整的用户操作系统。

而相对的容器技术是和我们的宿主机之间一起去共享硬件资源与操作系统，进而可以实现资源的动态分配。每个容器包含其独有的应用和其所有的依赖包，但是与其他容器一起共享内核资源。另外容器在宿主机操作系统中，在用户空间以分离的进程运行。

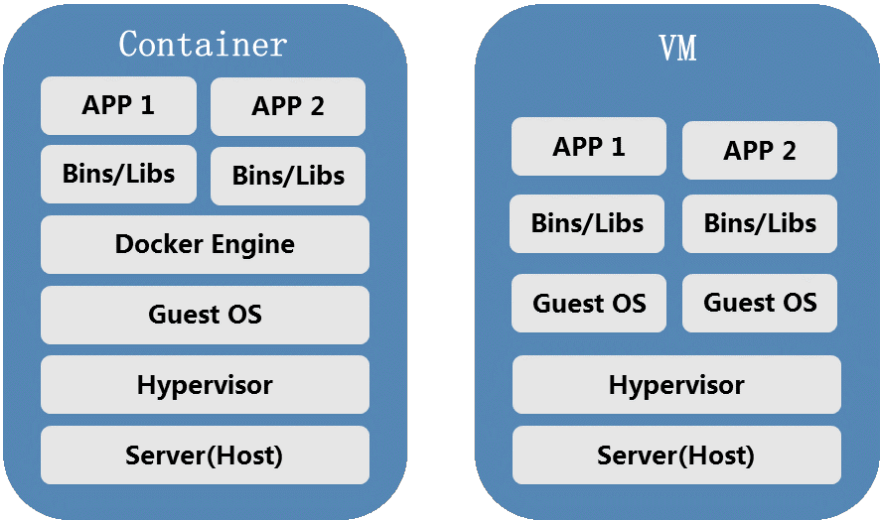


图1.1 容器与虚拟机对比图

总体来说，由于在启动应用之前不需要操作系统的启动过程，容器在启动速度方面要远远高于虚拟机，其速度几乎可以与在服务器上直接启动应用的速度相媲美。并且在镜像大小与资源消耗上容器也优于虚拟机，这使得容器在相同的物理资源上可以更加便捷的传输分发，提高利用率，减少承载容量。开发者通过使用容器，可以轻松打包应用程序的代码、配置和依赖关系，将其变成容易使用的构建块，从而实现环境一致性和版本控制等诸多目标。此外容器还可以帮助保证应用程序快速、可靠、一致地部署期间不受部署环境的影响。

1.2 Docker特点及用途

沙箱是一个虚拟系统程序，其提供的运行环境相对于每一个运行的程序都是相互独立的，不会对现有的系统产生影响，也就是说沙箱提供了一个权限，用来限制应用程序去访问系统资源。Docker容器则完全使用沙箱机制，保证容器与容器之间没有任何接口。另外沙箱机制使得测试环境到生产环境的部署与项目的迁移过程中，应用组件经Docker的封装之后可保证应用程序与运行环境保持一致，减少在环境搭建过程中的工作量。Docker几乎没有性能开销,可以在不依赖于任何语言、框架包括系统情况下很容易地在机器和数据中心中运行。

由于Docker基于LXC的轻量级虚拟化的特点，Docker相比KVM之类最明显的特点就是启动快，资源占用小。正是因为这些特点Docker具有广泛的用途，本文将从如下八个用途作出具体介绍。

1. 简化配置：作为Docker的初始目的，他能够将环境与配置在代码中部署，并且类似于VM，其配置能够在各种环境中使用，实际上也就是将应用环境和底层环境实现了解耦。

2. 代码流水线管理：可以对代码以流式pipeline管道化进行管理，从测试机器到生产环境机器这个流程中都能有效管理。虽然环境之间会存在细小的差别，但Docker可以提供跨环境一致性微环境，实现从开发到部署的流畅发布。

3. 开发人员的生产化：为了能够更加充分的了解服务的性能，往往将开发环境无限的去趋近生产环境，就此可以将多个Docker装载一系列服务运行在单机上最大程度模拟生产分布式部署的环境。

4. 应用隔离：当需要在一台服务器上运行多个应用，需要将monolithic的应用切分为很多微服务。因此为实现应用之间的解耦，可以将多个应用服务部署在多个Docker中来达到目的。
5. 服务合并：Docker能以更加紧密资源提供更有效的服务合并以降低费用，不占用更多的操作系统内存。
6. 多租户：Docker可以作为云计算的多租户容器，它能够容易为每个租户创建运行对应的多个实例。
7. 调试能力：Docker提供了很多适用于容器的工具。它们提供了很多功能,包括可以为容器设置检查点,设置版本,查看两个容器之间的差别等等,这些特性可以帮助开发者调试服务中出现的Bug。
8. 快速部署：在Docker为进程创建一个容器过程中,因为不需要启动一个操作系统,所以可以将时间大大缩短到秒级别.同时可以在数据中心创建销毁资源而无须担心重新启动带来的开销。通常数据中心的资源利用率只有30%,通过使用Docker并进行有效的资源分配可以提高资源的利用率。

2. Docker相关组件

Docker的运行离不开内部各组件的分工合作，为了更加细化对Docker的理解，下面将从镜像，容器，仓库等组件进行讲解。

2.1 Docker image

与VM的镜像相似，Docker镜像包含了特定内容的打包。其只属于固定的只读模板，一般放置在公有或私有的Docker hub之上，供Docker进程下载或上传。简单说Docker镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。在镜像中是不包含任何动态数据，其内容在构建之后也不会被改变。

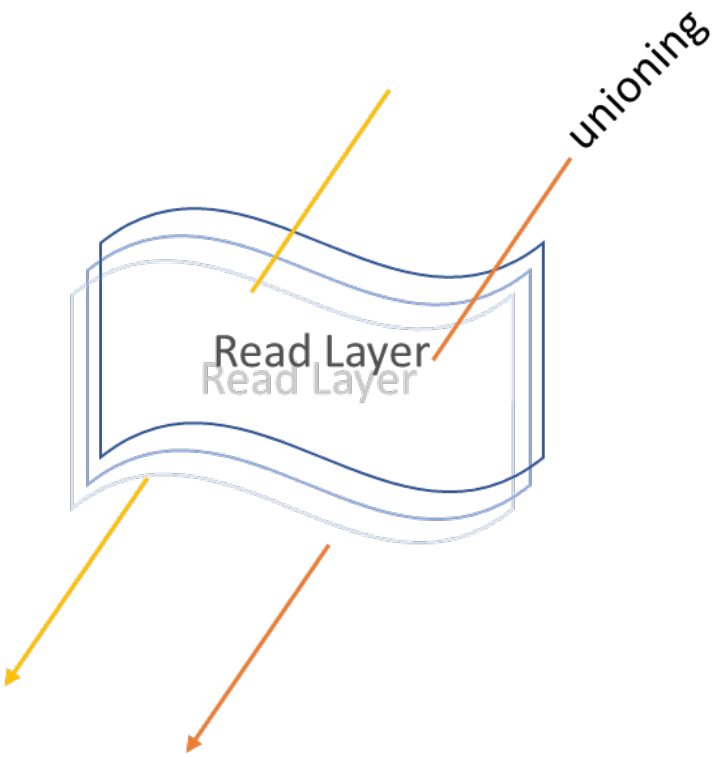


图2.1 只读层统一视角图

如图2.1所示，从侧面角度观察，它们是多个只读层（read-only layer）互相重叠而成，并且除去最后一层永远都会有一个指针指向下一层。这些只读层其实就是Docker内部实现的细节，它们都可以在主机的文件系统上进行访问。而统一文件系统(union file system)技术将这些只读层合并到一个文件系统之中，并为这些层定义了一个统一的视角。这样许多繁琐的只读层就会被隐藏起来，如图中箭头指向的视角一样，在用户看来只有一个文件系统。所以可以说镜像（Image）就是一堆只读层的统一视角。

2.2 Docker Container

如图2.2所示，容器(container)的定义和镜像(image)几乎一模一样，也是一堆层的统一视角，而唯一区别在于容器的最上面那一层是可读可写层。所以说Docker容器可以说是Docker镜像的实例，可以通过指定相关的磁盘、CPU、内存配额，端口映射等参数来建立所需要的相关容器，其与虚拟化平台中的虚拟主机非常类似可以类比理解。

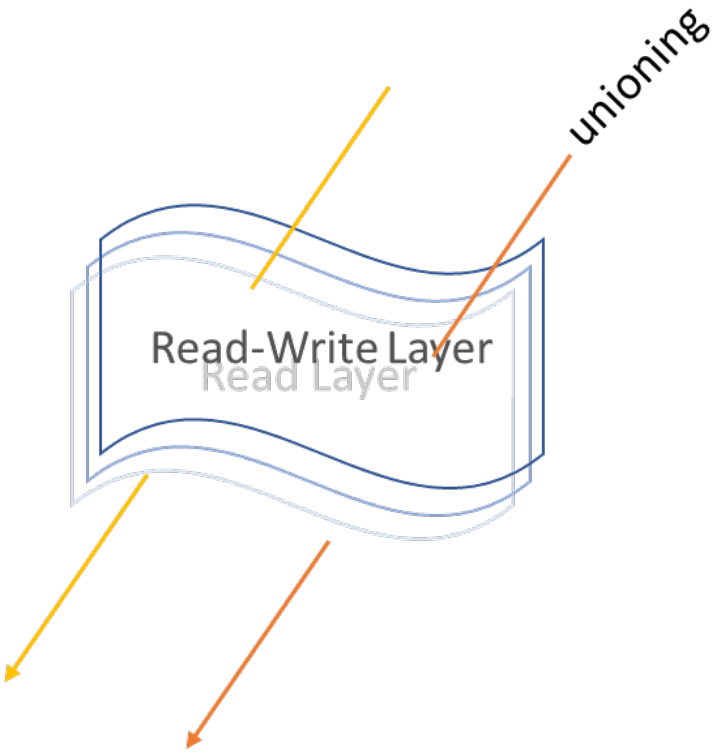


图2.2 容器统一视角图

由于容器的定义并没有提及是否要运行容器，所以实际上，容器=镜像+读写层。在完成部署之后Docker容器会提供具体的应用服务。

2.2 Docker Repository

Docker仓库也可以说是Docker Hub是存放镜像文件的地方。新建立好的镜像往往很容易在当前的宿主上运行，所以如果需要在其它的服务器上运用这个镜像，就需要一个集中的存储和分发镜像的服务，而Docker Registry(仓库注册服务器)就可以提供这样的服务。这里需要区分清楚仓库(Repository)和仓库注册服务器(Registry)之间的区别，Docker仓库的概念跟Git类似，注册服务器可以理解为GitHub这样的托管服务。也就是说一个仓库注册服务器（Docker Registry）中包含有多个仓库(Docker Repository)，每个仓库则包含有多个标签(Tag)，标签与镜像一一对应来标识同一软件中各个不同版本的镜像。所以说，镜像仓库是Docker用来集中存放镜像文件的地方类似于我们之前常用的代码仓库。仓库又可以分public(公有仓库)与private(私有仓库)两种形式。Docker Registry公有仓库是开放给用户使用、用户可以在其中管理镜像的Registry服务，同时用户可以免费上传、下载公开的镜像，并可能提供收费服务来使用户管理私有镜像。除了公开服务之外，用户还可以在本地搭建私有Docker Registry。Docker官方提供了Docker Registry镜像，可以直接使用作为私有Registry服务。当用户创建了属于自己的镜像以后就可以通过push命令将它上传到公有或私有的仓库之中，如此一来下次在另外一台服务器上使用这个镜像时，只需从仓库上pull下来即可。

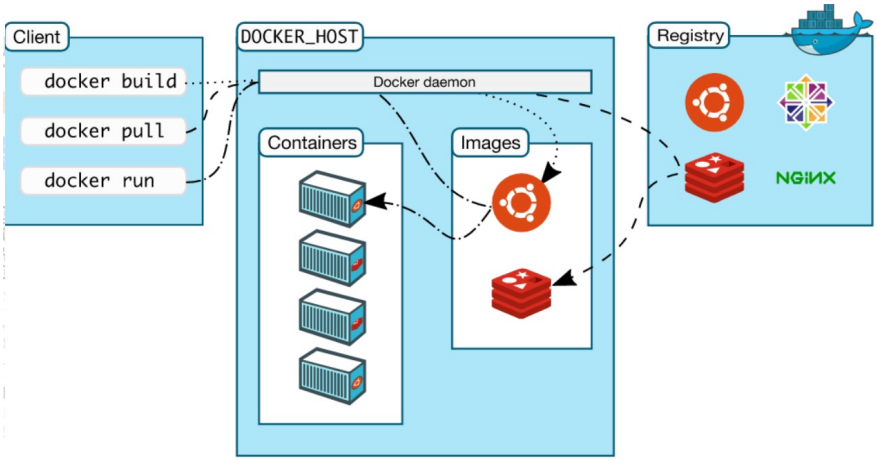


图2.3 镜像分发系统图

如图2.3所示是Docker客户端、服务端和Docker仓库（即Docker Hub和Docker Cloud），默认情况下Docker会在Docker中央仓库寻找镜像文件，这种利用仓库管理镜像的设计理念与Git相类似，而这个仓库是也可以通过修改配置来指定，甚至可以创建属于用户自己的私有仓库。

2.3 Docker其他体系架构

Docker文件系统的构成是AUFS（AnotherUnionFS）。其支持将不同目录挂载到同一个虚拟文件系

统之下。因为分层机制与copy-on-write特性，各个容器之间可以共享底层的只读文件系统，进而可以达到节省存储空间，快速部署，便捷升级的功能。

Docker通过LXC以及Cgroup来实现相关环节的隔离。LXC可以实现Docker的基础容器化，而Cgroup则实现了对CPU、内存使用的隔离。它们使得Docker容器可以更加的方便部署与迁移。

Docker Volume还会在宿主机上建立相关的volume，用于存储持久化的数据。其映射既可以是自动分配的目录（隐形）也可以是人工指定的目录（显性）。除此之外Volume还具有传递性，用户可以通过volume-from的方式，在其他的容器中通过数据容器访问Volume中的数据。

Docker Link属于一种Docker内部的连接方式。在常规的Docker使用方法中，需要将容器的端口与Host进行映射，将端口暴露出来。而采用Docker link的方式，则类似于沙箱机制一样可以直接进行容器层面的关联。

Docker会在宿主机上默认启用一个Docker0的虚拟网卡，宿主机上的所有容器都会通过Docker0与外界进行通信，同时容器之间的通信也会通过Docker0端口进行。在非host模式下，Docker0是Docker与主机网络之间沟通的桥梁，而在host模式下容器将直接使用宿主机的网络接口。

3. H3Cloud OS中Docker常用命令介绍

H3Cloud OS云管理平台基于业界标准的OpenStack开源项目，能够帮助用户快速搭建起方便管理、方便维护、可大规模扩展的多租户云计算平台，提供多租户管理能力，通过自定义组织结构、审批流程，满足不同租户需求，提供灵活的计费策略，满足不同的运营需求，通过构建私有网络和业务编排，让用户自由搭建和管理自己的私有云环境，实现云基础架构服务（Infrastructure as a Service, IaaS）。下面将针对H3Cloud OS中Docker的常用命令作出介绍。

1. 通过# docker ps命令查看H3Cloud OS大云平台使用容器进程的运行状态，具体如图3.1所示：

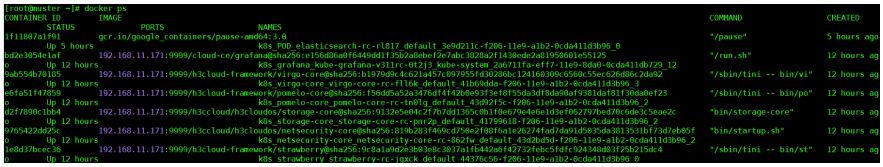


图3.1 容器进程运行状态图

其中第一列CONTAINER ID为容器的UUID信息，第二列IMAGE为容器的镜像名称，第四列CREATED为容器进程创建时间，第五列STATUS为容器运行状态显示已运行的时长。

此外运行命令# docker ps | grep openstack则是如图3.2所示的openstack相关的容器状态。

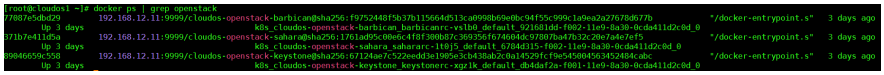


图3.2 openstack相关容器状态图

如图所示，H3Cloud OS的控制器容器名称为“192.168.12.11:9999/cloudos-openstack-barbican@sha256:f9752448f5b37b115664d513ca0998b69e0bc94f55c999cla9ea2a27678d677b”，对应的UUID值为“77087e5dbd29”，UUID为唯一值；

2.进入容器操作,执行命令/opt/bin/kubectl -s 127.0.0.1:8888 exec -it <UUID> bash。可以首先通过命令/opt/bin/kubectl -s 127.0.0.1:8888 get pod | grep nova先搜索到节点的UUID信息，接着如图3.3所示对应的容器的UUID值为novarc-1lj617，随后执行命令：/opt/bin/kubectl -s 127.0.0.1:8888 exec -it novarc-1lj617 bash进入容器。

```
[root@cloudos1 ~]# /opt/bin/kubectl -s 127.0.0.1:8888 get pod | grep nova
novarc-1lj17                    1/1       Running            0          3d
[root@cloudos1 ~]# /opt/bin/kubectl -s 127.0.0.1:8888 exec -it novarc-1lj17  bash
[root@nova-service /]#
[root@nova-service /]#
```

图3.3 进入容器操作图

3.执行命令# hostname查看H3Cloud OS主机名如图3.4所示。

```
[root@cloudos1 ~]# hostname
cloudos1
```

图3.4 主机名图

4.在控制节点或计算节点容器中，如果要查看openstack相关进程服务的状态，需要先执行加载环境变量，用于openstac内部组件间授权。如图3.5所示，执行命令# source /root/admin-openrc.sh该命令运行正常无任何回显输出。

```
[root@nova-service /]#
[root@nova-service /]# source /root/admin-openrc.sh
[root@nova-service /]#
[root@nova-service /]#
```

图3.5 内部组件间授权命令图

5.在计算节点容器内执行命令# nova service-list来查看nova进程状态。

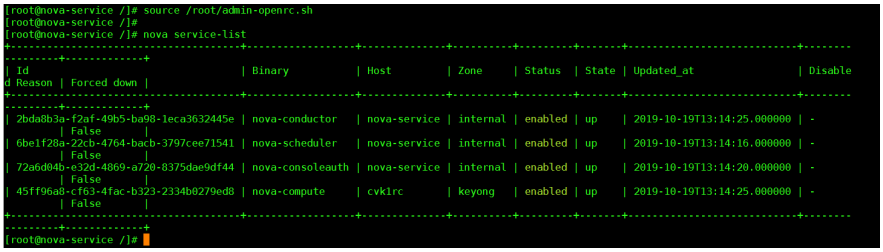


图3.6 nova进程状态图

执行结果如图3.6所示，反馈出的是nova所有服务状态信息，包括服务Id、服务名称、服务所在主机名、计算可用域名称、服务运行状态、服务启动时间、服务终止运行的原因。以Id为2bda8b3a-f2af-49b5-ba98-1eca3632445e的nova-conductor为例,对应的计算可用域名称为internal。对应的状态为“可用”，并处于“up”状态。服务启动时间为：2019-10-19T13:14:25.000000。这里值得注意的是在执行nova service-list命令前必须执行source /root/admin-openrc.sh否则会出现如图3.7所示的报错信息。

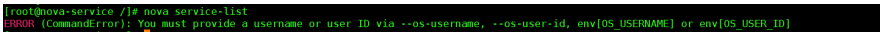
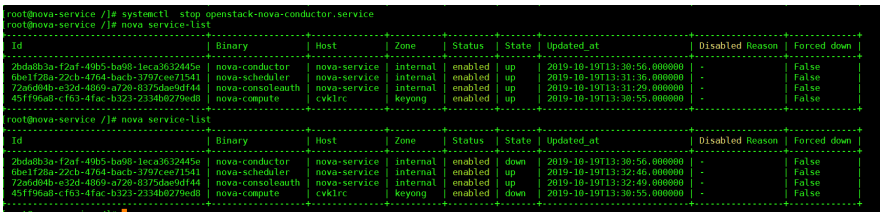


图3.7 命令报错图

6.手动停止和启动nova服务操作。必须进入对应容器中，执行命令# systemctl stop openstack-<Binary>.service如若要开启，则将stop改为start即可。如图3.8所示此处以nova-conductor为例执行。



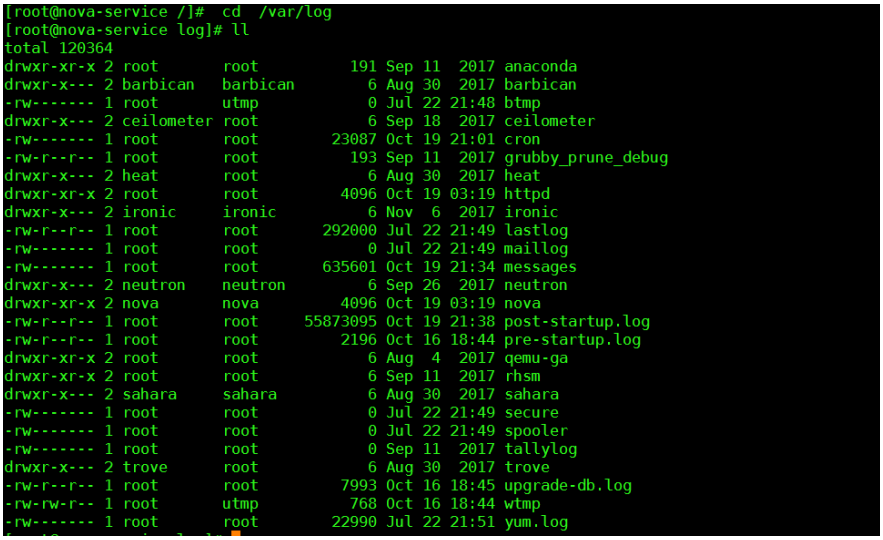


图3.8手动停止和启动命令（注意开启与停止都需等待约2分钟左右）

7.在计算节点容器内查看日志信息如图3.9所示，日志所在目录为：/var/log

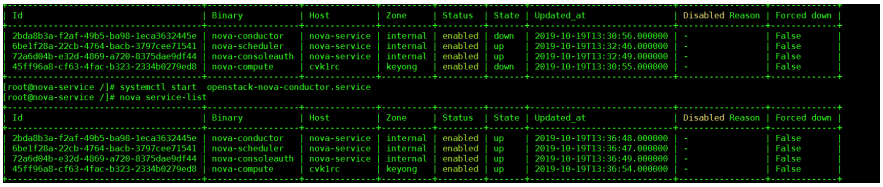
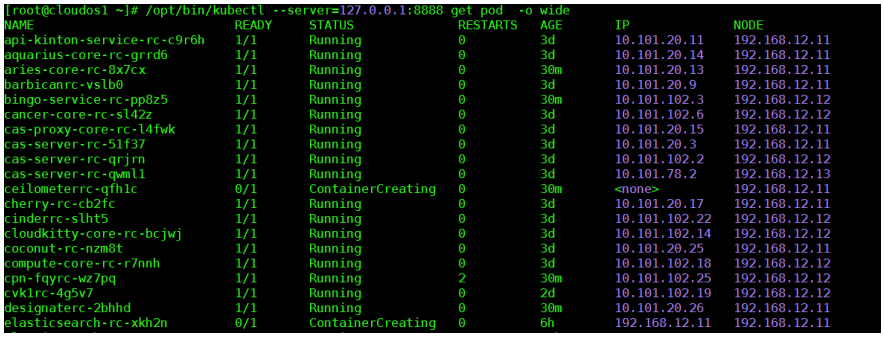


图3.9 日志信息图

9.执行命令# /opt/bin/kubectl --server=127.0.0.1:8888 get pod -o wide即使用kubernetes管理工具来查看容器进程运行状态信息，注意执行该命令，无需进入任何容器，如图3.10所示。



```
[root@cloudos1 ~]# /opt/bin/kubectl --server=127.0.0.1:8888 get pod  -o wide
NAME                          READY     STATUS              RESTARTS   AGE       IP              NODE
api-kinton-service-rc-c9r6h   1/1       Running             0          3d        10.101.20.11    192.168.12.11
aquarius-core-rc-grrd6        1/1       Running             0          3d        10.101.20.14    192.168.12.11
aries-core-rc-8x7cx           1/1       Running             0          30m       10.101.20.13    192.168.12.11
barbicanrc-vslb0              1/1       Running             0          3d        10.101.20.9     192.168.12.11
bingo-service-rc-pp8z5        1/1       Running             0          30m       10.101.102.3    192.168.12.12
cancer-core-rc-sl42z          1/1       Running             0          3d        10.101.102.6    192.168.12.12
cas-proxy-core-rc-l4fwk       1/1       Running             0          3d        10.101.20.15    192.168.12.11
cas-server-rc-51f37           1/1       Running             0          3d        10.101.20.3     192.168.12.11
cas-server-rc-qrjrn           1/1       Running             0          3d        10.101.102.2    192.168.12.12
cas-server-rc-qwml1           1/1       Running             0          3d        10.101.78.2     192.168.12.13
ceilometerrc-qfh1c            0/1       ContainerCreating   0          30m       <none>          192.168.12.11
cherry-rc-cb2fc               1/1       Running             0          3d        10.101.20.17    192.168.12.11
cinderrc-slht5                1/1       Running             0          3d        10.101.102.22   192.168.12.12
cloudkitty-core-rc-bcjwj      1/1       Running             0          3d        10.101.102.14   192.168.12.12
coconut-rc-nzm8t              1/1       Running             0          3d        10.101.20.25    192.168.12.11
compute-core-rc-r7nnh         1/1       Running             0          3d        10.101.102.18   192.168.12.12
cpn-fqyrc-wz7pq               1/1       Running             2          30m       10.101.102.25   192.168.12.12
cvk1rc-4g5v7                  1/1       Running             0          2d        10.101.102.19   192.168.12.12
designaterc-2bhhd             1/1       Running             0          30m       10.101.20.26    192.168.12.11
elasticsearch-rc-xkh2n        0/1       ContainerCreating   0          6h        192.168.12.11   192.168.12.11
```

图3.10 容器进程运行状态信息图

配置关键点

本文档主要介绍了Docker引擎的基本知识，重点放在了Docker各组件之上，详细介绍镜像，容器以及仓库等的具体诠释。

第一节通过对容器与传统的虚拟机对比，传统虚拟机通过中间层将一台或多台独立的机器虚拟应用在硬件之上，而容器则直接应用在操作系统之上的用户空间，来解释Docker容器的优势在于磁盘占用的空间更小，在于可以提供更加优质的服务，服务更多的用户，进而展开对Docker的概念，以及它的目标与八大用途的理解。

第二节通过介绍Docker Client / Daemon，Docker image，Docker container以及Docker repository四大组件的概念与职能，进而理解Docker的组成以及镜像分发过程。之后探究Docker容器的具体技术如namespaces与Cgroups控制组，用来管理与分配资源。

第三节通过H3Cloud OS云管理平台来执行部分的Docker命令，例如查看H3Cloud OS大云平台使用容器进程的运行状态，进入容器操作，在控制节点或计算节点容器中，如果要查看openstack相关进程服务的状态，和手动停止和启动nova服务操作等基本命令。

其实，Docker和虚拟机类似都是一种虚拟化技术，只是Docker比虚拟机能加的轻量级，更快，更容易移植。可以把它理解为集装箱的装成，也就是不同种类商品放在不同集装箱里面，然而一艘货船可以放装不同商品的集装箱，这个就是Docker的思路。将不同应用程序需要不同环境，把程序和它所需环境整体看作一个容器，放在Docker中运行，而Docker却可以放很多不同的容器同时运行。

附件下载： H3Cloud OS产品Docker引擎功能的配置-付青云.doc