Kubernetes应用部署模型解析（部署篇）

发表于 2015-06-12 07:06 | 1654次阅读 | 来源 CSDN | 3 条评论 | 作者 龚永生

Kubernetes 容器

摘要：在介绍部署之前需要了解的原理和概念之后，本文以一个简单的nginx服务来展示了复制器和Service的使用，特别通过对Service的cluster IP和NodePort的分析，使得读者能够了解这个模型中的网络特性。

【编者按】Kubernetes可用来管理Linux容器集群，加速开发和简化运维（即DevOps）。但目前网络上关于Kubernetes的文章介绍性远多于实际使用。本系列文章着眼于实际部署，带您快速掌握Kubernetes。在介绍部署之前需要了解的原理和概念之后，作者在本文中以一个简单的nginx服务来展示了复制器和Service的使用，特别通过对Service的cluster IP和NodePort的分析，使得读者能够了解这个模型中的网络特性。

一个简单的应用

讲了这么多的原理和概念，本章我们就部署一个简单应用来感受一下Kubernetes的部署模型。

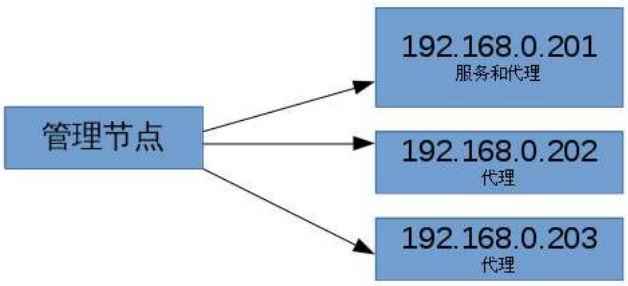
部署Kubernetes集群

在 kubernetes github站点 上有数十种针对各种环境的部署文档，本文选择基于**ubuntu**的集群部署方案。在没有使用本地**docker**镜像的情况下，在部署过程中需要确保能够访问站点gcr.io。

基于 Ubuntu的集群部署方案文档 写得比较详细，按照它的步骤几乎不会出错。在进行真正的部署之前，一定要确保：

1. 所有的节点安装了docker version 1.2+ 和 bridge-utils
2. 如果没有本地的docker registry, 要确保节点能访问互联网gcr.io
3. 确保管理节点能够ssh 访问所有节点。比如ssh gongysh@192.168.0.201 ls

这里我们集群将采用下图显示的结构。我们将在管理节点上运行集群管理命令。我们将有一个服务和代理混合的节点，还有两个纯的代理节点。



首先我们要下载kubernetes的代码到管理节点上：

```
$ git clone https://github.com/GoogleCloudPlatform/kubernetes.git
```

然后进行本地构建：

```
cd kubernetes
./build/run.sh hack/build-do.sh
```

修改config-default.sh定义集群，本文使用的几个关键配置如下：

```
gongysh@fedora20:~/git/kubernetes/cluster/ubuntu$ cat config-default.sh
#!/bin/bash
# Define all your cluster nodes, MASTER node comes first"
# And separated with blank space like <user_1@ip_1> <user_2@ip_2>
<user_3@ip_3>
export nodes="gongysh@192.168.0.201 gongysh@192.168.0.202
gongysh@192.168.0.203"
# Define all your nodes role: a(master) or i(minion) or ai(both master and
minion), must be the order same
export roles=("ai" "i" "i")
# Define minion numbers
export NUM_MINIONS=${NUM_MINIONS:-3}
# define the IP range used for service portal.
# according to rfc 1918 ref: https://tools.ietf.org/html/rfc1918 choose a
private ip range here.
export SERVICE_CLUSTER_IP_RANGE=192.168.3.0/24
# define the IP range used for flannel overlay network, should not
conflict with above SERVICE_CLUSTER_IP_RANGE range
export FLANNEL_NET=172.16.0.0/16
....
```

最后运行集群构建命令：

```
$ cd cluster
$ KUBERNETES_PROVIDER=ubuntu ./kube-up.sh
```

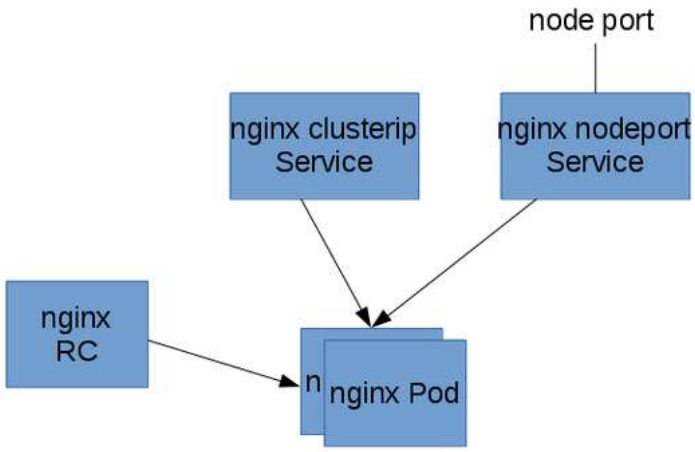
当你看到：

```
Kubernetes cluster is running.  The master is running at:

  http://192.168.0.201

... calling validate-cluster
Found 3 nodes.
     1  NAME            LABELS     STATUS
     2  192.168.0.201   <none>     Ready
     3  192.168.0.202   <none>     Ready
     4  192.168.0.203   <none>     Ready
Validate output:

Cluster validation succeeded
Done, listing cluster services:

Kubernetes master is running at http://192.168.0.201:8080
```

表明集群构建成功。

部署nginx应用

我们以下面的图来安装一个简单的静态内容的nginx应用：



首先，我们用复制器启动一个2个备份的nginx Pod。然后在前面挂Service，一个service只能被集群内部访问，一个能被集群外的节点访问。下面所有的命令都是在管理节点上运行的。

部署**nginx pod** 和复制器

如下表所示：

```
$ cat nginx-rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-controller
spec:
  replicas: 2
  selector:
    name: nginx
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

我们定义了一个nginx pod复制器，复制份数为2，我们使用nginx docker镜像。

执行下面的操作创建nginx pod复制器:

```
$  kubectl -s http://192.168.0.201:8080 create -f nginx-rc.yaml
```

由于kubernetes要去gcr.io下载gcr.io/google_containers/pause镜像，然后下载nginx镜像，所以所创建的Pod需要等待一些时间才能处于running状态。

```
$  kubectl -s http://192.168.0.201:8080 get pods
NAME                    READY     REASON    RESTARTS   AGE
nginx-controller-6zr34  1/1       Running   0          48m
nginx-controller-njlgt  1/1       Running   0          48m
```

我们可以使用describe 命令查看pod所分到的节点:

```
$  $ kubectl -s http://192.168.0.201:8080 describe pod nginx-controller-
6zr34 2>/dev/null | grep Node:
Node:                           192.168.0.203/192.168.0.203
$ kubectl -s http://192.168.0.201:8080 describe pod nginx-controller-njlgt
2>/dev/null | grep Node:
```

```
Node:                           192.168.0.201/192.168.0.201
```

从上表可以看出，这个复制器启动了两个Pod，分别运行在192.168.0.201和203代理节点主机上。

部署节点内部可访问的**nginx service**

Service的type有ClusterIP和NodePort之分，缺省是ClusterIP，这种类型的Service只能在集群内部访问。下表是本文用的配置文件：

```
$ cat nginx-service-clusterip.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-clusterip
spec:
  ports:
    - port: 8001
      targetPort: 80
      protocol: TCP
  selector:
    name: nginx
```

执行下面的命令创建service:

```
$ kubectl -s http://192.168.0.201:8080 create -f ./nginx-service-
clusterip.yaml
services/nginx-service
$ kubectl -s http://192.168.0.201:8080 get service
NAME                    LABELS
SELECTOR     IP(S)            PORT(S)
kubernetes              component=apiserver,provider=kubernetes   <none>
192.168.3.1     443/TCP
nginx-service-clusterip   <none>
name=nginx   192.168.3.91   8001/TCP
```

验证service的可访问性：

上面的输出告诉我们这个Service的Cluster IP是192.168.3.91，端口是8001。下面我们验证这个PortalNet IP的工作情况：

```
$ ssh 192.168.0.202 curl -s 192.168.3.91:8001
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

```
</html>
```

从前面部署复制器的部分我们知道nginx Pod运行在201和203节点上。上面我们特意从202代理节点上访问我们的服务来体现Service Cluster IP在所有集群代理节点的可到达性。

部署外部可访问的**nginx service**

下面我们创建NodePort类型的Service，这种类型的Service在集群外部是可以访问。下表是本文用的配置文件：

```
$ cat nginx-service-nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-nodeport
spec:
  ports:
    - port: 8000
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    name: nginx
```

执行下面的命令创建service:

```
$ kubectl -s http://192.168.0.201:8080 create -f ./nginx-service-
nodeport.yaml
services/nginx-service-nodeport
$ kubectl -s http://192.168.0.201:8080 get service
NAME                    LABELS
SELECTOR     IP(S)          PORT(S)
kubernetes              component=apiserver,provider=kubernetes   <none>
192.168.3.1    443/TCP
nginx-service-clusterip   <none>
name=nginx   192.168.3.91   8001/TCP
nginx-service-nodeport    <none>
name=nginx   192.168.3.84   8000/TCP
```

使用下面的命令获得这个service的节点级别的端口：

```
$ kubectl -s http://192.168.0.201:8080 describe service nginx-service-
nodeport 2>/dev/null | grep NodePort
Type:                   NodePort
NodePort:               <unnamed>       32606/TCP
```

验证service的可访问性：

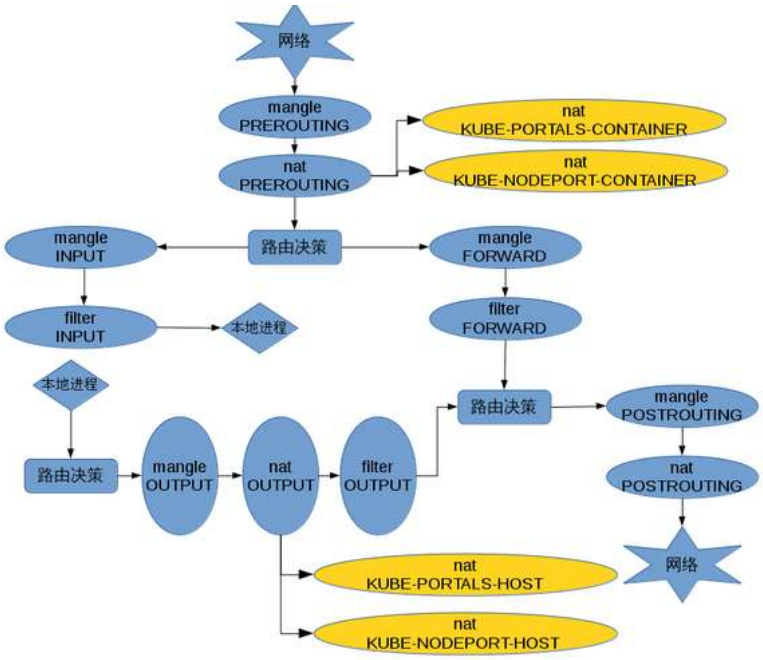
上面的输出告诉我们这个Service的节点级别端口是32606。下面我们验证这个Service的工作情况：

```
$ curl 192.168.0.201:32606
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

```
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

代理节点上的**IP tables**规则解析

下面的图是IPTables中流量经过的table和chain。



可以看出，Kubernetes在nat表中插入了下面四条chain：

1. KUBE-PORTALS-CONTAINER

这个chain主要是处理所有service对象的cluster IP和port到kube-proxy本地端口的映射。比如下面规则：

```
-A KUBE-PORTALS-CONTAINER -d 192.168.3.84/32 -p tcp -m comment --comment
"default/nginx-service-nodeport:" -m tcp --dport 8000 -j REDIRECT --to-
ports 43981
```

就是为nginx-service-nodeport服务的Cluster IP准备的。其中192.168.3.84/32是该服务获得的Cluster IP，端口8000是其在定义文件中指定的spec.ports.port。43981则是kube-proxy为这个service分配的本地端口。规则的意思是到192.168.3.84:8000的流量重定向到43981。

2. KUBE-NODEPORT-CONTAINER

这条chain上则串连着类型为NodePort的service的NodePort规则。比如下面规则：

```
-A KUBE-NODEPORT-CONTAINER -p tcp -m comment --comment "default/nginx-
service-nodeport:" -m tcp --dport 32606 -j REDIRECT --to-ports 43981
```

就是为nginx-service-nodeport服务的NodePort 32606准备的。意思是访问本地32606端口的流量重新定向到43981，后者是kube-proxy为这个service分配的本地端口。

3. KUBE-PORTALS-HOST

这条chain上也关联着各个service的Cluster IP和Port的规则，比如：

```
-A KUBE-PORTALS-HOST -d 192.168.3.84/32 -p tcp -m comment --comment
"default/nginx-service-nodeport:" -m tcp --dport 8000 -j DNAT --to-
destination 192.168.0.201:43981
```

这条规则是和KUBE-PORTALS-CONTAINER类似的，只不过流量来自于本地进程。

4. KUBE-NODEPORT-HOST

这条chain上则关联着类型为NodePort的service的NodePort规则。比如下面规则：

```
-A KUBE-NODEPORT-HOST -p tcp -m comment --comment "default/nginx-service-
nodeport:" -m tcp --dport 30975 -j DNAT --to-destination
192.168.0.201:43981
```

这条规则是和KUBE-NODEPORT-CONTAINER类似的，只不过流量来自于本地进程。

总结

笔者认为Docker已经不是仅代表容器本身，而是一组以应用部署为中心的技术，产品和最佳实践生态系统。Kubernetes以其出身，文档的成熟度，社区的支持在这个生态系统中表现得比较突出。在部署Kubernetes时，我们首先要理解Kubernetes的组件结构，它们有哪些角色，各个角色的作用是什么和它们之接的通信。在应用部署时，了解Kubernetes的应用模型是非常重要的。笔者认为复制器和Service的概念是Kubernetes模型的核心，复制器和Service共同完成了应用的高可用性要求。最后本文以一个简单的nginx服务来展示了复制器和Service的使用，特别通过对Service的cluster IP和NodePort的分析，使得读者能够了解这个模型中的网络特性。

最后就是容器技术的选型，本文使用Docker作为容器，其实Kubernetes也支持CoreOS的rkt容器。kubelet的参数--container_runtime用于选择使用的容器技术。（责编/周建丁）

作者简介：龚永生，九州云架构师。多年Linux系统开发，J2EE产品和云计算相关技术研发经验。目前活跃在OpenStack社区的各个项目上，主要技术方向是虚拟网络项目Neutron，是Neutron项目早期的主要贡献者之一。

fanax 2015-06-30 17:08

KUBE-PORTALS-CONTAINER

这部分是clusterIP，内部访问的service，

你的描述中“192.168.3.84/32是该服务获得的Cluster IP，端口8000”，为什么是192.168.3.84，cluster IP按照前面的分配好像应该是192.168.3.91:8001？

同理KUBE-PORTALS-HOST中也是这种情况，不明白？

请解释下，谢谢

回复

qq690388648 2015-06-26 20:12

谢谢撸主的分享，很有用！

回复

FreyrLin 2015-06-12 11:30

IP Tables 部分的分析非常精彩！赞！

回复

共1页 首页 上一页 1 下一页 末页