

第十三章 搭建 Kubernetes 的 web 管理界面-基于 k8s 搭建 redis 集群案例

本节所讲内容：

13.1 部署 Kubernetes Dashboard web 界面

13.1.1 创建 dashboard-deployment.yaml deployment 配置文件

13.1.2 创建 dashboard-service.yaml service 配置文件

13.1.3 准备 kubernetes 相关的镜像

13.1.4 启动 dashboard 的 deployment 和 service

13.1.5 排错经验分享

13.1.6 查看 kubernetes dashboard web 界面

13.1.7 销毁 web 界面相关应用

13.2 在 kubernetes 上面的集群上搭建基于 redis 和 docker 的留言簿案例

13.2.1 创建 Redis master deployment 配置文件

13.2.2 创建 redis master service 配置文件

13.2.3 创建 redis slave deployment 配置文件

13.2.4 创建 slave service 配置文件

13.2.5 创建 frontend guestbook deployment 配置文件

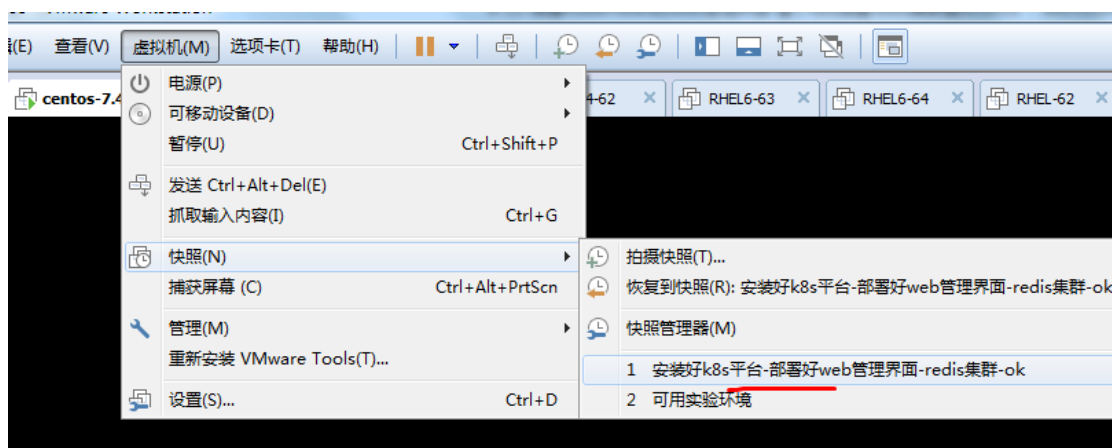
13.2.6 创建 frontend guestbook service 配置文件

13.2.7 查看外部网络访问 guestbook

总结使用 k8s 集群的步骤：

准备 deployment 和 service 配置文件-》导入相关镜像-》启动 deployment 和 service

实验环境：将三台虚拟机还原到以下状态：

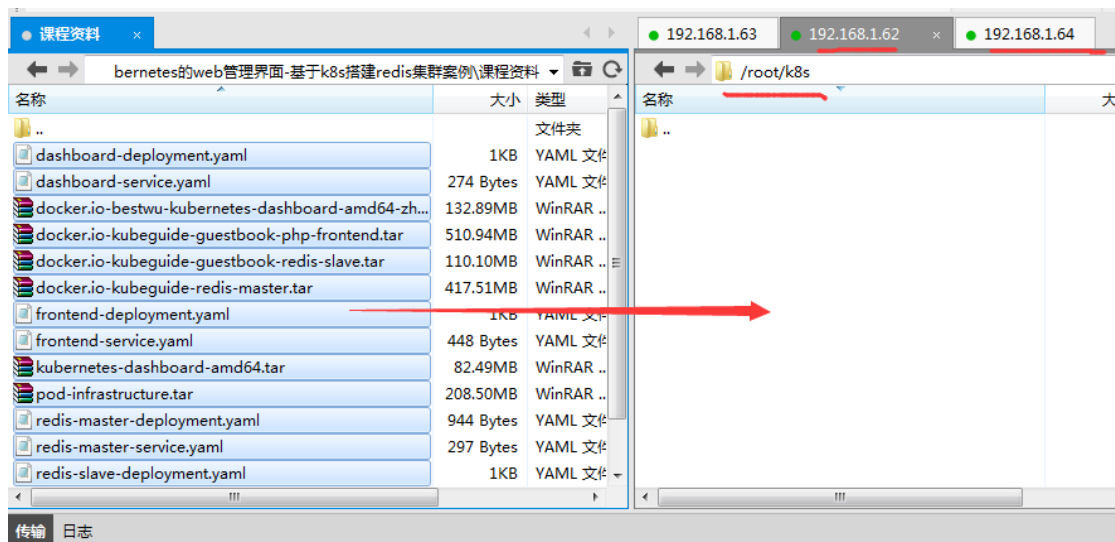


最好每个节点给 4G 内存

本地上传镜像到 node1 和 node2 上

```
[root@node1 ~]# mkdir /root/k8s
```

```
[root@node2 ~]# mkdir /root/k8s
```



启动服务：

```
[root@xuegod63 ~]# systemctl restart kube-apiserver kube-controller-manager
kube-scheduler flanneld etcd
```

```
[root@xuegod63 ~]# systemctl enable kube-apiserver kube-controller-manager
kube-scheduler flanneld etcd
```

查看环境：

```
[root@master ~]# kubectl get node
```

NAME	STATUS	AGE
node1	Ready	4d
node2	Ready	4d

到此，说明我的 kubernetes 环境是正常。接下就可以实验。

13.1 部署 Kubernetes Dashboard web 界面

Kubernetes Dashboard（仪表盘）是一个旨在将通用的基于 Web 的监控和操作界面加入 Kubernetes 的项目。

13.1.1 创建 dashboard-deployment.yaml 配置文件

```
[root@master ~]# vim /etc/kubernetes/dashboard-deployment.yaml #自己创建这个文件，插入以下内容，注意或更改以下红色部分
```

```
apiVersion: extensions/v1beta1
```

```
kind: Deployment
```

```
metadata:
```

```
# Keep the name in sync with image version and
```

```
# gce/coreos/kube-manifests/addons/dashboard counterparts
```

```
name: kubernetes-dashboard-latest
```

```
namespace: kube-system
```

```
spec: #定义 pod 属性
```

```
replicas: 1
```

```
template:
```

```

metadata:
  labels:
    k8s-app: kubernetes-dashboard
    version: latest
    kubernetes.io/cluster-service: "true"
spec: #定义容器的属性
  containers:
    - name: kubernetes-dashboard
      image: docker.io/bestwu/kubernetes-dashboard-amd64:v1.6.3 #这里修改成可以找到的镜像源。/bestwu/kubernetes-dashboard-amd64 这个是一个中文的 web 界面镜像
      imagePullPolicy: IfNotPresent
      resources:
        # keep request = limit to keep this container in guaranteed class
        limits: #关于 pod 使用的 cpu 和内存硬件资源做限制
          cpu: 100m
          memory: 50Mi
        requests:
          cpu: 100m
          memory: 50Mi
      ports:
        - containerPort: 9090
      args:
        - --apiserver-host=http://192.168.1.63:8080 #使用我给你的 deployment 文件时，这里写成自己的 apiserver 服务器地址和端口。
      livenessProbe:
        httpGet:
          path: /
          port: 9090
        initialDelaySeconds: 30
        timeoutSeconds: 30

```

13.1.2 创建编辑 dashboard-service.yaml 文件：

[root@master ~]# vim /etc/kubernetes/dashboard-service.yaml #插入以下内容

apiVersion: v1

kind: Service

metadata:

name: kubernetes-dashboard #这里要和上面的 deployment 中定义一样

namespace: kube-system #这里要和上面的 deployment 中定义一样

labels:

k8s-app: kubernetes-dashboard

kubernetes.io/cluster-service: "true"

spec:

selector:

k8s-app: kubernetes-dashboard

ports:

- port: 80

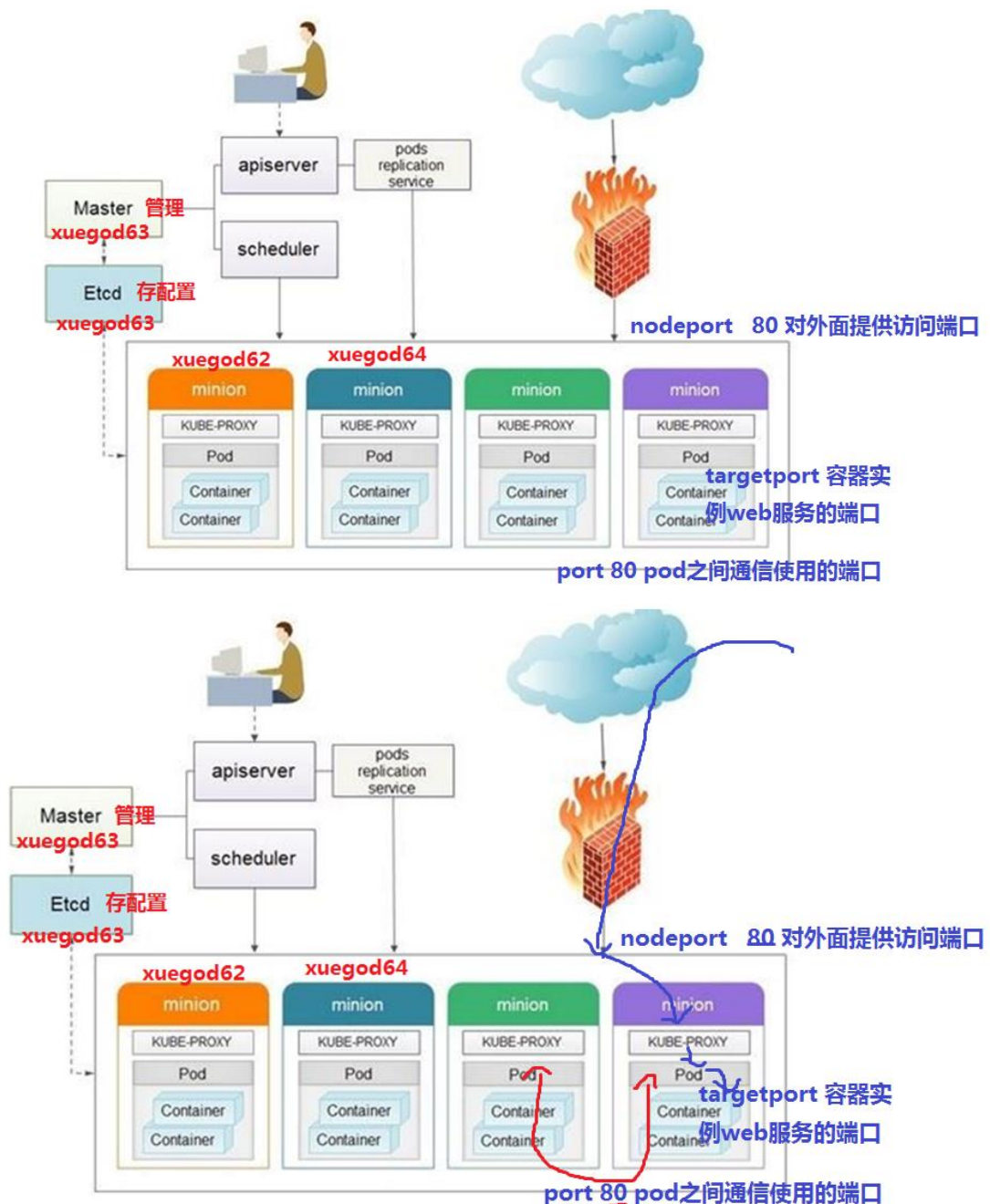
targetPort: 9090

扩展：service 的三种端口:

port：service 暴露在 cluster ip 上的端口，port 是提供给集群内部客户访问 service 的入口。

nodePort：nodePort 是 k8s 提供给集群外部客户访问 service 入口的一种方式。

targetPort：targetPort 是 pod 中容器实例上的端口，从 port 和 nodePort 上到来的数据最终经过 kube-proxy 流入到后端 pod 的 targetPort 上进入容器。



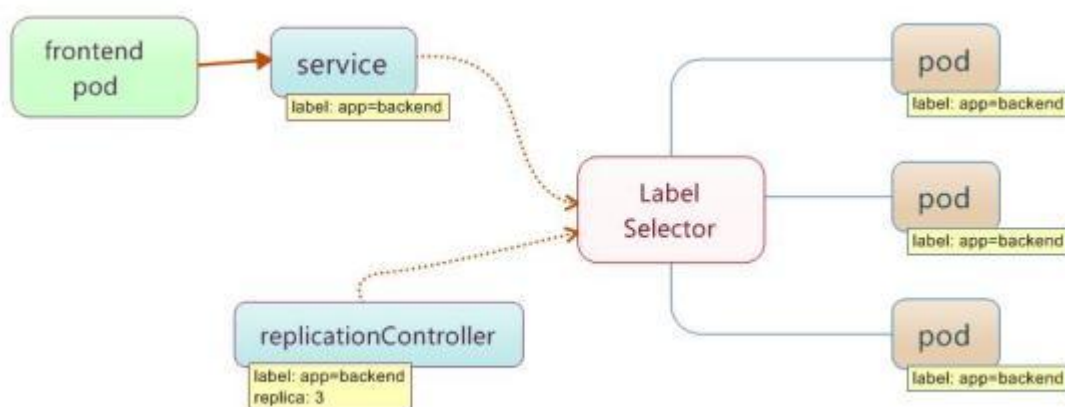
port、nodePort 总结：port 和 nodePort 都是 service 的端口，前者暴露给集群内客户访问服务，后者暴露给集群外客户访问服务。从这两个端口到来的数据都需要经过反向代理 kube-proxy 流入

后端 pod 的 targetPod，从而到达 pod 上的容器内。

service 概述：service 是 pod 的路由代理抽象，用于解决 pod 之间的服务发现问题。因为 pod 的运行状态可动态变化(比如切换机器了、扩容过程中被终止了等)，所以访问端不能以写死 IP 的方式去访问该 pod 提供的服务。service 的引入旨在保证 pod 的动态变化对访问端透明，访问端只需要知道 service 的地址，由 service 来提供代理。

replicationController：是 pod 的复制抽象，用于解决 pod 的扩容缩容问题。通常，分布式应用为了性能或高可用性的考虑，需要复制多份资源，并且根据负载情况动态伸缩。通过 replicationController，我们可以指定一个应用需要几份复制，Kubernetes 将为每份复制创建一个 pod，并且保证实际运行 pod 数量总是与该复制数量相等(例如，当前某个 pod 宕机时，自动创建新的 pod 来替换)。

总结：service 和 replicationController 只是建立在 pod 之上的抽象，最终是要作用于 pod 的，那么它们如何跟 pod 联系起来呢？这就要引入 label 的概念：label 其实很好理解，就是为 pod 加上可用于搜索或关联的一组 key/value 标签，而 service 和 replicationController 正是通过 label 来与 pod 关联的。如下图所示，有三个 pod 都有 label 为"app=backend"，创建 service 和 replicationController 时可以指定同样的 label:"app=backend"，再通过 label selector 机制，就将它们与这三个 pod 关联起来了。例如，当有其他 frontend pod 访问该 service 时，会自动会转发到其中的一个 backend pod。



13.1.3 准备 kubernetes 相关的镜像

在官方的 dashboard-deployment.yaml 中定义了 dashboard 所用的镜像：gcr.io/google_containers/kubernetes-dashboard-amd64:v1.5.1，启动 k8s 的 pod 还需要一个额外的镜像：registry.access.redhat.com/rhel7/pod-infrastructure:latest，这两个镜像在国内下载比较慢。可以使用 docker 自带的源先下载下来：

镜像获取方法 1：本地上传镜像到 node1 和 node2 上

node1 和 node2 都要导入以下 2 个镜像：

```
[root@node1 ~]# cd /root/k8s/
```

```
[root@node1 k8s]# docker load -i pod-infrastructure.tar
```

```
# docker load -i docker.io-bestwu-kubernetes-dashboard-amd64-zh.tar
```

注：这个在后期启动时，就更方便了。

镜像获取方法 2：在线下载镜像：

在 xuegod62 上，下载镜像：

```
[root@xuegod62 ~]# docker pull registry.access.redhat.com/rhel7/pod-infrastructure:latest
```

这是红帽的官方 docker 镜像下载站点，可以直接访问。

gcr.io 这个网址在国内，没有办法访问，我使用 docker.io 中的镜像：

```
[root@xuegod62 ~]# docker search kubernetes-dashboard-amd
```

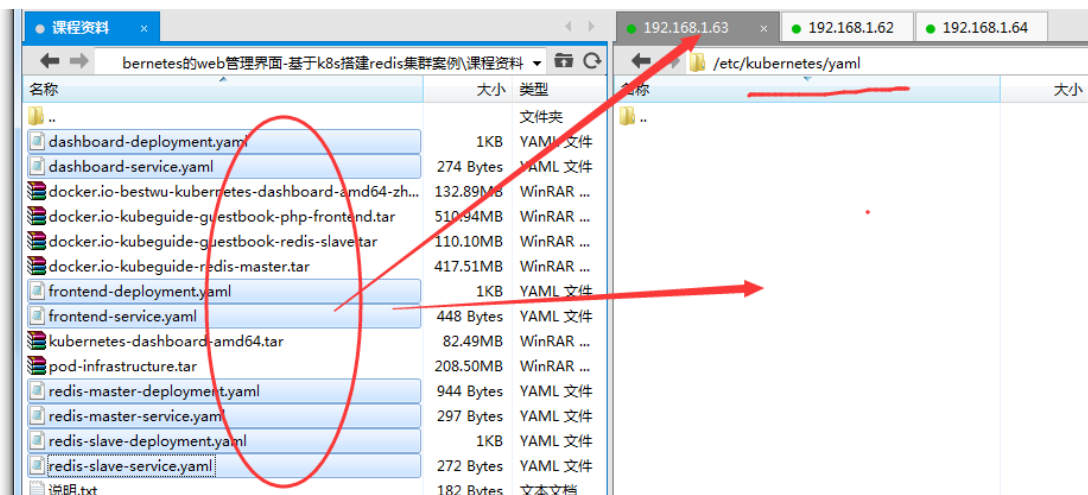
```
[root@xuegod62 ~]# docker pull docker.io/mritd/kubernetes-dashboard-amd64
```

13.1.4 启动 dashboard 的 deployment 和 service

```
[root@master kubernetes]# mkdir /etc/kubernetes/yaml
```

```
[root@master kubernetes]# cd /etc/kubernetes/yaml
```

将所有的 yaml 配置文件都上传到 **master** 的这个 /etc/kubernetes/yaml 目录



```
[root@master kubernetes]# cd /etc/kubernetes/yaml
```

```
# kubectl create -f /etc/kubernetes/yaml /dashboard-deployment.yaml
```

deployment "kubernetes-dashboard-latest" created

```
[root@master ~]# kubectl create -f /etc/kubernetes/yaml/dashboard-service.yaml
```

service "kubernetes-dashboard" created

到此，dashboard 搭建完成。

查看运行结果：

```
[root@k8s-master ~]# kubectl get deployment --all-namespaces
```

NAMESPACE	NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE
kube-system	kubernetes-dashboard-latest	1	1	1	1h

desired [dɪˈzaɪəd] 希望，渴望的；CURRENT 现在；UP-TO-DATE 最新的；available

可用

注：因为我们定义了 namespace，所以这需要加上--all-namespaces 才可以显示出来，默认只显示 namespaces=default 的 deployment。

```
[root@k8s-master ~]# kubectl get svc --all-namespaces
```

NAMESPACE	NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
-----------	------	------------	-------------	---------	-----

default	kubernetes	10.254.0.1	<none>	443/TCP	9d
kube-system	kubernetes-dashboard	10.254.44.119	<none>	80/TCP	1h

[root@master kubernetes]# kubectl get pod -o wide --all-namespaces

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	kubernetes-dashboard-latest-2249290861-jz4r6	1/1	Running	0	2m	10.255.92.2	node2

[root@master test]# kubectl get pod -n kube-system -o wide

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kubernetes-dashboard-latest-2661119796-4lt8q	1/1	Running	2	122d	10.255.61.3	node1

13.1.5 排错经验分享

创建成功 deployment 后，查看 pod 时，发现以下错误：

[root@master kubernetes]# kubectl get pod -o wide --all-namespaces

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	kubernetes-dashboard-latest-2351462241-slg41	0/1	CrashLoopBackOff	5	5m	10.255.9.5	node1

查看报错信息：

#kubectl logs -f kubernetes-dashboard-latest-2351462241-slg41 -n kube-system #显示

Error from server:

Get

https://node1:10250/containerLogs/kube-system/kubernetes-dashboard-latest-2351462241-slg41/kubernetes-dashboard?follow=true: dial tcp 192.168.1.62:10250: getsockopt: connection refused

注：

解决：

[root@node1 ~]# grep -v '^#' /etc/kubernetes/kubelet

改：KUBELET_ADDRESS="--address=127.0.0.1"

为：KUBELET_ADDRESS="--address=0.0.0.0"

[root@node1 ~]#systemctl restart kubelet.service

在 node2 上也同样执行以上命令。

[root@node2 ~]# grep -v '^#' /etc/kubernetes/kubelet

改：KUBELET_ADDRESS="--address=127.0.0.1"

为：KUBELET_ADDRESS="--address=0.0.0.0"

[root@node2 ~]#systemctl restart kubelet.service

报错 2：

[root@master ~]# kubectl logs kubernetes-dashboard-latest-2661119796-64km9 -n kube-system

Error from server: Get

<https://node2:10250/containerLogs/kube-system/kubernetes-dashboard-latest-2661119796-64km9/kubernetes-dashboard: dial tcp 192.168.1.64:10250: getsockopt: no route to host>

解决：

```
[root@node2 ~]# iptables -F #清空防火墙
```

```
[root@node2 ~]# systemctl stop firewalld
```

```
[root@node2 ~]# systemctl disable firewalld
```

清空防火墙后，再查看日志：

```
[root@master ~]# kubectl logs kubernetes-dashboard-latest-2661119796-4lt8q -n kube-system
```

Starting HTTP server on port 9090

Creating API server client for http://192.168.1.63:8080

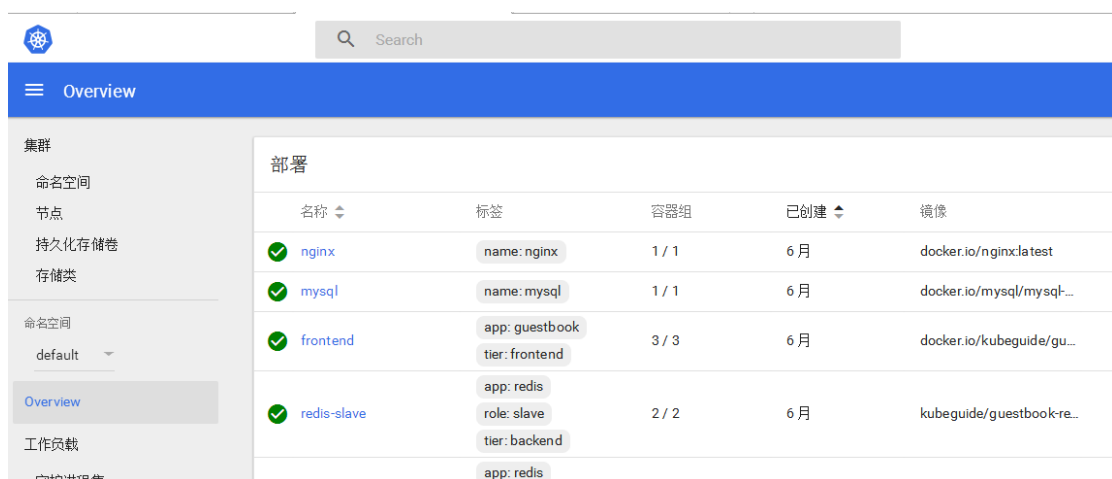
Successful initial request to the apiserver, version: v1.5.2

Creating in-cluster Heapster client

弹出以上界面说明我们配置成功。

13.1.6 查看 kubernetes dashboard web 界面

界面验证，浏览器访问：<http://192.168.1.63:8080/ui>



名称	标签	容器组	已创建	镜像
nginx	name: nginx	1 / 1	6 月	docker.io/nginx:latest
mysql	name: mysql	1 / 1	6 月	docker.io/mysql/mysql-...
frontend	app: guestbook tier: frontend	3 / 3	6 月	docker.io/kubeguide/gu...
redis-slave	app: redis role: slave tier: backend	2 / 2	6 月	kubeguide/guestbook-re...

注：到这里先做一个快照，用于后期自己学习时使用。

13.1.7 销毁 web 界面相关应用

```
[root@master ~]# kubectl delete deployment kubernetes-dashboard-latest --namespace=kube-system
```

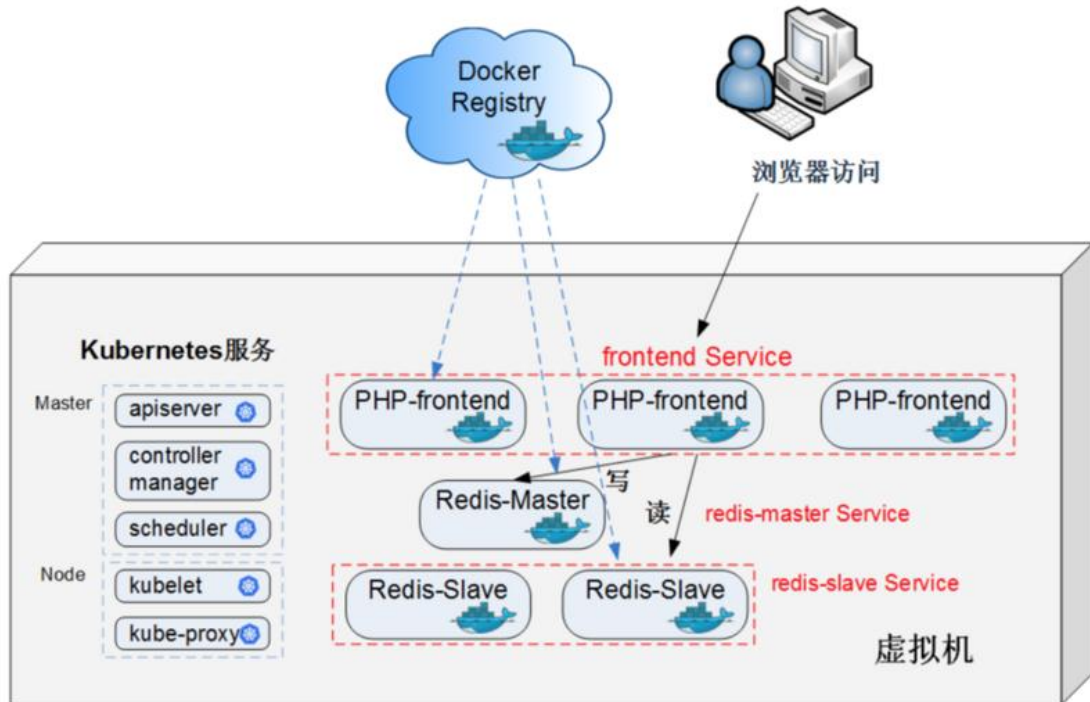
```
[root@master ~]# kubectl delete svc kubernetes-dashboard --namespace=kube-system
```

13.2 在 kubernetes 上面的集群上搭建基于 redis 和 docker 的留言簿案例

实验环境：

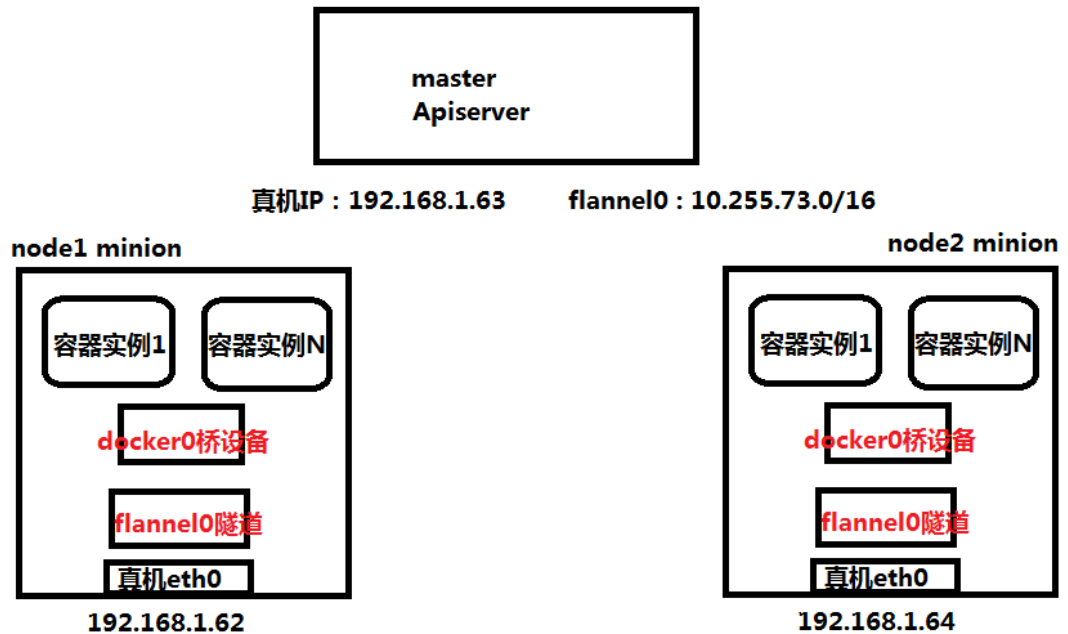
需要三个 docker 镜像：1、php-frontend web 前端镜像，2、redis master 3、redis slave

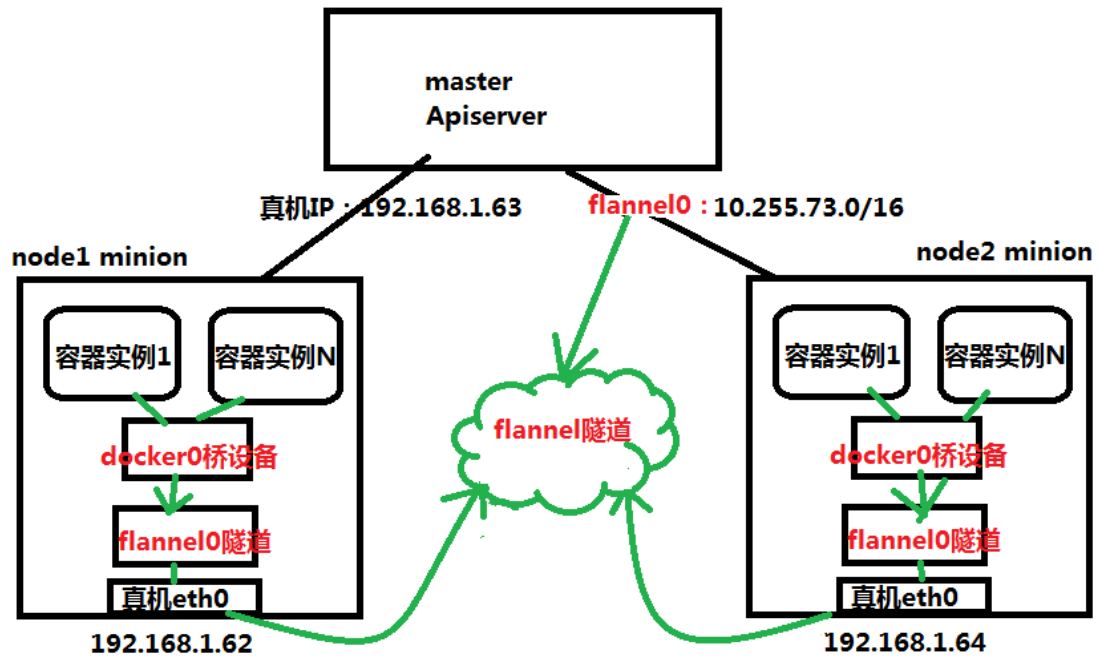
其中 web 前端通过 javascript redis api 和 redis master 交互
整体结构如下：



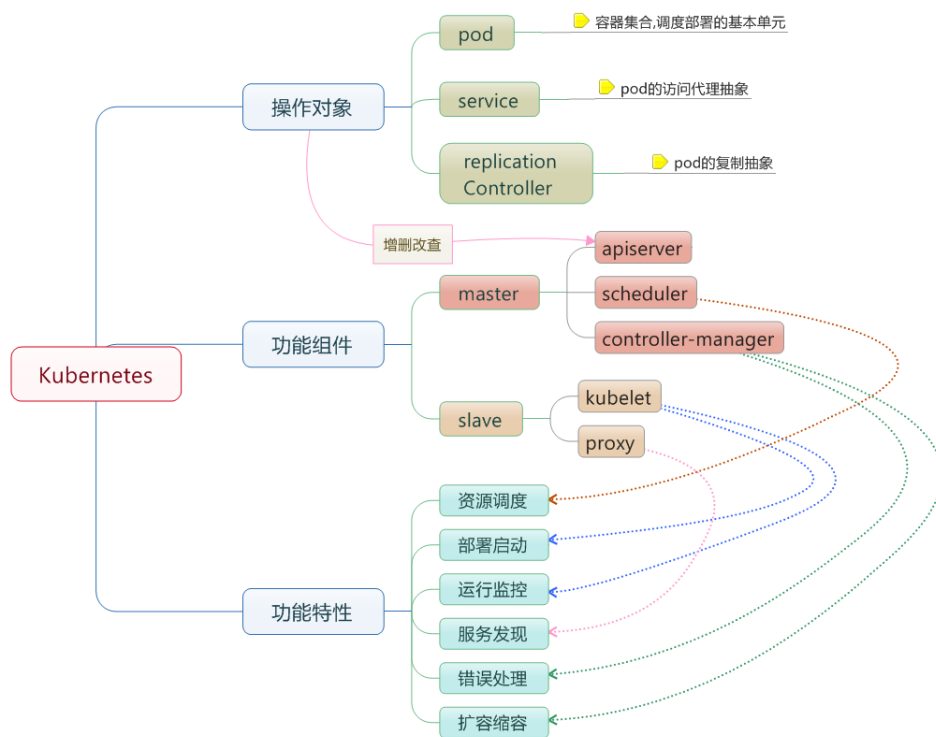
registry [ˈredʒɪstri] 记录，登记

底层网络结构：通过 flannel 实现不同物理机之间 docker 容器实例的通信。

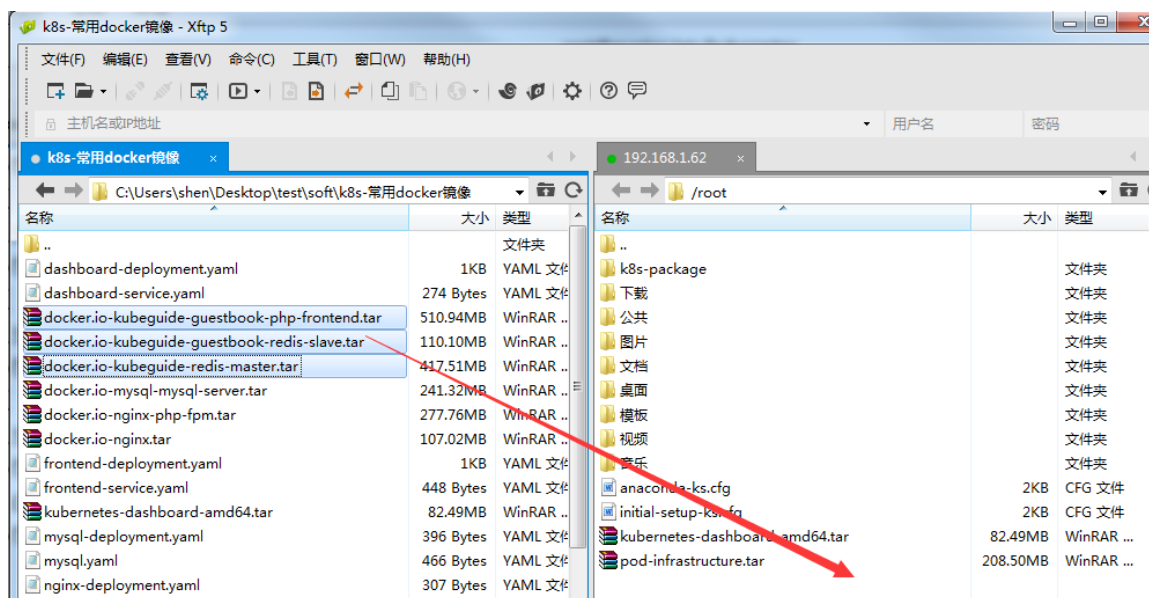




kubernetes 体系架构



上传到镜像到 node1 和 node2 上并导入



```
[root@node1 ~]# scp docker.io-kubeguide-* 192.168.1.64:/opt/
```

导入镜像：

```
[root@node1 ~]# docker load -i docker.io-kubeguide-redis-master.tar
```

```
[root@node1 ~]# docker load -i docker.io-kubeguide-guestbook-redis-slave.tar
```

```
[root@node1 ~]# docker load -i docker.io-kubeguide-guestbook-php-frontend.tar
```

```
[root@node2 ~]# docker load -i docker.io-kubeguide-redis-master.tar
```

```
[root@node2 ~]# docker load -i docker.io-kubeguide-guestbook-redis-slave.tar
```

```
[root@node2 ~]# docker load -i docker.io-kubeguide-guestbook-php-frontend.tar
```

13.2.1 创建 Redis master deployment 配置文件

Deployment 作用：通过 Deployment 描述您想要的目标状态是什么。

上传 redis-master-deployment.yaml 到 xuegod63

```
[root@master ~]# vim redis-master-deployment.yaml
```

```
apiVersion: extensions/v1beta1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: redis-master
```

```
  # these labels can be applied automatically
```

```
  # from the labels in the pod template if not set #如果没有设置，这些标签可以自动应用
```

于 POD 模板中的标签。

```
  # labels:
```

```
  #   app: redis
```

```
  #   role: master
```

```
  #   tier: backend
```

```
spec:
```

```
  # this replicas value is default
```

```
  # modify it according to your case
```

```
  replicas: 1 #此副本值默认为根据你的情况修改它。
```

```
  # selector can be applied automatically
```

from the labels in the pod template if not set #如果没有设置，选择器可以从 POD 模板中的标签自动应用。

selector:

matchLabels:

app: guestbook

role: master

tier: backend #后以运行 tier [tɪə(r)] 等级

template: #模板，如果没有设置，选择器自动使用从此模板中默认的标签

metadata:

labels:

app: redis

role: master

tier: backend

spec:

containers:

- name: master

image: docker.io/kubeguide/redis-master #这个镜像，可以改成自己的，后面

redis-slave 和 guestbook-php-frontend 相关的 yaml 文件中，我已经修改自己的镜像地址

imagePullPolicy: IfNotPresent

resources:

requests:

cpu: 100m

memory: 100Mi

ports:

- containerPort: 6379

注：我这里直接上传 redis-master-deployment.yaml 上传到/etc/kubernetes/yaml/目录下

[root@master ~]# cd /etc/kubernetes/yaml/

kubectl create -f /etc/kubernetes/yaml/redis-master-deployment.yaml

deployment "redis-master" created

[root@master kubernetes]# kubectl get deploy #查看刚创建的 rc

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
redis-master	1	1	1	0	2m

desired [dɪˈzaɪəd] 渴望的 current [ˈkʌrənt] 当前 UP-TO-DATE 运行天数

available [əˈveɪləbl] 可用 age 年龄 运行时间

[root@master kubernetes]# kubectl get pods #查看 pod

NAME	READY	STATUS	RESTARTS	AGE
redis-master-517881005-pk5nt	0/1	ContainerCreating	0	2m

ready [ˈredi] 准备好了 restarts 重启次数 age 运行时间

13.2.2 创建 redis master service 配置文件

[root@master kubernetes]# vim redis-master-service.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: redis-master # metadata.name 定义了 service 的服务名。 spec.selector 确定了
pod 对应到本服务
  labels: #这里的定义表明拥有 redis-master 标签的 pod 属于 redis-master 服务。
    app: redis
    role: master
    tier: backend
spec:
  ports:
    # the port that this service should serve on
    - port: 6379 #ports 部分中的 targetPort 属性用来确定提供该服务的容器所暴露
(EXPOSE) 的端口号, 即具体的服务进程在容器内的 targetPort 上提供服务, 而 port 属性则定义来
Server 的虚拟端口。
      targetPort: 6379
  selector:
    app: redis
    role: master
    tier: backend

```

注：其中 metadata.name 是 service 的服务名，spec.selector 确定了 pod 对应到本服务，这里的定义表明拥有 redis-master 标签的 pod 属于 redis-master 服务。另外 ports 部分中的 targetPort 属性用来确定提供该服务的容器所暴露(EXPOSE) 的端口号，即具体的服务进程在容器内的 targetPort 上提供服务，而 port 属性则定义来 Server 的虚拟端口。

把 redis-master-service.yaml 上传到/etc/kubernetes/yaml/目录下

```
[root@master ~]# cd /etc/kubernetes/
```

```
[root@master ~]# rz #上传到 linux 上
```

```
# kubectl create -f /etc/kubernetes/yaml/redis-master-service.yaml
service "redis-master" created
```

```
[root@master ~]# kubectl get svc
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	10.254.0.1	<none>	443/TCP	2h
redis-master	10.254.218.230	<none>	6379/TCP	57m

上面的运行成功后，所有 pods 都能发现 redis master 运行在 6379 端口，从 slave 到 master 流量走向会有以下两步：

- 1) 一个 redis slave 会连接到 redis master service 的 port 上
- 2) 流量会从 service 节点上的 port 到 targetPort, 如果 targetPort 未指定，默认和 port 一致
- 3、启动 replicated slave pod

虽然 redis master 是一个单独的 pod，redis slaves 是一个 replicated pod，在 Kubernetes 中，一个 Replication Controller 负责管理一个 replicated pod 的多个实例

13.2.3 创建 redis slave deployment 配置文件

上传到 redis-slave-deployment.yaml 到 /etc/kubernetes/yaml/下 :

```
[root@master kubernetes]# cd /etc/kubernetes/yaml/  
# kubectl create -f /etc/kubernetes/yaml/redis-slave-deployment.yaml  
deployment "redis-slave" created
```

```
[root@master kubernetes]# kubectl get deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
redis-master	1	1	1	0	1h
redis-slave	2	2	2	0	2m

```
[root@master kubernetes]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
redis-master-517881005-csb2t	0/1	ContainerCreating	0	1h
redis-slave-2251973062-2pfhj	0/1	ContainerCreating	0	3m
redis-slave-2251973062-ltzzs	0/1	ContainerCreating	0	3m

```
[root@master kubernetes]# kubectl get pods -o wide
```

#-o output , wide 宽, 详细显示

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
redis-master-517881005-csb2t	0/1	ContainerCreating	0	1h	<none>	node2
redis-slave-2251973062-2pfhj	0/1	ContainerCreating	0	4m	<none>	node1
redis-slave-2251973062-ltzzs	0/1	ContainerCreating	0	4m	<none>	127.0.0.1

可以看到一个 master pod 和两个 slave pod

13.2.4 创建 slave service 配置文件

上传 redis-slave-service.yaml 到/etc/kubernetes/yaml/下

```
# kubectl create -f /etc/kubernetes/yaml/redis-slave-service.yaml  
service "redis-slave" created
```

```
[root@master kubernetes]# kubectl get svc -o wide
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
kubernetes	10.254.0.1	<none>	443/TCP	2h	<none>
redis-master	10.254.218.230	<none>	6379/TCP	1h	app=redis,role=master,tier=backend
redis-slave	10.254.212.83	<none>	6379/TCP	18s	app=redis,role=slave,tier=backend

13.2.5 创建 frontend guestbook deployment 配置文件

这是一个简单的 PHP 服务, 用来和 master service (写请求) 或 slave service (读请求) 交互
上传: frontend-deployment.yaml 和 frontend-service.yaml 到/etc/kubernetes下:

```
[root@master kubernetes]# cd /etc/kubernetes/yaml/  
[root@master kubernetes]# kubectl create -f frontend-deployment.yaml  
deployment "frontend" created
```

```
[root@master kubernetes]# kubectl get deploy
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
frontend	3	3	3	0	18s
redis-master	1	1	1	1	1h
redis-slave	2	2	2	0	4m

```
[root@master kubernetes]# kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
frontend-1308123497-2mxpw	0/1	ContainerCreating	0	14s	<none>	node1
frontend-1308123497-9vdp1	0/1	ContainerCreating	0	14s	<none>	node1
frontend-1308123497-dcsjz	0/1	ContainerCreating	0	14s	<none>	node2
redis-master-517881005-1bs0b	0/1	ContainerCreating	0	4m	<none>	node2
redis-slave-2251973062-jqls2	0/1	ContainerCreating	0	1m	<none>	node2
redis-slave-2251973062-pv4v5	0/1	ContainerCreating	0	1m	<none>	node1

可以看到一个 redis master，两个 redis slave 和三个 frontend pods

注：现在显示正在创建容器，需要等这些都运行成功后，才可以访问。这个需要等一会，才会都运行起来。

```
[root@master kubernetes]# kubectl get pod
NAME                                READY    STATUS
frontend-1308123497-2mxpw           1/1     Running
frontend-1308123497-9vdp1           1/1     Running
frontend-1308123497-dcsjz           0/1     ContainerCreating
redis-master-517881005-1bs0b         1/1     Running
redis-slave-2251973062-jqls2         0/1     ContainerCreating
redis-slave-2251973062-pv4v5         1/1     Running
```

13.2.6 创建 frontend guestbook service 配置文件

和其他 service 一样，你可以创建一个 service 管理 frontend pods

上传到 frontend-service.yaml 到 linux

```
# kubectl create -f /etc/kubernetes/yaml/frontend-service.yaml
```

service "frontend" created

```
[root@master kubernetes]# kubectl get svc -o wide
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
frontend	10.254.120.178	<nodes>	80:30001/TCP	13s	app=guestbook,tier=frontend
kubernetes	10.254.0.1	<none>	443/TCP	2h	<none>
redis-master	10.254.218.230	<none>	6379/TCP	1h	app=redis,role=master,tier=backend
redis-slave	10.254.212.83	<none>	6379/TCP	3m	app=redis,role=slave,tier=backend

注：service 的 cluster-ip 是 k8s 系统中的虚拟 ip 地址，只能在内部访问。

13.2.7 查看外部网络访问 guestbook

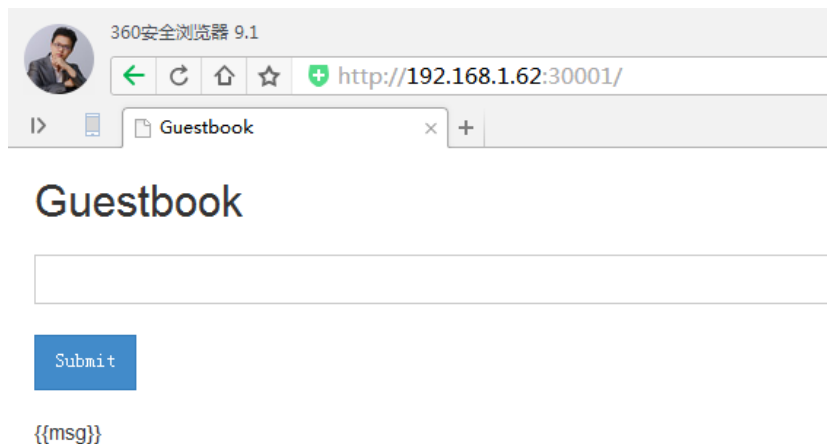
登录 node1 查看端口：

```
[root@node1 ~]# netstat -antup | grep 30001
```

tcp	0	0	192.168.1.64:37777	192.168.1.64:30001	ESTABLISHED	65331/curl
tcp	0	0	192.168.1.64:37778	192.168.1.64:30001	ESTABLISHED	590/curl

tcp6 5 0 :::30001 :::* LISTEN 57558/kube-proxy

http://192.168.1.62:30001 可以直接访问了



排错和使用心得汇总：

- 1、请核对.yaml文件的格式，有时候空格多了少了就报错
- 2、yaml文件中的image可以提前下载好，如 `docker pull kubeguide/tomcat-app:v1` 避免下载时间过程问题

- 3、使用阿里云的docker镜像，下载那些下载不成功的镜像

```
[root@node1 ~]# vim /etc/docker/daemon.json #改成以下内容
```

改： {}

为：

```
{
  "registry-mirrors": ["https://e9yneuy4.mirror.aliyuncs.com"]
}
```

```
[root@node1 ~]# systemctl daemon-reload
```

```
[root@node1 ~]# systemctl restart docker
```

```
[root@node1 ~]# docker pull docker.io/centos #再下载，就可以了。
```

- 4、可以使用 kubectl 部署 kubernetes

kubeadm 是 Kubernetes 官方提供的快速安装和初始化 Kubernetes 集群的工具。

总结：

13.1 部署 Kubernetes Dashboard web 界面

13.1.1 创建 dashboard-deployment.yaml deployment 配置文件

13.1.2 创建 dashboard-service.yaml service 配置文件

13.1.3 准备 kubernetes 相关的镜像

13.1.4 启动 dashboard 的 deployment 和 service

13.1.5 排错经验分享

13.1.6 查看 kubernetes dashboard web 界面

13.1.7 销毁 web 界面相关应用

13.2 在 kubernetes 上面的集群上搭建基于 redis 和 docker 的留言簿案例

- 13.2.1 创建 Redis master deployment 配置文件
- 13.2.2 创建 redis master service 配置文件
- 13.2.3 创建 redis slave deployment 配置文件
- 13.2.4 创建 slave service 配置文件
- 13.2.5 创建 frontend guestbook deployment 配置文件
- 13.2.6 创建 frontend guestbook service 配置文件
- 13.2.7 查看外部网络访问 guestbook

`docker.io-bestwu-docker.io-bestwu-kubernetes-dashboard-amd64-zh.tar` 导入这个，就可以实现中文版本。

<https://blog.csdn.net/huqigang/article/details/77550741>