**K8s实践文档**

[**1** **环境信息** 2](#_Toc474677238)

[**2** **Systemd基础** 2](#_Toc474677239)

[**3** **Docker基础** 2](#_Toc474677243)

[**3.1** **安装** 2](#_Toc474677244)

[**3.2** **常用管理命令** 3](#_Toc474677250)

[**3.3** **国内镜像加速设置** 5](#_Toc474677251)

[**3.4** **私有仓库建设** 5](#_Toc474677256)

[**3.4.1** **自带的仓库启动** 5](#_Toc474677257)

[**3.4.2** **第三方私有仓库harbor** 6](#_Toc474677258)

[**3.5** **Dockerfile及构建** 6](#_Toc474677262)

[**3.5.1** **Dockerfile指令说明** 7](#_Toc474677263)

[**3.5.2** **构建** 9](#_Toc474677264)

[**3.6** **Docker存储** 9](#_Toc474677266)

[**4** **Kubernetes基础** 10](#_Toc474677267)

[**4.1** **Kubernetes介绍** 10](#_Toc474677268)

[**4.2** **Kubernetes的安装** 11](#_Toc474677269)

[**4.2.1** **V1.4版的yum安装及配置（基于centos7.3-64）** 11](#_Toc474677270)

[**4.2.2** **V1.5.2（目前最新版）的docker镜像方式安装** 13](#_Toc474677272)

[**4.3** **启动实例及pod、rc、svc配置文件编排说明** 14](#_Toc474677276)

[**4.3.1** **一个极简的nginx示例配置** 14](#_Toc474677277)

[**4.3.2** **Pod配置说明** 15](#_Toc474677278)

[**4.3.3** **Rc配置说明** 16](#_Toc474677279)

[**4.3.4** **Svc配置说明** 16](#_Toc474677280)

[**4.4** **核心管理工具Kubectl** 17](#_Toc474677281)

[**4.5** **Kubernetes的常用组件** 17](#_Toc474677284)

[**4.5.1** **Kube-dns** 17](#_Toc474677285)

[**4.5.2** **kubernetes-dashboard** 17](#_Toc474677286)

[**4.5.3 heapster 17**](#_Toc474677290)

[**4.5.4 ingress 17**](#_Toc474677294)

[**4.6 踩过的坑 18**](#_Toc474677297)

[**4.6.1 Kebeadm方式安装目前不支持docker最新版的1.13 18**](#_Toc474677298)

[**4.6.2 Kubeadm方式安装对btrfs格式支持不好。 18**](#_Toc474677303)

[**4.7 要走的路 18**](#_Toc474677305)

[**4.7.1 Kube-dns的功能测试 18**](#_Toc474677306)

[**4.7.2 Ingress的插件安装及功能使用测试 18**](#_Toc474677308)

[**4.7.3 Deployment实践 18**](#_Toc474677310)

[**4.7.4 Dockerfile的模板建立 18**](#_Toc474677313)

[**4.7.5 Nodeselector 18**](#_Toc474677316)

[**4.7.6 资源控制 18**](#_Toc474677319)

[**4.7.7 性能测试 19**](#_Toc474677320)

[**4.7.8 内部应用迁移测试 19**](#_Toc474677321)

[**4.7.9 持久化存储应用所需的分布式文件系统环境 19**](#_Toc474677322)

[**4.7.10 持久化应用迁移测试 19**](#_Toc474677324)

1. 环境信息

|  |  |  |  |
| --- | --- | --- | --- |
| **IP地址** | **软件版本** | **功能** | **备注** |
| 192.168.5.60 | Docker-1.12.5(btrfs)etcd kubernetes-masterflannel kubernetes-node | 1.4.0版本master+node | Yum方式安装，系统服务方式启动 |
| 192.168.5.61 | Docker-1.12.5(btrfs)flannel kubernetes-node | 1.4.0版本node |  |
| 192.168.5.62 | Docker-1.12.5(btrfs)KubeadmkubectlKubeletkubernetes-cni | 1.52版本master+node（docker存储类型btrfs） | Docker镜像方式启动 |
| 192.168.5.63 | Docker-1.11.2(overlay)KubeadmkubectlKubeletkubernetes-cni | 1.52版本master+node（docker存储类型overlay） | Docker镜像方式启动 |
| 192.168.5.64 | Docker-1.11.2(devicemapper) | Harbor仓库 | docker-compose |

1. Systemd基础

在服务管理，启动参数修改等部分需要用到。Centos7的核心。下面这两篇文档介绍的比较清楚

<http://www.ruanyifeng.com/blog/2016/03/systemd-tutorial-commands.html>

 <http://www.ruanyifeng.com/blog/2016/03/systemd-tutorial-part-two.html>

1. **Docker基础**
	1. 安装

最新版安装：

官网或者阿里云的镜像站点均可。

#curl -sSL https://get.docker.com/ | sh

#curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh -

指定版本安装：

 Docker-engine和docker-engine-selinux两个软件包需要一起安装。

可在<http://mirrors.aliyun.com/docker-engine/yum/repo/main/centos/7/Packages> 找到指定版本的docker程序下载安装，推荐使用yum方式安装本地或者url的软件包，可以解决依赖关系

yum localinstall http://mirrors.aliyun.com/docker-engine/yum/repo/main/centos/7/Packages/docker-engine-1.11.2-1.el7.centos.x86\_64.rpm <http://mirrors.aliyun.com/docker-engine/yum/repo/main/centos/7/Packages/docker-engine-selinux-1.11.2-1.el7.centos.noarch.rpm>

* 1. **常用管理命令**

#####镜像操作###################

docker search centos

#搜索镜像

docker pull centos:

#获取centos镜像

docker tag centos:v1 192.168.5.64/centos:v1

#给镜像打上tag

docker push 192.168.5.64/centos:v1

#上传镜像到私有仓库5.64

docker save –o centos.tar centos:v1

# centos为镜像name 导出的镜像就在当前所在目录

docker load -i centos.tar

#导入镜像

docker images

#查看镜像

docker rmi 镜像id

#删除镜像

docker rmi $(docker images -q -f dangling=true)

#删除所有未打 dangling 标签的镜像

#########容器操作###############

docker run –i -t -d -p 23:22 -v /home/soft:/home/soft --name centostest 904d6c400333

 #-t - 分配一个（伪）tty (link is external)

 #-i - 交互模式 (so we can interact with it)

 #-d 以后台模式运行

 #-p 将容器中的22端口映射出来

 #-v 映射数据卷，其实就是把宿主机的目录挂载到容器上 #--name给容器进行命如果不命名则默认会有个名字。此处的名字是centos904d6c400333启动容器是基于哪个镜像来启动的，镜像id，此处也可以是镜像的name:tag 镜像名字加镜像版本

docker ps

#-- 查看已经启动的镜像

docker ps –a

# 查看所有的镜像

docker stop b94c2269ea72

#容器id/容器name

docker start b94c2269ea72

#启动容器

docker attach b94c2269ea72

#进入容器

docker rm 27e55af81534

#删除一个正在运行的容器

docker rm –f 27e55af81534

#强制删除一个正在运行的容器

#########容器日志操作############################

docker run -i -t --name test2 -d centos /bin/sh -c "while true;do echo hello world;sleep 1;done" #后台启动docker -d

##启动时可以指定 --log-driver="syslog" 将日志输出到syslog中

docker ps

#查看启动的镜像

docker logs test2

#查看日志

docker logs test2 -f

#日志跟踪(全部输出)

docker logs --tail 10 test2

#tail -10

docker logs --tail 0 -f test2

#tail -f

#########其它常用############################

docker info

docker top test2

#查看docker的进程

docker stats test2

#docker的cpu 内存使用等

docker exec -d test2 touch /etc/testfile

#后台运行的容器交互，可以为正在运行的test2 新建一个文件 -u可以指定用户属主

docker exec -t -i test2 /bin/bash

#切换进入交互式

docker run --restart=always --name test3 -d centos /bin/sh -c "while true;do echo hello world;sleep 1;done" #--restart=always表示容器退出就会自动重启。

#--restart=on-failure:5 表示非0退出是会尝试自动重启，最多重启5次。 1.2引入

docker inspect test3

#获取容器的所有信息，类似ansible的setup模块

docker inspect --format='{{ .State.Running }}' test3

#获取容器的Running状态

docker inspect --format='{{ .NetworkSettings.IPAddress }}' test3

#获取IP

docker inspect --format='{{ .Name }} {{ .State.Running }}' test3 test2

#查看多个容器的信息

docker rm -f `docker ps -a -q`

#删除所有容器，-a 所有 -q 只返回ID Dockerfile及构建

* 1. **国内镜像加速设置**

阿里私有加速设置帮助文档：

<https://cr.console.aliyun.com/?spm=5176.100239.blogcont29941.12.3dataB#/accelerator>

具体操作基本就是去阿里官网申请一个专属的加速地址，而后在自己的docker程序启动参数加上这个地址，登录账号就可以加速docker.io的镜像和可以下载阿里docker云仓库别人上传的镜像了。

操作实例：

cp -n /lib/systemd/system/docker.service /etc/systemd/system/docker.service

vi /etc/systemd/system/docker.service

 ExecStart=/usr/bin/docker-current daemon --registry-mirror=https://5zrvze64.mirror.aliyuncs.com

systemctl daemon-reload

service docker restart

docker login --username=mingyan\_xiong@163.com registry.cn-hangzhou.aliyuncs.com

docker pull registry.cn-hangzhou.aliyuncs.com/mykubernetes/kubernetes-dashboard-amd64:v1.5.0

* 1. **私有仓库建设**
		1. **自带的仓库启动**

docker run -d -p 5000:5000 registry

#1.3后需要在/lib/systemd/system/docker.service 添加--insecure-registry localhost:5000

systemctl daemon-reload

service docker restart

docker images

#查看本地的image的ID信息，方便下面给服务器上名称打tag

docker tag e63ccdb74c3b localhost:5000/xmy/centos\_web

#打上tag信息，不指定版本则为last

docker push localhost:5000/xmy/centos\_web

#上传镜像

docker images localhost:5000/xmy/centos\_web

#查看本地仓库的镜像

* + 1. **第三方私有仓库harbor**

Harbor是VMware公司的中国团队开源的一个docker私有仓库软件，相比于docker自带的仓库多了UI界面还有权限控制等功能。

Github项目地址：<https://github.com/vmware/harbor/>

Harbor部署：

git clone https://github.com/vmware/harbor

vi harbor.cfg {

 # hostname 设置访问地址，支持IP，域名，主机名，禁止设置127.0.0.1

 hostname = reg.mydomain.com

 # 访问协议，可设置 http,https

 ui\_url\_protocol = http

 # harbor WEB UI登陆使用的密码

 harbor\_admin\_password = Harbor12345

 # 认证方式，这里支持多种认证方式，默认是 db\_auth ，既mysql数据库存储认证。

 # 这里还支持 ldap 以及 本地文件存储方式。

 auth\_mode = db\_auth

 }

./prepare

yum install python-pip

pip install --upgrade pip

pip install docker-compose

pip install --upgrade backports.ssl\_match\_hostname

mv docker-compose.tpl docker-compose.yml

docker-compose up –d

* 1. **Dockerfile及构建**
		1. **Dockerfile指令说明**

 CMD{

 docker run -d -p 192.168.120.204:80:80 --name web2 centos\_web:v1 /bin/true

 CMD ["/bin/true"]

 CMD传递参数

 CMD ["/bin/bash","-l"]

 docke run 命令可以覆盖CMD命令，只有一条生效，多条只认最后一条，如果多个进程可以使用supervisor管理

 }

 ENTRYPOINT{

 docker run 提供的cmd参数信息会被当做参数传递

 ENTRYPOINT ["/usr/sbin/nginx"]

 docker run -d -p 80 --name web1 centos\_web:v1 -g "daemon off;" #启动的时候就不用指定nginx

 #也可以直接写作下面方式

 ENTRYPOINT ["/usr/sbin/nginx", "-g", "daemon off;"]

 #ENTRYPOINT和CMD结合使用

 ENTRYPOINT ["/usr/sbin/nginx"]

 CMD ["-h"]

 这样docker run的时候可以指定CMD，如不指定则默认使用-h参数

 # docker run的时候可以使用--entrypoint 来覆盖ENTRYPOINT参数

 }

 WORKDIR{

 指定一个工作目录，上面两个选项会被指定在这个目录下执行

 WORKDIR /opt/webapp/db

 RUN bundle install

 WORKDIR /opt/webapp

 ENTRYPOINT ["rackup"]

 使用-w 参数覆盖

 }

 ENV{

 用在构建过程中的环境变量

 ENV RAM\_PATH /home/soft/

 #设置多个环境变量

 ENV RAM\_PATH=/home/soft test="test"

 #示例

 ENV WORK\_DIR=/opt/soft

 WORKDIR $WORK\_DIR

 -e 参数可以传递环境变量，只是在运行时有效

 }

 USER{

 指定运行的用户

 USER nginx

 USER user:group

 -u 参数可以覆盖，默认为root

 }

 VOLUME{

 为容器添加卷

 \*卷可在容器间共享和重用

 \*修改是及时生效

 \*卷会一直存在，直到没有容器使用

 VOLUME [ "/opt/project", "/data" ]

 }

 ADD{

 将构建环境下的文件和目录复制到镜像中

 ADD software.lic /opt/app/software.lic

 ADD http://wordpress.org/latest.zip /root/wordpress.zip

 #ADD还可以直接将gzip bzip2 xz的文件解压到文件夹

 ADD last.tar.gz /var/www/wordpress/

 ####ADD参数会使构建缓存变得无效

 }

 COPY{

 只复制不会文件提取和解压

 cp /cond.d/ /opt/app/cond.d/

 需要和dockerfile放在同一级目录

 }

 LABEL{

 元数据，键值对形式。用于docker inspect 查阅，做标识符用

 LABEL VERSION="1.0" type="test"

 }

 STOPSINGNAL/ARG{

 少用，一个是给一个系统调用信号给容器。一个是在构建时指定运行时的变量（预定义变量）

 }

 ONBUILD{

 可以复用dockerfile文件，类似模板文件使用，只递归一次。详细使用需查阅

 }

* + 1. **构建**

**示例dockerfile**

[root@k8s-node1 test]# ls

Dockerfile jdk profile resin test

[root@k8s-node1 test]# cat Dockerfile

# Version: 0.1

FROM 192.168.20.198:5000/centos

MAINTAINER xmy "mingyan\_xiong@163.com"

ENV REFRESHED\_AT 2016-10-04

ADD jdk /usr/local/jdk

ADD resin /usr/local/resin

ADD test /usr/local/wwwweb/test

EXPOSE 8090

CMD /usr/local/resin/bin/resin.sh -server test start &&sleep 3; tail -f /usr/local/resin/log/jvm-test.log

 **构建及调试**

docker build -t="centos\_resin\_test:v1" .

如果构建失败，可以从失败的前一个ID进入容器手动执行排错

docker run -t -i 构建最后一步成功的ID /bin/bash

 --no-cache参数可以忽略之前的构建缓存。构建缓存可以使构建过程更快。

* 1. **Docker存储**

<https://docs.docker.com/engine/userguide/storagedriver>

当前docker支持的storage driver包括：aufs，btrfs，devicemapper，OverlayFs，ZFS 5种存储驱动，在centos最新版的环境下，docker1.13之前默认的存储引擎是devicemapper，docker1.13\_XFS默认是overlayFS。目前主流在用的是devicemapper、btrfs和overlayFS。



Devicemapper：

 Docker1.12及以下在centos上默认的存储驱动，默认是使用loop环回设备存储块，默认大小是100G，官网推荐使用devicemapper的话最好使用重新挂载独立的lvm设备卷的方式，官网有操作实例，在多数的rhel系列下为推荐存储驱动。

<https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/>

OverlayFs：

 Overlay现在有两个版本的存储驱动，overlay2的存储驱动方式已经加入到内核4.0中，在centos7.3下，docker1.13默认的存储驱动已经是overlay。想比较与前者，overlay速度会更快。支持多容器的缓存共享。

<https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/>

btrfs：

 btrfs一直被认为是下一代的linux存储系统。相比于其它存储驱动，它支持更多的高级功能，比如写时复制、快照模式。Docker官网认为btrfs非常适合docker使用。但是目前btrfs还在发展阶段。其特性缺点也比较明显，在小文件操作上会比较慢，不支持页面缓存等。

<https://docs.docker.com/engine/userguide/storagedriver/btrfs-driver/>

1. **Kubernetes基础**
	1. **Kubernetes介绍**

Kubernetes是一个开源的，用于管理云平台中多个主机上的容器化的应用，Kubernetes的目标是让部署容器化的应用简单并且高效（powerful）,Kubernetes提供了应用部署，规划，更新，维护的一种机制。

Kubernetes一个核心的特点就是能够自主的管理容器来保证云平台中的容器按照用户的期望状态运行着（比如用户想让apache一直运行，用户不需要关心怎么去做，Kubernetes会自动去监控，然后去重启，新建，总之，让apache一直提供服务），管理员可以加载一个微型服务，让规划器来找到合适的位置，同时，Kubernetes也系统提升工具以及人性化方面，让用户能够方便的部署自己的应用（就像canary deployments）。

中文文档：

 <https://www.kubernetes.org.cn/k8s>

上面链接是k8s的中文文档，部署部分比较老了，可以不用看，但是它的设计架构、名词解释还有kubectl介绍以及插件部分一定要看。入门基本认识必备。

官方手册：

 <https://kubernetes.io/docs/user-guide/>

GitHub项目地址：

 <https://github.com/kubernetes>

Kubernetes集群组件:

　　- etcd 一个高可用的K/V键值对存储和服务发现系统

　　- flannel 实现夸主机的容器网络的通信

　　- kube-apiserver 提供kubernetes集群的API调用

　　- kube-controller-manager 确保集群服务

　　- kube-scheduler 调度容器，分配到Node

　　- kubelet 在Node节点上按照配置文件中定义的容器规格启动容器

　　- kube-proxy 提供网络代理服务

* 1. **Kubernetes的安装**
		1. **V1.4版的yum安装及配置（基于centos7.3-64）**

**EPEL源安装**

yum -y install epel-release

或者

wget -P /etc/yum.repos.d/ <http://mirrors.aliyun.com/repo/epel-7.repo>

 **master节点**

yum -y install etcd kubernetes-master

 **编辑如下两个配置文件**

 grep -Ev "#|^$" /etc/etcd/etcd.conf

 ETCD\_NAME=default

 ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

 ETCD\_LISTEN\_CLIENT\_URLS=<http://0.0.0.0:2379>

 ETCD\_ADVERTISE\_CLIENT\_URLS="http://0.0.0.0:2379"

 grep -Ev "#|^$" /etc/kubernetes/apiserver

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

 KUBE\_API\_PORT="--port=8080"

 KUBELET\_PORT="--kubelet-port=10250"

 KUBE\_ETCD\_SERVERS="--etcd-servers=http://127.0.0.1:2379"

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

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

 KUBE\_API\_ARGS=""

 主要修改127.0.0.1的监听为0.0.0.0。

  **启动服务**

for SERVICES in etcd kube-apiserver kube-controller-manager kube-scheduler; do systemctl restart $SERVICES;systemctl enable $SERVICES;systemctl status $SERVICES ; done

 **写入网络信息至ETCD存储**

etcdctl mk /atomic.io/network/config '{"Network":"172.17.0.0/16"}'

 更多的ETCD操作命令及功能可百度了解。

 **Minion节点**

yum install flannel kubernetes-node

  **编辑配置如下三个配置文件**

 grep -Ev "#|^$" /etc/sysconfig/flanneld

 FLANNEL\_ETCD\_ENDPOINTS=<http://192.168.20.198:2379>

 #指定到master节点IP

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

 grep -Ev "#|^$" /etc/kubernetes/config

 KUBE\_LOGTOSTDERR="--logtostderr=true"

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

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

 KUBE\_MASTER="--master=http://192.168.20.198:8080"

 #master节点的IP

 grep -Ev "#|^$" /etc/kubernetes/kubelet

 KUBELET\_ADDRESS="--address=0.0.0.0"

 KUBELET\_PORT="--port=10250"

 KUBELET\_HOSTNAME="--hostname-override=192.168.20.197"

 #自己的内网IP

 KUBELET\_API\_SERVER="--api-servers=http://192.168.20.198:8080"

 #master节点的IP

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

 #KUBELET\_ARGS=""

 **启动服务**

for SERVICES in kube-proxy kubelet docker flanneld;do systemctl restart $SERVICES;systemctl enable $SERVICES;systemctl status $SERVICES; done

  **master验证节点是否加入集群**



* + 1. V1.5.2（目前最新版）的docker镜像方式安装

官网参考链接：

 <https://kubernetes.io/docs/getting-started-guides/kubeadm/>

**注：kubeadm安装方式目前支持最新的docker的版本是1.12，最新版的1.13是不支持的。安装好k8s后，会发现网络桥接不成功。**



**Hosts添加以下条目（google的项目被墙了）**

61.91.161.217 gcr.io

61.91.161.217 [www.gcr.io](http://www.gcr.io)

 **master安装**

git clone https://github.com/kubernetes/release.git

/opt/release/rpm/docker-build.sh

 #构建完成后在该目录下会生成output，里面有主程序的rpm包

 kubeadm-1.6.0-0.alpha.0.2074.a092d8e0f95f52.x86\_64.rpm

 kubelet-1.5.2-0.x86\_64.rpm

 kubectl-1.5.2-0.x86\_64.rpm

 kubernetes-cni-0.3.0.1-0.07a8a2.x86\_64.rpm

yum localinstall ./\*.rpm

kubeadm init --api-advertise-addresses=192.168.5.63 --pod-network-cidr 10.244.0.0/16

操作说明：首先去k8s官网git下载release项目，进去里面的rpm目录执行docker-build.sh构建生成程序所需的主程序软件包。安装完所有软件包之后就可以使用kubeadm来初始化了。其中--api-advertise-addresses需要时群集master的IP，1.5版后支持master群集了。这个地方可以有多个IP，--pod-network-cidr，在使用flannel为网络支持插件的时候必须的参数。其它网络插件可以不指定。当运行init初始化后，程序会自动去grc.io的docker仓库下载程序所需的其它组件。等待一段时候之后会有如下反馈：

Your Kubernetes master has initialized successfully!

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

 http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following on each node:

kubeadm join --token=e10910.87429cca504bb339 192.168.5.63

 记录保存好--token=e10910.87429cca504bb339，节点加入集群需要。

Kubeadm：安装管理

Kubelet：kube主程序

Kubectl：核心管理调度群集管理

kubernetes-cni：cni网络插件

 **如果需要master同时兼并minion节点功能（能够分配到pod实例）**

kubectl taint nodes --all dedicated-

 **master节点应用网络插件配置（flannel）**

支持的网络组件列表：<https://kubernetes.io/docs/admin/addons/>

 Flannel的yaml配置文件：

<https://github.com/coreos/flannel/blob/master/Documentation/kube-flannel.yml>

 直接复制这个文件在master节点保存为flannel.yml文件

kubectl apply -f flannel.yml

 等待网络组件状态为running后，即可去minion节点加入集群了。



 **Minion节点加入集群**

 kubeadm-1.6.0-0.alpha.0.2074.a092d8e0f95f52.x86\_64.rpm

 kubelet-1.5.2-0.x86\_64.rpm

 kubectl-1.5.2-0.x86\_64.rpm

 kubernetes-cni-0.3.0.1-0.07a8a2.x86\_64.rpm

yum localinstall ./\*.rpm

kubeadm join --token=e10910.87429cca504bb339 192.168.5.63

* 1. 启动实例及pod、rc、svc配置文件编排说明
		1. 一个极简的nginx示例配置

apiVersion: v1

kind: ReplicationController

metadata:

 name: nginx-rc

spec:

 replicas: 2

 selector:

 name: nginx

 template:

 metadata:

 labels:

 name: nginx

 spec:

 containers:

 - name: nginx

 image: 192.168.5.64/test/nginx:1.11.5

 imagePullPolicy: IfNotPresent

 ports:

 - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

 name: nginx-svc

spec:

 type: NodePort

 ports:

 - port: 80

 nodePort: 30002

 protocol: TCP

 name: http

 selector:

 name: nginx

注：配置文件的编写，支持yml和json两种格式，yml文件编写切记不要使用tab键去补齐排版。正常缩进是以两个空格为单位编写。否则会有样式问题。上面配置文件包含rc和svc两个模块的最简单配置。

* + 1. **Pod配置说明**

[**https://kubernetes.io/docs/user-guide/pods/multi-container/**](https://kubernetes.io/docs/user-guide/pods/multi-container/)







* + 1. Rc配置说明

[**https://kubernetes.io/docs/user-guide/replication-controller**](https://kubernetes.io/docs/user-guide/replication-controller)



* + 1. **Svc配置说明**

<https://kubernetes.io/docs/user-guide/services/>





* 1. **核心管理工具Kubectl**

<https://kubernetes.io/docs/user-guide/kubectl/>

<http://blog.csdn.net/xingwangc2014/article/details/51204224>

* 1. Kubernetes的常用组件
		1. **Kube-dns**

服务发现功能，主要是为了解决在群集内clusterIP发生变化别的应用调用不到服务的问题。就是提供一个动态的DNS功能。在1.3版kube-dns的核心功能提供使用的是skydns，在1.5版本后kube-dns默认内置使用了dnsmasq来提供服务。

Skydns的安装：

<http://www.cnblogs.com/xkops/p/6232392.html>

* + 1. **k****ubernetes-dashboard**

k8s的UI管理界面提供，可以完成命令行90%的功能。

<https://kubernetes.io/docs/user-guide/ui/>

部分图表功能需要heapster插件支持。

* + 1. heapster

资源使用的一个监控工具。

<https://kubernetes.io/docs/user-guide/monitoring/>

主要是将cAdvisor收集到的资源使用和性能分析数据存储整合在重新展示出来。

* + 1. ingress

7层的服务暴露方法，利用的就是nginx的反向代理功能。和nodeport方式相比选择性会更强。而且节点所需的端口开销更小。

<https://kubernetes.io/docs/user-guide/ingress/>

* 1. 踩过的坑
		1. Kebeadm方式安装目前不支持docker最新版的1.13

全程安装都不会出现问题，也没有错误提示。当所pods状态为running的时候，发现nodeport方式暴露出来的服务端口监听正常就是访问不到。排查过程中对iptables、网卡名称、还有flannel网络插件都分别进行过各种折腾。最后用上VMware的网络插件还是一样的。才重新去官网爬文。找到这么一段docker版本的说明信息。



后来切换到1.11.2和1.12.5版均能正常工作。

主要的问题分析：kube的cni网络插件是在docker0的网卡上做的桥接，而后使用相应的网络组件来提供网络支撑的底层服务。应该是docer1.13在网络这边有做接口改动，kube还未更新相应的方法导致。

* + 1. Kubeadm方式安装对btrfs格式支持不好。

Kube还不支持btrfs存储驱动。使用kubeadm方式安装日志一直会刷一个读取不到空间大小的错误信息。

* 1. 要走的路
		1. Kube-dns的功能测试

使用的环境会比较多。这个需要测试了解清楚。

* + 1. Ingress的插件安装及功能使用测试

这个还是一个比较重要的功能点。如果测试没问题的话。基本可以将我们的接入nginx和应用nginx的功能区分开。

* + 1. Deployment实践

Kubectl rolling-update使用命令类似前端更新，而使用deployment配置文件的更新，更像是后端更新有专门的controller来负载这个工作。

<https://kubernetes.io/docs/user-guide/deployments/>

* + 1. Dockerfile的模板建立

Docker镜像包的大小控制，灵活性通用性控制。一个示例

<http://blog.tingyun.com/web/article/detail/825>

* + 1. Nodeselector

部分项目部署在指定节点的测试

<https://kubernetes.io/docs/user-guide/node-selection/>

* + 1. 资源控制
		2. 性能测试
		3. 内部应用迁移测试
		4. 持久化存储应用所需的分布式文件系统环境

<https://kubernetes.io/docs/user-guide/volumes/>

* + 1. 持久化应用迁移测试