说明

本文从0搭建kubernetes集群，并阐述每个步骤的结果验证办法，以及每步骤可能出现的问题以及对应解决方法。本人笨拙，若有不对，情指出。

kubernetes简述

在开始搭建kubernetes集群之前，简短的介绍下什么是kubernetes，以及我们用它来做什么。

如果要搭建一个完善的微服务系统

- 首先，这个系统需要一套能够很好的管理我们各个服务的服务，这个应该包含了"run","stop","scale","roll update"等功能。
- 其次，这个系统还需要一套能够很好的监控服务的功能，例如服务的健康状态，CPU，内存的状态等。
- 再次，这个系统需要一套完善的日志收集、查询的功能，线上的所有问及都能够通过查询日志来进行分析，所以一套完善的日志系统是必不可少的。
- 这三个基础之上，我们可能还需要更多高级的服务，例如自动扩容，这些需要根据系统的需求去搭建。

而kubernetes是这么一个工具，它能够让我们很方便的搭建出一个分布式系统，同时它提供了很多的基础工具或服务，能够让我们比较轻松的管理，监控这个集群，并提供了非常方便的日志查询系统（这些属于额外插件），让我们能够定位线上问题。在服务运行结构上，kubernetes基于容器（例如docker，rkt）,并在此之上抽象了一层pod、service等机制。具体可查看官方文档《官方文档》

本文搭建基础

本例子搭建在AWS的EC2上，因不想与AWS耦合太紧密，所以只用了EC2做虚拟机。EC2共4台，一台etcd服务器，一台做master，两台做node.

作用	内网IP	外网IP
etcd	etcd_inner_ip	etcd_outer_ip
master	master_inner_ip	master_outer_ip
node1	node1_inner_ip	node1_outer_ip
node2	node2_inner_ip	node2_outer_ip

- 系统：coreos

一个专为容器技术打造的os

- docker: 1.10

容器运行环境

- kubernetes: 1.4.0 (1.3.7,1.3.6也可以)
- etcd: 3.1.0

一个稳定的key、value存储服务

- flannel: 0.6.2

用于kubernetes的覆盖网络

搭建步骤

1, 搭建ETCD服务

kubernetes采用etcd作为配置存储服务，所以需要先搭建etcd服务

下载etcd. 这里根据自己的系统下载对应的，这里是coreos 64位，所以下载linux-amd64

```
wget https://github.com/coreos/etcd/releases/download/v3.0.10/etcd
```

解压

```
tar zxvf etcd-v3.0.10-linux-amd64.tar.gz
```

转到etcd目录

```
cd etcd-v3.0.10-linux-amd64
```

执行

```
./etcd --listen-client-urls http://etcd_inner_ip:2379,http://etcd_
```

执行上面那条命令的意思是：运行单机版本的一个etcd节点，并通过2379和4001对外提供服务

检验运行结果：

```
./etcdctl --endpoint http://etcd_inner_ip:4001 set testkey testval
./etcdctl --endpoint http://etcd_inner_ip:4001  get testkey
```

执行上面两句的结果应该都会输出testvalue，此时则表示etcd单节点集群已经正常运行起来了，并且运行的日志保存在文件： `/var/log/etcd.log` 下面。

可能错误:

```
Error:  client: etcd cluster is unavailable or misconfigured
error #0: dial tcp etcd_inner_ip:2379: getsockopt: connection refu
error #1: dial tcp etcd_inner_ip:4001: getsockopt: connection refu
```

上面错误表示etcd服务并没有运行起来，可以参照 `/var/log/etcd.log` 的内容来进行修改。

2, 搭建Master

在master上面运行的部件共包涵下面几个： `API-Server` `Controller-Manager` `Scheduler` `kube-dns`

进入master机器，并下载kubernetes:

```
wget https://github.com/kubernetes/kubernetes/releases/download/v1
```

默认下载到 `/home/core` 目录下面。 解压:

```
tar zxvf kubernetes.tar.gz
cd kubernetes/server
tar zxvf kubernetes-server-linux-amd64.tar.gz
```

解释： `kubernetes.tar.gz` 这个是kubernetes的所有资源集合，`kubernetes-server-linux-amd64.tar.gz` 是针对不同平台的运行包集合，这里采用的是linux amd64.

添加至 `PATH` 路径

```
export PATH=/home/core/kubernetes/server/kubernetes/server/bin/:$P
```

2.1, 启动 kube-apiserver

```
kube-apiserver --logtostderr=true --v=0 --etcd-servers=http://etcd
```

以上命令启动了 kube-apiserver ， 并使用 http://etcd_inner_ip:4001 作为etcd配置服务, 同时监听的端口是 8080

检查方法:

```
curl http://master_outer_ip:8080
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1alpha1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v2alpha1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1alpha1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/policy",
    "/apis/policy/v1alpha1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1alpha1",
```

```
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/ping",
    "/logs",
    "/metrics",
    "/swaggerapi/",
    "/ui/",
    "/version"
  ]
}
```

能够看到相应的输出。

或者在master上执行：

```
kubectl version

Client Version: version.Info{Major:"1", Minor:"4", GitVersion:"v1.
Server Version: version.Info{Major:"1", Minor:"4", GitVersion:"v1.
```

如果能够同时看到client和api server的版本，则算是启动成功。

2.2, 启动Controller-Manager和Scheduler

```
kube-controller-manager --logtostderr=true --v=0 --master=http://m

kube-scheduler --logtostderr=true --v=0 --master=http://master_inn
```

检验：

```
kubectl get componentstatus
NAME                 STATUS    MESSAGE              ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

2.3, 启动kube-dns

```
kube-dns --kube-master-url=http://master_inner_ip:8080 --logtostde
```

至此kubernetes的master节点就搭建好了

3, 搭建node

node是我们应用程序的实际的运行地方，在node上，需要运行三个东西

```
kubelet
kube-proxy
docker
```

kubelet是master管理node的入口，抑或说是其代理，通过kubelet，能够管理node上运行的pods等。 kube-proxy则是服务代理，用来代理service。 docker 则是真正的容器运行的环境。

参照master，下载kubernetes，解压并添加到PATH路径.

3.1, 启动kubelet和kube-proxy

```
kubelet --logtostderr=true --v=0 --address=0.0.0.0 --api-servers=h
kube-proxy --logtostderr=true --v=0 --master=http://master_inner_i
```

检查node是否正确启动：

```
kubectl get nodes
NAME            STATUS    AGE
node1_name      Ready     0d
```

```
node2_name      Ready     0d
```

至此所有的node均启动完毕。

4, 搭建kubernetes的覆盖网络

由于docker本身的网络并不支持kubernetes的网络模型，所以我们需要在docker的网络模型之上搭建一个覆盖网络，以kubernetes的网络模型能够正常工作。这里我们使用的是flannel 0.6.2版本

下载flannel并解压:

```
cd /home/core/flannel
wget https://github.com/coreos/flannel/releases/download/v0.6.2/fl
tar zxvf flannel-v0.6.2-linux-amd64.tar.gz
```

由于flannel使用etcd作为配置服务，所以在运行flannel之前，我们需要在etcd中添加我们的网络配置：

```
etcdctl set /coreos.com/network/config '{"Network": "10.0.0.0/16"}
```

运行flannel:

```
./flanneld -etcd-endpoints=http://etcd_inner_ip:4001 >> /var/log/f
```

然后我们需要替换现有的docker网络：

```
iptables -t nat -F

ifconfig docker0 down

brctl delbr docker0
```

```
source /run/flannel/subnet.env

systemctl stop docker

docker daemon --bip=${FLANNEL_SUBNET} --mtu=${FLANNEL_MTU} >> /var
```

以上几步的作用是：先关闭docker网络，然后重启docker，并指定使用flannel作为其网络配置。

覆盖网络需要安装在所有节点上，包括master

至此一个基本的kubernetes集群就搭建好了

下篇介绍如何在此之上搭建dashboard