# 主机角色

192.168.100.2 master.danran.com master,etcd,private

192.168.100.3 node1.danran.com node1

192.168.100.4 node2.danran.com node2

## 配置各主机hosts文件

master，node1，node2各节点主机都需配置hosts解析文件

[root@master ~]# cat /etc/hosts

192.168.100.2 master.danran.com master etcd1 registry

192.168.100.3 node1.danram.com

192.168.100.4 node2.danran.com

## 各节点主机配置时间同步

因为在此各系统都可连接外部网络，故没有配置内网的ntp服务器，直接使用的外网的时间同步，也可配置内网的ntp服务端

时间同步可使用ntp或chronyd，在本公司redhat系统都使用ntp配置时间同步

[root@master ~]# yum -y install ntp

[root@master ~]# systemctl start ntpd

[root@master ~]# systemctl enable ntpd

各节点关闭firewalld.server及iptables.server，并确保其服务不会自动开启

[root@master ~]# systemctl disable firewalld

[root@master ~]# systemctl stop firewalld

[root@master ~]# systemctl disable iptables

[root@master ~]# systemctl stop iptables

[root@master ~]# systemctl status firewalld

● firewalld.service - firewalld - dynamic firewall daemon

Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)

Active: inactive (dead)

Docs: man:firewalld(1)

## 配置private registry节点

在本环境中，由于资源有限，故将registry配置在master节点中

安装registry程序包docker-distribution

[root@master ~]# yum -y install docker-distribution

安装nginx做private registry的代理设置

[root@master ~]# yum -y install nginx

配置registry文件，配置http为本机地址，由于使用的nginx反代，故可使用本机的私有地址，rootdirectory指定registry存放存储目录路径

[root@master ~]# cat /etc/docker-distribution/registry/config.yml

version: 0.1

log:

fields:

service: registry

storage:

cache:

layerinfo: inmemory

filesystem:

rootdirectory: /var/lib/registry

http:

addr: 127.0.0.1:5000

启动registry

[root@master ~]# systemctl start docker-distribution

[root@master ~]# systemctl enable docker-distribution

Created symlink from /etc/systemd/system/multi-user.target.wants/docker-distribution.service to /usr/lib/systemd/system/docker-distribution.service.

确认registry服务的5000端口已监听

[root@master ~]# ss -ntl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 127.0.0.1:25 \*:\*

LISTEN 0 128 127.0.0.1:5000 \*:\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 100 ::1:25 :::\*

配置nginx反向代理

[root@master ~]# vim /etc/nginx/conf.d/registry.conf

server {

listen 80;

server\_name registry.danran.com;

client\_max\_body\_size 0; \\不限制client上传时使用的body大小

location / {

proxy\_pass http://127.0.0.1:5000; \\指定代理的registry地址

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

proxy\_redirect off;

proxy\_buffering off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

# auth\_basic "Docker Registry Service"; \\启用registry认证功能，在此不配做registry的认证

# auth\_basic\_user\_file "/etc/nginx/.ngxpasswd";

}

}

启动nginx服务并确认80端口已监听

[root@master ~]# systemctl start nginx

[root@master ~]# systemctl enable nginx

Created symlink from /etc/systemd/system/multi-user.target.wants/nginx.service to /usr/lib/systemd/system/nginx.service.

[root@master ~]# ss -ntl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 \*:80 \*:\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 127.0.0.1:25 \*:\*

LISTEN 0 128 127.0.0.1:5000 \*:\*

LISTEN 0 128 :::80 :::\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 100 ::1:25 :::\*

# 部署etcd

在本次的测试环境架构中，etcd与master在同一节点上，故在此环境中的master上安装etcd组件

安装etcd

[root@master ~]# yum install etcd -y

查看etcd包中的相关文件，/usr/bin/etcd为主程序文件， /usr/bin/etcdctl为客户端程序文件

[root@master ~]# rpm -ql etcd | less

/etc/etcd

/etc/etcd/etcd.conf

/usr/bin/etcd

/usr/bin/etcdctl

/usr/lib/systemd/system/etcd.service

/usr/share/doc/etcd-3.2.15

/usr/share/doc/etcd-3.2.15/CONTRIBUTING.md

/usr/share/doc/etcd-3.2.15/README.md

/usr/share/doc/etcd-3.2.15/ROADMAP.md

/usr/share/doc/etcd-3.2.15/glide.lock

/usr/share/licenses/etcd-3.2.15

/usr/share/licenses/etcd-3.2.15/LICENSE

/var/lib/etcd

修改/etc/etcd/etcd.conf配置文件，修改提供etcd服务的监听IP及端口，在主机名解析中，etcd1及master都可解析

在/etc/etcd/etcd.conf文件中，[Member]字段定义集群中的成员关系，[Clustering]定义集群的属性，[proxy]定义集群的代理

[Security]安全相关的属性定义，[Logging]登录相关的定义，[Profiling]性能本身相关的定义，[Auth]认证配置

[root@master ~]# vim /etc/etcd/etcd.conf

#[Member]

ETCD\_NAME="etcd1" \\当前节点被识别时使用的name，默认为default

ETCD\_DATA\_DIR="/var/lib/etcd/etcd1.etcd" \\保存当前节点数据的目录

ETCD\_LISTEN\_CLIENT\_URLS="http://192.168.100.2:2379" \\监听接收客户端访问的入口地址及端口，修改IP为etcd1的2379端口

#[Clustering]

ETCD\_ADVERTISE\_CLIENT\_URLS="hhttp://192.168.100.2:2379" \\etcd集群环境中，当前节点通过etcd1地址的2379端口向外通信通告集群或成员身份

ETCD\_INITIAL\_CLUSTER="etcd1=http://etcd1:2380" \\当只有一个etcd单节点时，启用该选项表示只有单节点，2380为etcd集群内部通信协商的端口

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://etcd1:2380"

上述的各选项意义如下

ETCD\_NAME：当前节点的名称

ETCD\_LISTEN\_CLIENT\_URLS：当前节点用于与etcd客户端通信的URL列表，彼此间以逗号分隔；

ETCD\_ADVERTISE\_CLIENT\_URLS：当前节点向外通告的其用于与客户端通信的URL列表

ETCD\_INITIAL\_CLUSTER：用于etcd启动时初始化的集群配置；构建一个多节点的etcd集群时，需要让集群成员间通信的端口监听于对外通信的IP地址上，

默认为localhost；以下两个选项可用于此配置

ETCD\_LISTEN\_PEER\_USER：list of URLs to listen on for peer traffic

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：list of this member's peer URLs to advertise to the rest of the cluster

启动etcd服务

[root@master ~]# systemctl start etcd.service

[root@master ~]# systemctl enable etcd.service

Created symlink from /etc/systemd/system/multi-user.target.wants/etcd.service to /usr/lib/systemd/system/etcd.service.

2379及2380端口都已监听

[root@master ~]# ss -ntl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 192.168.100.2:2379 \*:\*

LISTEN 0 128 127.0.0.1:2380 \*:\*

LISTEN 0 128 \*:80 \*:\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 127.0.0.1:25 \*:\*

LISTEN 0 128 127.0.0.1:5000 \*:\*

LISTEN 0 128 :::80 :::\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 100 ::1:25 :::\*

查看etcd集群成员列表

[root@master ~]# etcdctl --help

etcdctl各子命令及功能

COMMANDS:

backup backup an etcd directory

cluster-health check the health of the etcd cluster

mk make a new key with a given value

mkdir make a new directory

rm remove a key or a directory

rmdir removes the key if it is an empty directory or a key-value pair

get retrieve the value of a key

ls retrieve a directory

set set the value of a key

setdir create a new directory or update an existing directory TTL

update update an existing key with a given value

updatedir update an existing directory

watch watch a key for changes

exec-watch watch a key for changes and exec an executable

member member add, remove and list subcommands

user user add, grant and revoke subcommands

role role add, grant and revoke subcommands

auth overall auth controls

help, h Shows a list of commands or help for one command

[root@master ~]# etcdctl -C http://master:2379 member list

ade526d28b1f92f7: name=etcd1 peerURLs=http://etcd1:2380 clientURLs=http://192.168.100.2:2379 isLeader=true

[root@master ~]# etcdctl -C http://master:2379 cluster-health

member ade526d28b1f92f7 is healthy: got healthy result from http://192.168.100.2:2379

cluster is healthy

etcd集群的启动方式支持三种机制

static(静态)：集群成员固定，初始化前直接指定成员主机URL即可；

etcd Discovery(etcd 发现)：集群成员地址不固定，此时可利用已经存在etcd集群来发现并创建一个新的集群，发现方式是etcd标识；

DNS Discovery(DNS 发现)：集群成员地址不固定，此时可利用已经存在etcd集群来发现并创建一个新的集群，发现方式是DNS SRV records；

# 部署kubernetes master节点

## Kubernetes master节点的配置解析

Kubernetes master包含三个核心组件：API Server、Scheduler和Controller Manager，他们分别对应于一个要启动的服务，kube-apiserver，kube-scheduler和kube-controller Manager

* 在master节点上安装kubernetes-master程序

Kubernetes master的配置文件位于/etc/kubernetes目录下，主要修改的文件有apiserver和config

配置apiserver，将各选项设定为符合环境的值

# 配置kube-apiserver监听的本机地址，配置多节点集群时，应该使用对外通信的地址

KUBE\_API\_ADDRESS=“—insecure-bind-address=0.0.0.0”

# 配置kube-apiserver监听的端口

KUBE\_API\_PORT=“—port=8080”

# Port minions listen on

# KUBELET\_PORT=“—kubelet-port=10250”

# 配置kubernetes集群要使用的etcd集群的成员列表

KUBE\_ETCD\_SERVERS=”—etcd-servers=http://etcd1:2379”

# 集群内各service可使用的地址范围

KUBE\_SERVICE\_ADDRESSES=”—service-cluster-ip-range=10.254.0.0/16”

# 默认的管理控制策略(若HTTP不加密则不能使用ServiceAccount认证参数)

KUBE\_ADMISSION\_CONTROL="—admission-control=NamespaceLifecycle，NamespaceExists,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQuota"

# 自定义的其他配置

KUBE\_API\_ARGS=""

* 配置公共配置文件config，定义master访问的接口

# 配置错误日志记录方式

KUBE\_LOGTOSTDERR="--logtostderr=true"

# 指定日志级别，0表示debug

KUBE\_LOG\_LEVEL="--v=0"

# 配置controller-manager，scheduler和proxy访问master服务的接口

KUBE\_MASTER="--master=http://master:8080"

# 设定本集群中是否允许运行特权docker容器

KUBE\_ALLOW\_PRIV="--allow-privileged=false"

## 实施部署kubernetes master

master节点安装kubernetes-master

[root@master ~]# yum -y install kubernetes-master

修改config文件，配置访问接口IP为master节点的8080端口

[root@master ~]# vim /etc/kubernetes/config

KUBE\_MASTER="--master=http://master:8080"

修改apiserver文件，配置监听本机的所有地址，etcd集群的主机列表为etcd1

[root@master ~]# vim /etc/kubernetes/apiserver

KUBE\_API\_ADDRESS="--insecure-bind-address=0.0.0.0"

KUBE\_ETCD\_SERVERS="--etcd-servers=http://etcd1:2379"

启动master节点，需将kube-apiserver、kube-scheduler和kube-controller-manager同时启用

在此次环境中，kube-apiserver、kube-scheduler和kube-controller-manager均运行于master节点，故可以直接启用这些程序

[root@master ~]# systemctl start kube-apiserver.service kube-scheduler.service kube-controller-manager.service

[root@master ~]# systemctl enable kube-apiserver.service kube-scheduler.service kube-controller-manager.service

Created symlink from /etc/systemd/system/multi-user.target.wants/kube-apiserver.service to /usr/lib/systemd/system/kube-apiserver.service.

Created symlink from /etc/systemd/system/multi-user.target.wants/kube-scheduler.service to /usr/lib/systemd/system/kube-scheduler.service.

Created symlink from /etc/systemd/system/multi-user.target.wants/kube-controller-manager.service to /usr/lib/systemd/system/kube-controller-manager.service.

查看监听端口，8080、6443、10251/10252等端口已监听

[root@master ~]# ss -ntl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 192.168.100.2:2379 \*:\*

LISTEN 0 128 127.0.0.1:2380 \*:\*

LISTEN 0 128 \*:80 \*:\*

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 127.0.0.1:25 \*:\*

LISTEN 0 128 127.0.0.1:5000 \*:\*

LISTEN 0 128 :::6443 :::\*

LISTEN 0 128 :::10251 :::\*

LISTEN 0 128 :::10252 :::\*

LISTEN 0 128 :::8080 :::\*

LISTEN 0 128 :::80 :::\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 100 ::1:25 :::\*

# 部署kubernetes集群的各Node

Kubernetes集群的各节点用于运行容器，因此部署kubernetes的相关组件之前需要事先提供容器相关的环境(Container runtime)，如docker，以下操作需要在各个node主机上分别操作。

确保Selinux是disabled状态

[root@node2 ~]# sed -i 's/^SELINUX=.\*/SELINUX=disabled/' /etc/selinux/config

[root@master ~]# getenforce

Disabled

## 部署并启动docker

Docker位于Extras仓库中，配置好yum仓库可直接yum安装

\*rhsm\*程序包用来生成redhat registry证书目录，从而可访问registry.access.redhat.com

yum –y install docker \*rhsm\*

因为使用国内地址，为了避免访问docker官方仓库速率慢的问题，故在此配置加速器，再此已配置aliyun的加速器为例

在已有阿里云账号的前提下，生成专属的加速器地址，链接如下

<https://cr.console.aliyun.com/#/accelerator>

选择相应的类别的系统配置docker的加速器

[root@master ~]# cd /etc/docker

[root@master docker]# vim daemon.json

{

"registry-mirrors": ["https://qh1x6x3z.mirror.aliyuncs.com"]

}

[root@master ~]# systemctl daemon-reload

添加额外的私有镜像仓库为registry.danran.com

[root@master ~]# vim /etc/sysconfig/docker

ADD\_REGISTRY="--add-registry registry.danran.com"

INSECURE\_REGISTRY="--insecure-registry registry.danran.com"

配置好docker的自定义选项后即可启动服务，如设置私有仓库的选项

[root@master ~]# systemctl start docker

[root@master ~]# systemctl enable docker

Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

测试docker

[root@master ~]# docker info

## 部署并启动kubelet和kubernetes Network Proxy

Kubernetes集群的各node分别包含两个重要组件：Kubelet和Kubernetes Network Proxy，分别对应一个服务kubelet和kube-proxy

* 配置kubelet

# kubelet服务监听的IP地址(set to 0.0.0.0 or “” for all interfaces)

KUBELET\_ADDRESS="--address=0.0.0.0"

# kubelet监听的端口

KUBELET\_PORT="--port=10250"

# 当前节点的主机名，留空表示使用本机的真实主机名

KUBELET\_HOSTNAME="--hostname-override=node1"

# API server的访问接口

KUBELET\_API\_SERVER="--api-servers=http://master:8080"

# Pod infrastructure container

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest"

# 按需定义的其他选项

KUBELT\_ARGS=””

* 配置公共配置文件config，定义master访问接口

# 配置错误日志记录方式

KUBE\_LOGTOSTDERR="--logtostderr=true"

# 指定日志级别，0表示dubug

KUBE\_LOG\_LEVEL="--v=0"

# 设定本集群中是否允许运行特权docker容器

KUBE\_ALLOW\_PRIV="--allow-privileged=false"

# 配置controller-manager，scheduler和proxy访问master服务的接口

KUBE\_MASTER="--master=http://master:8080"

Node节点安装kubernetes-node程序(master节点不需安装)

Kubernetes-nodes位于Extras仓库中

[root@node1 ~]# yum -y install kubernetes-node

修改kubernetes-node的相关配置

[root@node1 ~]# vim /etc/kubernetes/config

KUBE\_MASTER="--master=http://master:8080" 修改master的监听地址及端口

[root@node1 ~]# vim /etc/kubernetes/kubelet

KUBELET\_ADDRESS="--address=0.0.0.0" 修改为本机监听的所有地址

KUBELET\_PORT="--port=10250" 本机监听的端口，默认为10250端口

KUBELET\_HOSTNAME="--hostname-override=node1" 设置当前主机的主机名

KUBELET\_API\_SERVER="--api-servers=http://master:8080" 设置API server的监听地址

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest" 基础架构容器镜像

将kubernetes-node的其他节点上同样修改/etc/kubernetes/config及/etc/kubernetes/ kubelet文件，注意/etc/kubernetes/kubelet文件中的KUBELET\_HOSTNAME ="--hostname-override=node1"要不一致

在node节点启动kubelet及kube-proxy服务

[root@node1 ~]# systemctl start kubelet.service kube-proxy.service

[root@node1 ~]# systemctl enable kubelet.service kube-proxy.service

Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /usr/lib/systemd/system/kubelet.service.

Created symlink from /etc/systemd/system/multi-user.target.wants/kube-proxy.service to /usr/lib/systemd/system/kube-proxy.service.

本机的10250端口已监听

[root@node1 ~]# ss -ntl

State Recv-Q Send-Q Local Address:Port Peer Address:Port

LISTEN 0 128 \*:22 \*:\*

LISTEN 0 100 127.0.0.1:25 \*:\*

LISTEN 0 128 127.0.0.1:10248 \*:\*

LISTEN 0 128 :::10250 :::\*

LISTEN 0 128 :::10255 :::\*

LISTEN 0 128 :::22 :::\*

LISTEN 0 100 ::1:25 :::\*

LISTEN 0 128 :::4194 :::\*

# kubectl集群测试

1. Kubelet概述

Kubernetes的API server是集群服务的访问入口，kubelet通过API与集群交互完成各种管理工具，因此，kubelet需要在管理集群之前知悉集群的访问入口，并拥有访问凭证(如果集群配置了验证和授权机制)。这些信息可以命令行选项给出，也可通过配置文件kubeconfig file设定。

查看集群中的节点信息

[root@node2 ~]# kubectl -s http://master:8080 get nodes

NAME STATUS AGE

node1 Ready 23m

node2 Ready 1m

查看集群运行状态

[root@node2 ~]# kubectl -s http://master:8080 cluster-info

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

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

[root@master ~]# kubectl cluster-info

Kubernetes master is running at http://localhost:8080

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

1. Kubectl命令概述

[root@node2 ~]# kubectl

kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/kubernetes/kubernetes.

Basic Commands (Beginner):

create Create a resource by filename or stdin

expose Take a replication controller, service, deployment or pod and

expose it as a new Kubernetes Service

run Run a particular image on the cluster

set Set specific features on objects

Basic Commands (Intermediate):

get Display one or many resources

explain Documentation of resources

edit Edit a resource on the server

delete Delete resources by filenames, stdin, resources and names, or

by resources and label selector

Deploy Commands:

rollout Manage a deployment rollout

rolling-update Perform a rolling update of the given ReplicationController

scale Set a new size for a Deployment, ReplicaSet, Replication

Controller, or Job

autoscale Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:

certificate Modify certificate resources.

cluster-info Display cluster info

top Display Resource (CPU/Memory/Storage) usage

cordon Mark node as unschedulable

uncordon Mark node as schedulable

drain Drain node in preparation for maintenance

taint Update the taints on one or more nodes

Troubleshooting and Debugging Commands:

describe Show details of a specific resource or group of resources

logs Print the logs for a container in a pod

attach Attach to a running container

exec Execute a command in a container

port-forward Forward one or more local ports to a pod

proxy Run a proxy to the Kubernetes API server

cp Copy files and directories to and from containers.

Advanced Commands:

apply Apply a configuration to a resource by filename or stdin

patch Update field(s) of a resource using strategic merge patch

replace Replace a resource by filename or stdin

convert Convert config files between different API versions

Settings Commands:

label Update the labels on a resource

annotate Update the annotations on a resource

completion Output shell completion code for the given shell (bash or zsh)

Other Commands:

api-versions Print the supported API versions on the server, in the form of

"group/version"

config Modify kubeconfig files

help Help about any command

version Print the client and server version information

Use "kubectl <command> --help" for more information about a given command.

Use "kubectl options" for a list of global command-line options (applies to all commands).

1. 测试于kubernetes中运行容器

先在node1及node2节点上pull到本地busybox镜像

[root@node1 ~]# docker pull registry.access.redhat.com/rhel7/pod-infrastructure:latest

[root@node1 ~]# docker pull busybox

[root@node2~]# docker pull busybox

Using default tag: latest

Trying to pull repository registry.danran.com/busybox ...

Pulling repository registry.danran.com/busybox

Trying to pull repository docker.io/library/busybox ...

latest: Pulling from docker.io/library/busybox

f70adabe43c0: Pull complete

Digest: sha256:58ac43b2cc92c687a32c8be6278e50a063579655fe3090125dcb2af0ff9e1a64

Status: Downloaded newer image for docker.io/busybox:latest

[root@node1 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/busybox latest 8ac48589692a 12 days ago 1.15 MB

docker.io/redis 3.2 b05c3d76c8b3 4 weeks ago 99.7 MB

registry.access.redhat.com/rhel7/pod-infrastructure latest 99965fb98423 6 months ago 209 MB

Kubectl run命令能够创建并运行一个指定的可复制的镜像，他会自动创建一个deployment或者job来管理创建的容器。例如，创建一个交互登录的基于busybox镜像的容器bbox。

[root@node1 ~]# kubectl -s http://master:8080 run bbox --image=docker.io/busybox:la

test --dry-run

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

bbox 1 0 0 0 <unknown>

[root@node1 ~]# kubectl -s http://master:8080 run bbox -i -t --image=docker.io/busyb

ox:latest

查看集群中的“部署”(deployment)资源

[root@node2 ~]# kubectl -s http://master:8080 get deployments

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

bbox 1 0 0 0 2m

[root@node2 ~]# kubectl -s http://master:8080 get pods

以yaml格式查看启动的Pod的详细格式的信息

[root@master ~]# cat test.yml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

# Unique key of the Deployment instance

name: deployment-example

spec:

# 3 Pods should exist at all times.

replicas: 2

template:

metadata:

labels:

# Apply this label to pods and default

# the Deployment label selector to this value

app: nginx

spec:

containers:

- name: nginx

# Run this image

image: nginx:1.12

[root@master ~]# kubectl -s http://master:8080 create -f ./test.yml --recorddocker.io/nginx

[root@node2 ~]# kubectl -s http://master:8080 get –o yaml pods

删除集群中的资源

Kubectl delete可用于删除集群中的资源

[root@node2 ~]# kubectl -s http://master:8080 delete deployments --all

deployment "bbox" deleted

或

[root@node2 ~]# kubectl -s http://master:8080 delete deployment bbox

[root@node2 ~]# kubectl -s http://master:8080 get deployments

No resources found.

# Kubernetes的网络功能

Kubernetes需要其原子组件Pod彼此间可直接通信，无论他们位于集群中哪台主机之上。因此，每个Pod必须拥有一个可路由的IP地址，相互间无须NAT或端口映射即可进行数据交换。这有别于Docker的网络模型，使用默认网络模型的两个Docker主机无法满足此类需求。同时，同一个Pod内部的各容器间可直接通过loopback接口进行通信，例如同一pod内部的tomcat容器可直接经由lo接口访问mariadb server容器上的数据。这种模型中，Pod类似于laaS架构中的VM，同一VM上运行的多个application极容易迁移至Pod的容器上，且各Pod之间的通信也类似于跟VM之间的通信。

Kubernetes PAUSE container

Kubernetes的每个Pod中均存在一个名为“PAUSE”的容器，他持有Pod的IP地址等网络属性，并将网络名称空间共享给当前Pod中的其他容器，这些容器是Pod的joined网络类型的容器。

在Pod内部，每个容器以“映射容器(mapped container)”模型连接至PAUSE容器，通信过程中，目标地址为Pod IP地址的流量首先到达PAUSE容器，而后再由PAUSE容器转发至后端的一个或多个容器，此种模型中，PAUSE容器也被称之为“目标容器(target container)”，而后其他容器被称之为“源容器(source container)”。

概括来讲，Kubernetes的网络模型中存在四种通信模式

* + 同一Pod内部的容器间通信：籍由同一网络名称空间中的lo接口进行；
  + Pod间的通信：同一kubernetes集群中的所有Pod拥有同一IP网络中的地址，Pod彼此间可通过IP地址直接通信；
  + Pod与Servcie间的通信：Service的核心是VIP，它能够直接接受客户端的请求，并将请求代理至与此服务器关联的各Pod；其过程是到达Service的请求被发往各Node主机的Kube-proxy进程，并由其路由至相关的Pod；
  + 来自集群外部的访问：集群中的节点拥有公网IP(Externel IP)地址时，Kubernetes可将服务”暴露”至此些IP地址；到此些IP地址上的服务端口的访问流量会被路由至Service的端点，而后由kube-proxy转发至目标Pod；Externel IP非由Kubernetes管理，管理员需要完成此工作。

Kubernetes网络解决方案

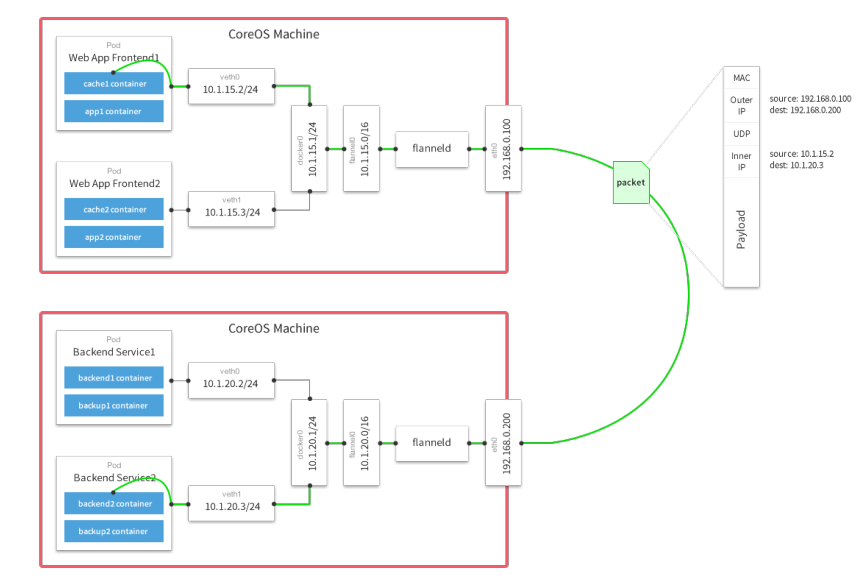
Kubernetes虽然设计了网络模型，但它自己并不实现此功能，而是借助于第三方解决方案，如Flannel、Calico、Weave、OpenContrail、Cisco Contiv和OpenVSwitch等等。

叠加网络flannel概述

Flannel是由CoreOS项目维护的虚拟网络解决方案，它是能够满足kubernetes网络模型需求的极简洁实现方式，但经由充分测试且可用于生产环境，部署代价极低。

运行时，Flannel需要首先确定自己使用的网络地址，例如“10.1.0.0/16”，而后为集群中的每个Docker’主机分配一个子网，例如“10.1.15.0/24”或“10.1.20.0/24”等，Dcoker主机将此子网地址作为本机上运行的容器的可用地址池。

同一Docker主机上的Pod间通信通过Docker0桥即可直接进行，而跨Docker主机的Pod间通信时，Pod的报文经由本机上的flannel桥隧道转发其目标网络所在的Docker主机的flannel桥，并到达目标网络，而后由目标Pod接收，如下图所示



Flannel将虚拟IP和主机地址的映射关系存在于etcd中，运行于各主机上的flannel进程负责监控etcd中与当前主机相关的配置信息，并完成报文路由。

安装flannel

Flannel需要运行于集群的各节点，包括master主机和各Node主机，flannel程序包位于镜像的Extras仓库中，配置好之后使用yum install安装即可。

[root@master ~]# yum -y install flannel

[root@node1 ~]# yum -y install flannel

[root@node2 ~]# yum -y install flannel

配置flannel

Flannel需要基于etcd存储数据，配置其使用etcd服务的URL，配置文件为/etc/sysconfig/flanneld。配置详情如下

# etcd服务器URL列表

FLANNEL\_ETCD\_ENDPOINTS=<http://etcd1:2379>

# flannel向etcd查询网络地址池相关配置时使用的key，其值需要手动创建

FLANNEL\_ETCD\_PREFIX=”/atomic.io/network”

# 传递flannel程序的其他选项

#FLANNEL\_OPTIONS=””

[root@master ~]# vim /etc/sysconfig/flannel

FLANNEL\_ETCD\_ENDPOINTS="http://etcd1:2379"

FLANNEL\_ETCD\_PREFIX="/atomic.io/network"

[root@node1 ~]# vim /etc/sysconfig/flannel

FLANNEL\_ETCD\_ENDPOINTS="http://etcd1:2379"

FLANNEL\_ETCD\_PREFIX="/atomic.io/network"

[root@node2 ~]# vim /etc/sysconfig/flannel

FLANNEL\_ETCD\_ENDPOINTS="http://etcd1:2379"

FLANNEL\_ETCD\_PREFIX="/atomic.io/network"

在etcd配置flanneld使用的地址池

[root@master ~]# etcdctl --endpoints http://etcd1:2379 ls /

/registry

[root@master ~]# etcdctl --endpoints http://etcd1:2379 mkdir /atomi

c.io/network

创建键值对

[root@master ~]# etcdctl --endpoints http://etcd1:2379 mk /atomic.i

o/network/config '{"Network": "10.55.0.0/16"}'

{"Network": "10.55.0.0/16"}

[root@master ~]# etcdctl --endpoints http://etcd1:2379 ls /atomic.i

o/network

/atomic.io/network/config

[root@master ~]# etcdctl --endpoints http://etcd1:2379 get /atomic.

io/network/config

{"Network": "10.55.0.0/16"}

启动flannel，并重启各服务

在master主机启动flannel服务

[root@master ~]# systemctl start flanneld

[root@master ~]# systemctl enable flanneld

Created symlink from /etc/systemd/system/multi-user.target.wants/f

lanneld.service to /usr/lib/systemd/system/flanneld.service.

Created symlink from /etc/systemd/system/docker.service.requires/f

lanneld.service to /usr/lib/systemd/system/flanneld.service.

在各Node主机上启动flannel服务，并重启docker服务即可完成配置

# systemctl start flanneld.service

# systemctl enable flannel.service

# systemctl restart docker.service

[root@node1 ~]# ip a | grep "flann\*\|docker"

4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN

inet 10.55.15.1/24 scope global docker0

5: flannel0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER\_UP> mtu 1472 qdisc pfifo\_fast state UNKNOWN qlen 500

inet 10.55.15.0/16 scope global flannel0

测试

启动多个Pod，查看其启动的主机及获得的IP地址，并测试个Pod中的容器间的连通性即可。

# Kubernetes Dashboard