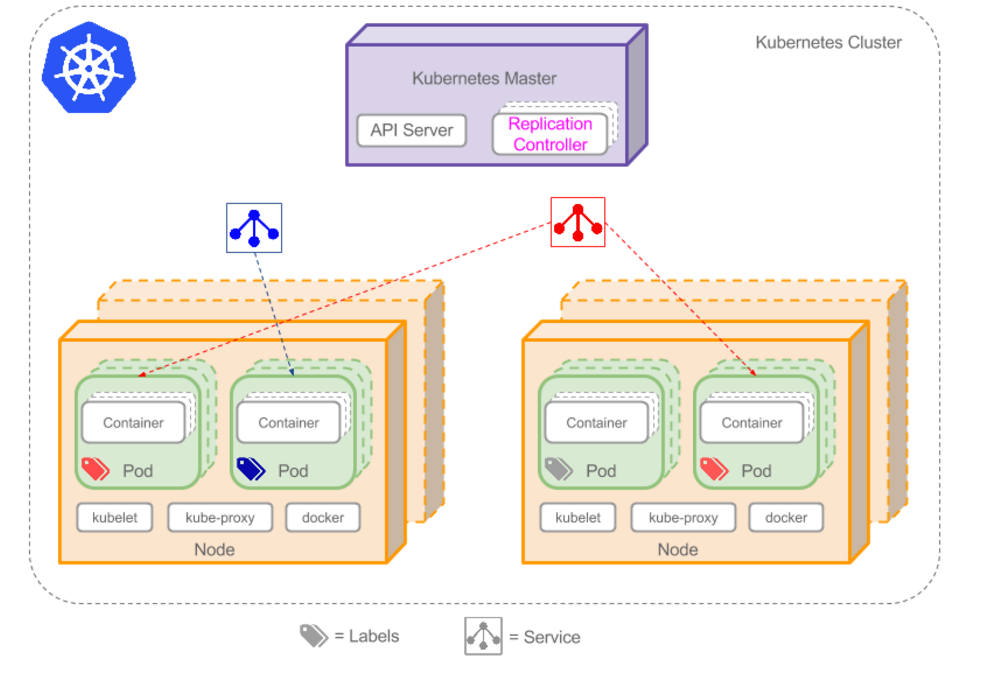
Kubernets

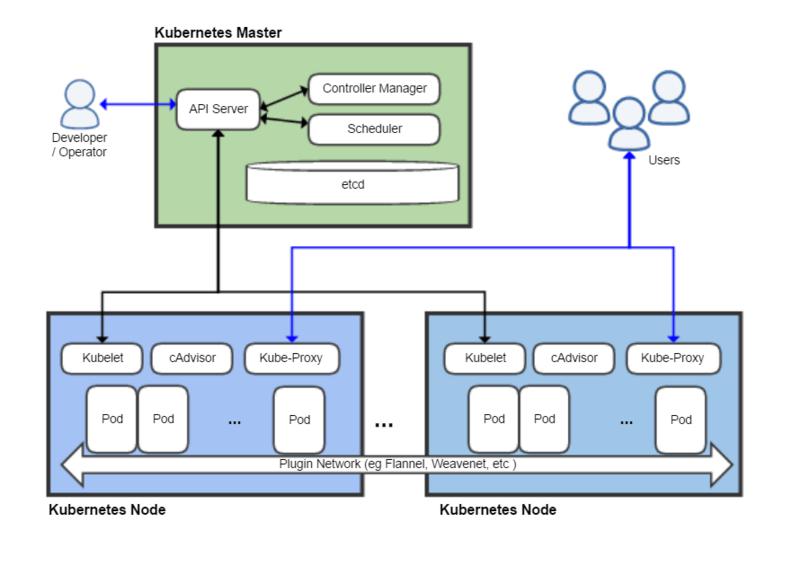
K8s是Google开源的容器集群管理系统，实现基于Docker构建容器，在主机集群之间提供一个能够自动化部署、可扩展、应用容器可运营平台，主要功能：

* 将多台Docker主机抽象为一个资源，以集群的方式管理容器，包括人物调度、资源管理、弹性伸缩、滚动升级等功能
* 使用编排系统YARM File快速构建容器集群，提供负责均衡，解决容器直接关联和通信问题
* 自动管理和修复容器，比如创建一个集群，其中共有十个容器，如果某个容器异常关闭那么会尝试重启或重启分配容器，始终保证会有十个容器在运行，否则停止其他容器。

# 1.Kubernets系统架构与概念

Kubernets集群由两类节点组成：Master和Node，在Master上运行etcd、API Server、Controller Manager和Scheduler等四个组件，负责对集群所有资源进行管理和调度。Node上运行Kubelet,Proxy和Docker Daemon三个组件，负责对本节点上的Pod生命周期进行管理，以实现服务代理的功能。另外所有节点上都可以运行Kubectl命令工具，提供Kubernetes集群管理工具，总体的系统架构如下图所示：





## 1.1 Master

Master主要是集群的管理节点，负责管理集群，提供集群的资源数据的访问入口。在Master上运行etcd、API Server、Scheduler和Controller Manager四个组件，其中后3个组件组成了Kubernets的总控中心，负责对集群中所有资源进行管控和调度。

1. etcd，轻量级、分布式的key-value的数据存储器，能够可靠的存储集群的配置数据和展现整个集群在某一个时间点的状态。其他的组件监视整个存储的变化以便更新所需要的状态。
2. API Server是一个关键组件，在HTTP协议上使用JSON为Kubernets对内外提供API服务，API Server处理和验证REST请求和更新etcd中API对象的状态。因此使得客户端能够在各个Worker节点上配置工作负载和容器
3. Scheduler，根据资源的可用性决定一个没被调度的Pod应该运行在哪个节点上。Scheduler追踪每个节点的资源使用请， 确保将调度的资源不超出剩下可用资源。为了达到这个目的，Schduler必须知道每个节点的可用资源和已经分配的资源情况
4. Controller Manager，管理DaemonSet、Replication等控制器的核心进程。这些控制器跟API服务器通信来创建、更新和删除它们所管理的资源（pod,Service端点等）

## 1.2 Node

Node是Kubernets集群架构中运行Pod的服务节点，用来承载被分配Pod的运行，是Pod运行的宿主机。在Node上运行kubelet，kube-proxy和Docker Daemon三个组件，其中前两个组件负责对本节点上的Pod生命周期进行管理，以及实现服务代理的功能。

1. Kubelet，负责每个节点的运行状态，确保节点中所有容器正在运行。其监视一个Pod的状态，该状态会通过心跳的方式发送给master。如果master侦测到一个节点失败，Replication Controller会在另一个正常的节点上启动相应的Pod
2. Kube-proxy，网络代理和负责均衡的实现，它和其他网络操作提供了服务抽象，根据ip地质和端口来路由外部请求到相应的容器
3. cAdvisor，监控和收集资源使用情况和性能指标，比如每个节点上的容器的CPU、内存和网络使用情况。

## 1.3 Pods

Kubernets的基本调度单位是Pod，各个组件的高层抽象。一个Pod由一个或者多个容器组成，部署在同一台物理主机上，相互之间能够共享资源。Kubernets中的集群内部的每一个Pod被指定唯一的IP地址，用户程序可以通过相应的端口号无冲突的连接各个Pod。Pod能够定义一个卷（Volume），比如一个本地磁盘目录或者一个网络磁盘，然后把它暴露给Pod中的容器。用户可以通过Kubernets API手动管理Pod，或者把管理工作交给一个管理器。

Pod的生命周期是通过Replication Controller进行管理，管理过程包括：通过模板进行定义，然后分配到一个Node上运行。状态包括：

* Pending，Pod定义正确，提交到Master，单其所包含的容器镜像还未完成创建。通常Master对Pod进行调度需要一些时间。
* Running，Pod已被分配到某个Node上，且其包含的所有容器镜像都已经创建完成，并成功运行
* Succeeded，Pod中所有容器都成功结束，并且不会被重启，这是Pode的最终状态
* Failed，Pod中所有容器都结束，但是有容器失败，也是Pod的一种最终状态。

对Pod的定义通过yaml或json格式的配置文件来完成，示例如下:

*apiVersion: v1*

*kind: Pod*

*metadata:*

*name: redis-slave*

*labels:*

*name: redis-slave*

*spec:*

*containers:*

*-name:slave*

*image:kubeguide/guestbook-redis-slave*

*env:*

*-name:GET\_HOSTS\_FROM*

*value: env*

*ports:*

*-containerPort: 6379*

每个Pod都有一个特殊的根容器：Puase容器，在kubelet中配置参数为：

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

## 1.4 Label

Label是Kubernets系统中的一个核心概念，以key/value键值对的形式附加到各种对象中，如Pod、Service、RC、Node等。其定义了这些对象的可识别属性，用来对他们进行管理和选择，Label可以在创建时附加到对象上，也可以在对象创建后通过API进行管理。

为对象定义好Label后，其他对象就可以使用Label Selector定义其作用对象，其定义由多个逗号分隔的条件组成，格式如下：

*"labels": {*

*"key1": "value1",*

*"key2": "value2"*

*}*

当前有两种Label Selector：

1. 基于等式（Equality-based），示例如下：

* *name = redis-slave: 选择所有包含Label中key="name"且value="redis-slave"的对象*
* *env != production: 选择所有包括Label中的key="env"且value不等于"production"的对象*

1. 基于集合（Set-based），使用集合操作等进行选择

* *name in (redis-master, redis-slave): 选择所有包含Label中的key="name"且value="redis-master"或"redis-slave"的对象；*
* *name not in (php-frontend): 选择所有包含Label中的key="name"且value不等于"php-frontend"的对象。*

在某些对象需要对另一些对象进行选择时，可以将多个Label进行组合，使用逗号分隔，基于等式的LabelSelector和基于集合的Label Selector可以任意组合，示例：

* *name=redis-slave,env!=production*
* *name not in (php-frontend),env!=production*

## 1.5 Replication Controller

RC确保任何时候Kubernets集群中有指定数量的pod复本（replicas）在运行，当Pod失败、被删除或被终结时，RC会自动创建新的Pod来保证副本数量。RC使用预先定义的pod模板创建pods，一旦创建成功，pod模板和创建的pods没有任何关联，可以修改pod模板而不会对已创建的pods有任何影响。

*apiVersion: v1*

*kind: ReplicationController*

*metadata:*

*name: nginx*

*spec:*

*replicas: 3*

*selector:*

*app: nginx*

*template:*

*metadata:*

*name: nginx*

*labels:*

*app: nginx*

*spec:*

*containers:*

*- name: nginx*

*image: nginx*

*ports:*

*- containerPort: 80*

上述配置运行三个nginx server。

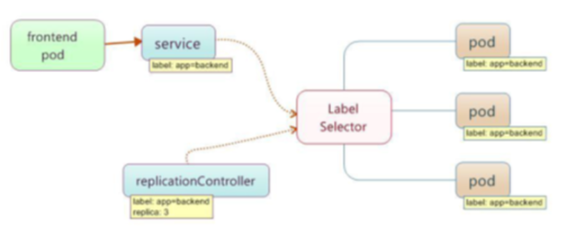
## 1.6 Deployment

Deployment提供一共更加加到的更新RC和Pod的机制，在Deployment中描述了所期望的集群状态，Deployment Controller会将现在的集群状态在一个可控的速度下逐步更新成所期望的集群状态。

Deployment主要职责同样是为了保证Pod的数量和健康，Deployment内部使用RC来实现，基本的功能与RC一样。一个Pod的创建、调度、绑定节点及在目标Node上启动对应的容器这一完整的过程需要一定时间，所有期待系统启动N个Pod副本的目标状态，实际上是一个连续变化的部署过程，因此相对于RC的最大升级是可以随时知道当前Pod的部署进度。

## 1.7 Service

Service是Kubernets最核心的概念，通过创建Service，可以为一组具有相同功能的容器应用提供一个统一的入口地址，并将请求进行负载分发到后端的各个容器应用上。Pod,RC与Service之间的关系如下图：



前端的应用Pod通过这个入口地址访问其背后的一组由Pod副本组成的集群实例，Service与其后端的Pod副本集群之间通过Label Selector来实现无缝对接 ，RC的作用实际上就是保证Service的服务能力和服务质量处于预期标准。

运行在Node上的Kube-proxy进程实际上是一个智能的软件负载均衡器，把对Service的请求转发到后端的某个Pod实例上，并在内部实现服务的负载均衡与会话保持机制。

对Service的定义文件模板的各属性如下所示：





示例如下：

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: redis-slave*

*labels:*

*name: redis-slave*

*spec:*

*ports:*

*- port: 6379*

*selector:*

*name: redis-slave*

通过上面的定义，Kubernets创建一个名为redis-slave的服务，并在6379端口上监听，spec.selector的定义表示该Service将包含所有具有name=redis-slave的Label的Pod。

在Pod正常启动后，系统将会根据Service的定义创建出与Pod相对应的Endpoint对象，以建立起Service与后端Pod的对象关系。随着Pod的创建、销毁、Endpoint对象也将被更新。Endpoint对象主要有Pod的IP地址和容器所需监听的端口号组成。

## 1.8 Namespace

Namespace用于实现多租户的资源隔离，将集群内部的资源对象分配到不同的Namespace中，形成逻辑上分组的不同项目、小组或用户组，便于不同的分组在共享使用整个集群的资源的同时还能被分别管理。Kubernetes集群在启动后，会创建一个名为default的NameSpace，NameSpace定义比较简单：

*apiVersion:v1*

*kind:Namespace*

*metadata:*

*name: development*

一旦创建了Namespace，可以在创建资源对象时就可以指定这个资源对象属于哪个Namespace。多租户的资源隔离，可以限定不同租户能占用的资源，如CPU使用量、内存使用量等。

## 1.9 Volume && Persistent Volume

Volume是Pod中能够被多个容器访问的共享目录，与Docker中的Volume类似但是有区别。Kubernetes中的Volume定义在Pod上，然后被一个Pod中的多个容器挂载到具体的文件目录下，Kubernetes中的Volumn与Pod生命周期相同，当其中的容器终止或者重启时，不会丢失。目前Kubernetes支持的Volume类型包括：awsElasticBlockStore,Cephfs，ConfigMap，Flocker、nfs、local等。

在Pod上生命一个Volume，然后在Docker容器中引用该Volumn并Mount到容器的某个目录上，举例如下：

*apiVersion: v1*

*kind: Pod*

*metadata:*

*name: test-pd*

*spec:*

*containers:*

*- image: k8s.gcr.io/test-webserver*

*name: test-container*

*volumeMounts:*

*- mountPath: /test-pd*

*name: test-volume*

*volumes:*

*- name: test-volume*

*hostPath:*

# directory location on host

*path: /data*

# this field is optional

*type: Directory*

在Pod上挂载宿主主机上的文件和目录。

该中类型的Volume是定义在Pod之上，属于计算资源的一部分，但是实际上，“网络存储”是相对独立于计算资源而存在的一种实体资源，例如使用虚机的时候，先定义一个网络存储，然后从中划出一个网盘挂载到虚拟机上。Persistent Volume（PV）及Persistent Volume Claim也是起到类似的作用。

PV可以理解成Kubernetes集群中的某个网络存储对于的一块存储，与Volume类似，但是又有下面的区别：

* PV只能是网络存储，不属于任何Node，但是可以在Node上访问。
* PV并不是定义在Pod上，而是独立于Pod之外定义
* PV只支持下面的类型：GCE Persistence Disks,NFS,GlusterFS，AWS ElasticBlockStore等

下面是NFS类型的PV的一个yaml定义文件，声明需要5Gi的存储空间：

*apiVersion:v1*

*kind:PersistentVolume*

*metadata:*

*name: pv0003*

*spec:*

*capacity:*

*storage:5GI*

*accessModes:*

*-ReadWriteOnce*

*nfs:*

*path: /somepath*

*server:172.17.0.2*

比较重要的是pv的accessModes属性。如果某个Pod想申请某种条件的PV，则定义一个PersistentVolumeClaim对象：

*Kind: PersistentVolumeClaim*

*apiVersion: v1*

*metadata:*

*name: myClaim*

*….*

在Pod的Volumn定义中引用上述PVC即可：

*volumnes:*

*-name: mypd*

*persistentVolumeClaim:*

*claimName: myClaim*

PV的状态包括：

* Available，空闲状态
* Bound，已经绑定到某个Pvc之上
* Released，定义的Pvc已经删除，但资源还没有被集群收回
* Failed: Pv自动回收失败。

# 2.Kubernets安装

## 2.1 安装Kubernets Master

通过yum安装kubernets

*$yum  install -y kubernetes etcd ntp.x86\_64 flannel*

参数的配置：

1. 配置/etc/etcd/etcd.conf，修改以下参数：

*ETCD\_LISTEN\_CLIENT\_URLS="*[*http://0.0.0.0:2379*](http://0.0.0.0:2379/)*"*

*ETCD\_ADVERTISE\_CLIENT\_URLS="*[*http://0.0.0.0:2379*](http://0.0.0.0:2379/)*"*

1. 配置/etc/kubernetes/config，修改以下参数：

*KUBE\_MASTER="--master=*[*http://10.254.20.110:8080*](http://10.254.20.110:8080/)*"*

1. 配置/etc/kubernetes/apiserver，修改以下参数：

*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://10.254.20.110:2379*](http://10.254.20.110:2379/)*"*

*KUBE\_SERVICE\_ADDRESSES="--service-cluster-ip-range=172.8.0.0/13"*

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

KUBE\_ADMISSION\_CONTROL中去掉ServiceAccount参数。

1. 启动服务

*for SRV   in ntpd etcd kube-apiserver kube-scheduler kube-controller-manager;*

*do*

*systemctl start $SRV*

*systemctl enable $SRV*

*systemctl status $SRV*

*done*

1. 在MASTER节点上修改etcd的配置，设定Minion中的flannel所使用的子网范围

*etcdctl   mk /*[*atomic.io/network/config*](http://atomic.io/network/config)*'{"Network":"172.17.0.0/16",   "SubnetMin": "172.17.1.0", "SubnetMax":   "172.17.254.0"}'*

1. 在MASTER节点上的flannel配置/etc/sysconfig/flanneld，设定etcd的相关信息，其中10.254.20.110为Master的ip地址

*FLANNEL\_ETCD\_ENDPOINTS="*[*http://10.254.20.110:2379*](http://10.254.20.110:2379/)*"*

*FLANNEL\_ETCD="*[*http://10.254.20.110:2379*](http://10.254.20.110:2379/)*"*

执行启动命令：systemctl start flanneld

## 2.2 安装Kubernets Slave(Minion)

使用yum安装k8s软件

*yum install -y ntp.x86\_64 flannel   kubernetes*

配置/etc/kubernetes/config的以下参数：

1）/etc/kubernetes/config

*KUBE\_MASTER="--master=*[*http://10.254.20.110:8080*](http://10.254.20.110:8080/)*"*

2）配置/etc/kubernetes/kubelet的以下参数：

*KUBELET\_ADDRESS="--address=0.0.0.0"*

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

*KUBELET\_HOSTNAME="--hostname-override=10.254.20.112"   （各自IP）*

*KUBELET\_API\_SERVER="--api-servers=*[*http://10.254.20.110:8080*](http://10.254.20.110:8080/)*"*

3）修改Minion节点上的flannel的配置/etc/sysconfig/flanneld，设定etcd的相关信息，其中10.254.20.110为master的ip地址

*FLANNEL\_ETCD\_ENDPOINTS="*[*http://10.254.20.110:2379*](http://10.254.20.110:2379/)*"*

*FLANNEL\_ETCD="*[*http://10.254.20.110:2379*](http://10.254.20.110:2379/)*"*

4）启动相关服务

*for SRV   in ntpd flanneld docker kube-proxy kubelet;*

*do*

*systemctl start $SRV*

*systemctl enable $SRV*

*systemctl status $SRV*

*done*

## 2.3 安装Kubernets Dashboard

通过以上步骤，已经搭建好了K8S集群，查看状态如下：

*# kubectl get nodes -o wide*

*NAME STATUS AGE EXTERNAL-IP*

*10.139.4.106 Ready 1h <none>*

*10.139.4.107 Ready 1h <none>*

1. 在所有的Minion上执行:

*docker   pull clouding/kubernetes-dashboard-amd64:v1.5.0*

1. 然后获取kerbernetes-dashboard.yaml启动文件

*wget*[*https://rawgit.com/kubernetes/dashboard/master/src/deploy/kubernetes-dashboard.yaml*](https://rawgit.com/kubernetes/dashboard/master/src/deploy/kubernetes-dashboard.yaml)

注意：在从官网上下载对应版本的keburnets-dashboard配置文件，否则解析会有问题。

修改kubernetes-dashboard.yaml以下字段：

*spec:*

*containers:*

*- name: kubernetes-dashboard*

*image:*[*docker.io/clouding/kubernetes-dashboard-amd64:v1.5.0*](http://docker.io/clouding/kubernetes-dashboard-amd64:v1.5.0)

*imagePullPolicy:   IfNotPresent*

*ports:*

*- containerPort: 9090*

*protocol: TCP*

*args:*

*# Uncomment the following line to   manually specify Kubernetes API server Host*

*# If not specified, Dashboard will   attempt to auto discover the API server and connect*

*# to it. Uncomment only if the default   does not work.*

*- --apiserver-host=http://10.139.4.107:8080*

1. 启动kubernetes dashboard服务

*kubectl   create -f kubernetes-dashboard.yaml*

1. Dashboard的执行状态

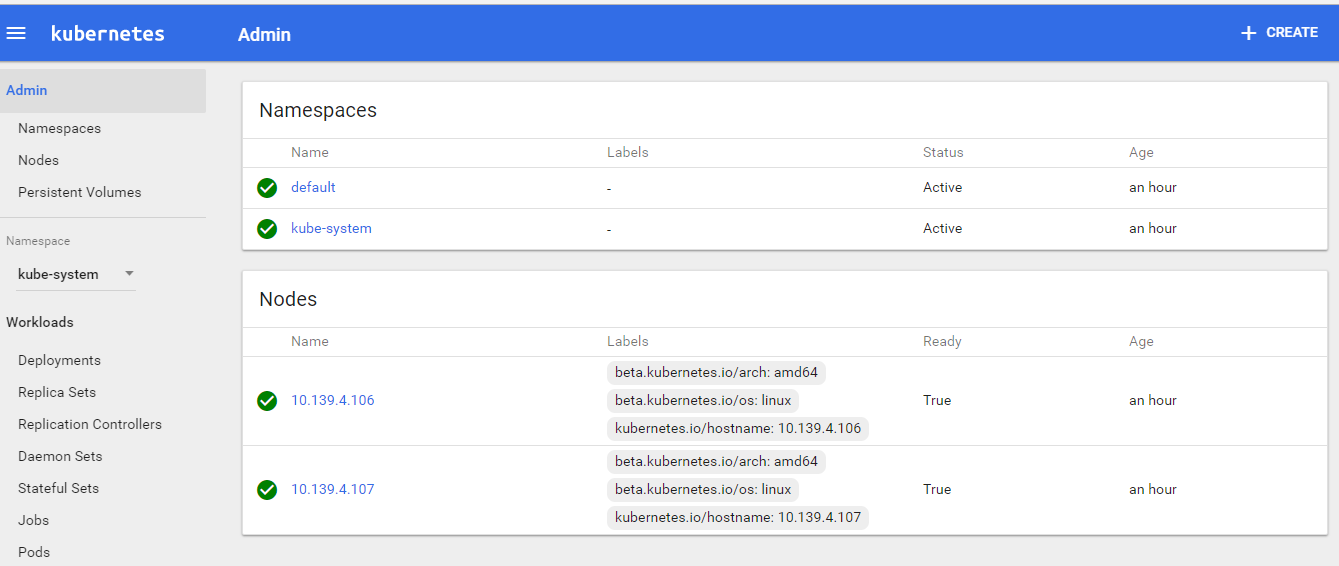
*# kubectl get pods --all-namespaces*

*NAMESPACE NAME READY STATUS RESTARTS AGE*

*kube-system kubernetes-dashboard-246849558-wxfvm 1/1 Running 0 52m*

当状态为RUNNING时，打开浏览器，在地址栏中输入：[*http://10.139.4.107:8080/ui/*](http://10.254.20.110:8080/ui/)

页面如下：



至此安装Kubernetes成功。

# 3. Kubernetes的使用

创建Kubernets应用有三种方式，如下所示：

## 3.1 kubectl run命令

直接通过kubectl run命令来创建一个应用，命令格式如下：

*kubectl run --image=nginx nginx-app --port=81 --env="DOMAIN=cluster"*

输出如下：

*deployment "nginx-app" created*

查看如下：

*$kubectl get deployments*

*NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE*

*my-nginx 1 1 1 1 1d*

*nginx-app 1 1 1 1 46s*

*# kubectl get pods*

*NAME READY STATUS RESTARTS AGE*

*nginx-app-2003097476-4p3q6 1/1 Running 0 1m*

使用expose，将该rc上的nginx创建service，格式如下：

*# kubectl expose deployment nginx-app --port=85 --name=nginx-http*

*service "nginx-http" exposed*

查看service，如下 ：

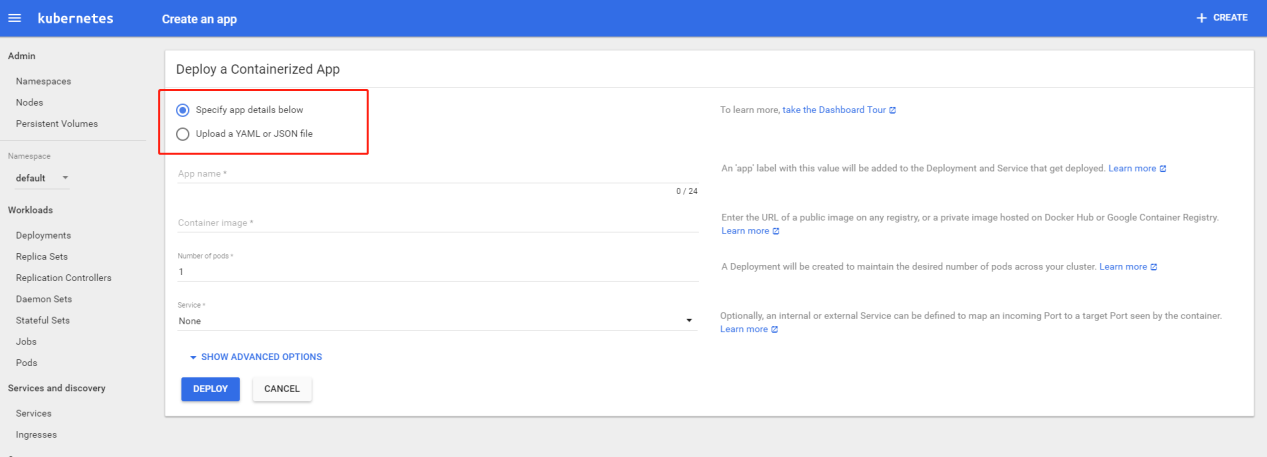
*# kubectl get services*

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

*nginx-http 172.11.121.137 <none> 85/TCP 14s*

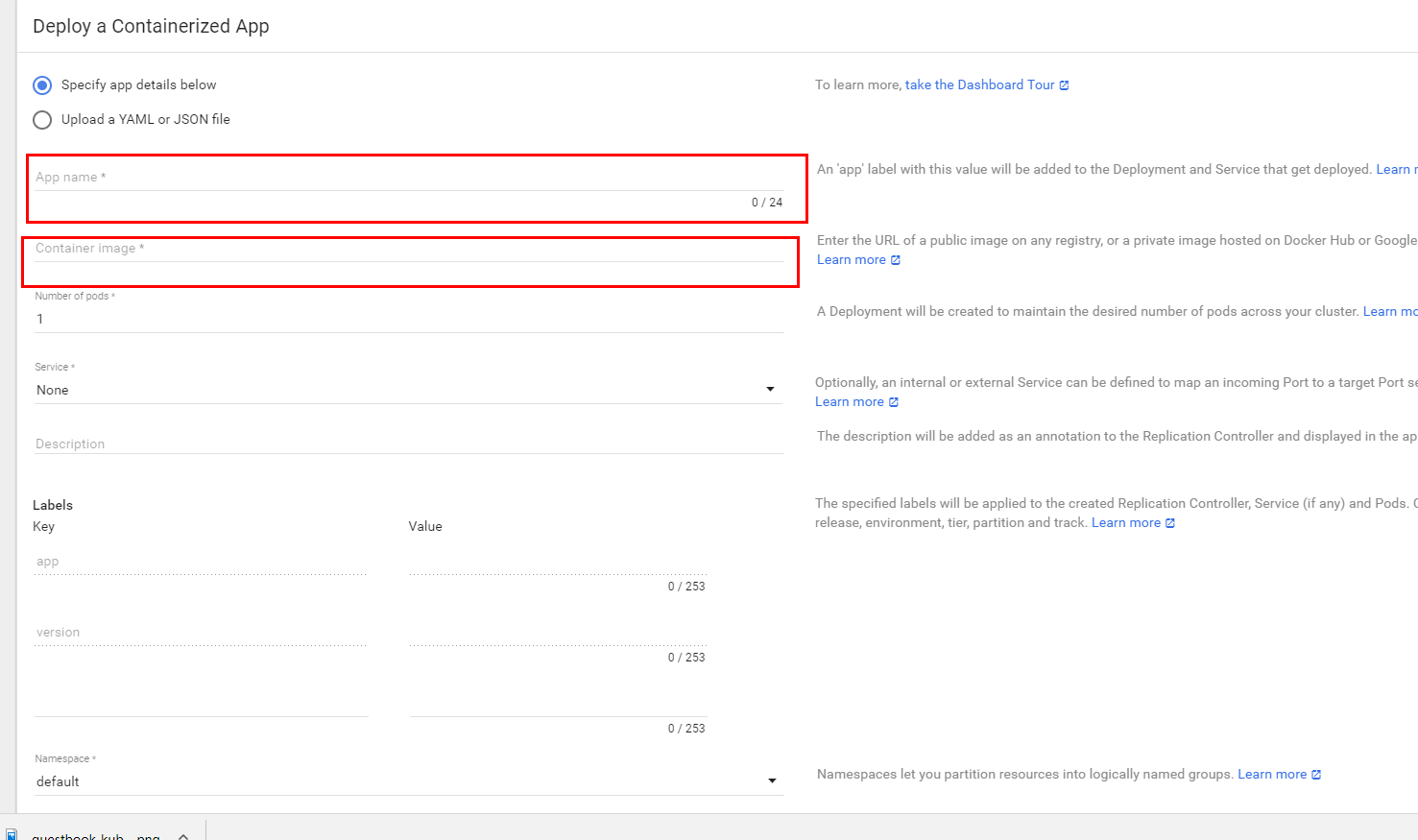
## 3.2 通过dashboard

第二种方式是使用dashboard，配置页面如下：



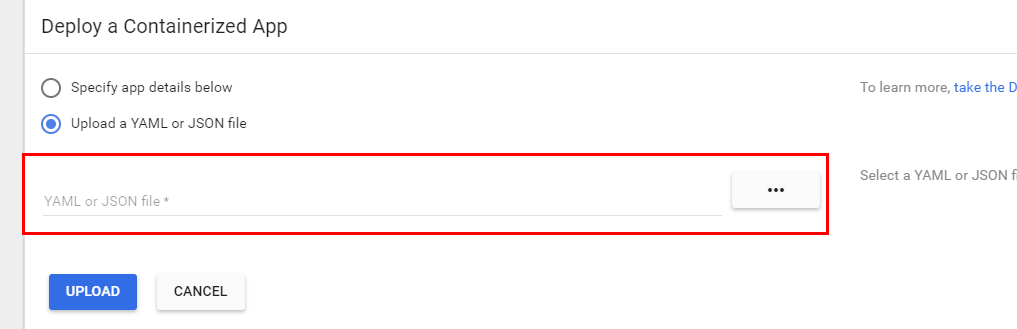
有两种选项：

* Specify app details



在页面上指定具体的参数

方式2：直接上传本地的YAML配置文件



## 3.3 yaml

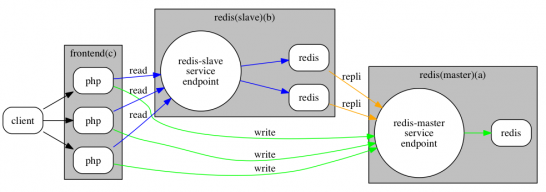
通过yaml文件的定义，通过kubectl create命令来创建应用，格式如下：

*kubectl create -f FILENAME [flags]*

通过-f 参数指定具体组件的定义，这种方式也是最常用的方式，在下面进行详细介绍。

# 4.Gusetbook示例: yaml

下面是一个通过3个微服务组成的留言板（Guestbook）系统，进一步的深入Pod,RC,Service等核心对象用法和技巧。系统启动后在网页中显示一条Hello World留言，系统架构如下：



基于PHP+REDIS的分布式Web应用，前端PHP Web网站通过访问后端Redis来完成用户留言的查询和添加等功能。同时，Redis以Master+Slave的模式进行部署，实现数据读写分离。Web层基于PHP的Apache服务，启动三个实例组成集群，为客户端对网站的访问提供负载均衡。Redis Master启动一个实例用户写操作，Slave启动两个实例用于读操作，Redis Master与Slave之间具备数据同步机制。

示例使用使用三个Docker镜像：

*docker pull docker.io/kubeguide/guestbook-redis-slave*

*docker pull docker.io/kubeguide/guestbook-php-frontend*

*docker pull docker.io/kubeguide/redis-master*

在kubelet节点上下载这三个镜像。

## 4.1 创建redis-master

定义Service，然后定义RC来创建和控制相关联的Pod，或者先定义RC来创建Pod，然后定义与之关联的Service。

首先创建redis-master的RC定义文件：redis-master-controller.yaml，内容如下：

*apiVersion: v1*

*kind: ReplicationController*

*metadata:*

*name: redis-master*

*labels:*

*name: redis-master*

*spec:*

*replicas: 1*

*selector:*

*name: redis-master*

*template:*

*metadata:*

*labels:*

*name: redis-master*

*spec:*

*containers:*

*- name: master*

*image: docker.io/kubeguide/redis-master*

*ports:*

*- containerPort: 6379*

其中kind字段为ReplicationController，表示这是一个RC；spec.selector是RC的Pod选择器，即监控和管理拥有这些标签的Pod实例，确保当前集群上仅有一个POD实例在运行。

创建redis-master命令如下：

*kubectl create -f redis-master-controller.yaml*

系统提示创建成功，用Kubectl查看刚刚创建的redis-master：

*kubectl get rc*

输出如下:

*NAME DESIRED CURRENT READY AGE*

*redis-master 1 1 1 29m*

接下来运行pod命令，查询pod:

*$kubectl get pod*

*NAME READY STATUS RESTARTS AGE*

*redis-master-xt7kx 1/1 Running 0 29m*

至此redis-master已经创建并成功运行。然后创建与之关联的Service（redis-master-server.yaml），定义如下：

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: redis-master*

*labels:*

*name: redis-master*

*spec:*

*ports:*

*- port: 6379*

*targetPort: 6379*

*selector:*

*name: redis-master*

其中metadata.name是Service的服务名，spec.selecto确定选择哪些pod，port属性定义的是Service的虚拟机端口号，targetPort是指定后面Pod内容器应用监听的端口号。

创建service命令如下：

*kubectl create -f redis-master-service.yaml*

系统提示Service “redis-master” created创建成功，运行kubectl命令可以查看刚刚创建的service:

*kubectl get services*

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

*redis-master 172.12.118.222 <none> 6379/TCP 25m*

其中redis-master服务被分配了一个值为172.12.118.222的虚拟IP地址，其他新创建的pod可以通过这个虚拟ip地址+端口（6379）来访问这个服务。

## 4.2 创建redis-slave RC和Service

上一步已经启动了redis-master服务，下面创建redis-slave，两个副本，每个副本的redis进程都与redis-master进行数据同步，与redis-master共同组成一个具备读写分离能力的Redis集群。Guestbook的PHP网页通过访问redis-slave服务来读取留言数据，redis-slave的RC文件：redis-slave-controller.yaml，完成内容如下：

*apiVersion: v1*

*kind: ReplicationController*

*metadata:*

*name: redis-slave*

*labels:*

*name: redis-slave*

*spec:*

*replicas: 2*

*selector:*

*name: redis-slave*

*template:*

*metadata:*

*labels:*

*name: redis-slave*

*spec:*

*containers:*

*- name: slave*

*image: docker.io/kubeguide/guestbook-redis-slave*

*env:*

*- name: GET\_HOSTS\_FROM*

*value: env*

*ports:*

*- containerPort: 6379*

在容器的配置部分设置一个环境变量GET\_HOSTS\_FROM=env，从环境中获取redis-master服务的ip地址。在创建redis-slave pod时，系统自动在容器内部生成之前已经创建好的redis-master service相关的环境变量。运行kubectl create命令创建RC:

*kubectl create -f redis-slave-controller.yaml*

通过kubectl get rc可以查看到创建好的rc。然后创建redis-slave服务，定义如下：

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: redis-slave*

*labels:*

*name: redis-slave*

*spec:*

*ports:*

*- port: 6379*

*selector:*

*name: redis-slave*

运行kubectl创建service：

*kubectl create -f redis-slave-service.yaml*

通过kubectl查看service:

*# kubectl get services*

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

*kubernetes 172.8.0.1 <none> 443/TCP 20h*

*redis-master 172.12.118.222 <none> 6379/TCP 36m*

*redis-slave 172.11.134.47 <none> 6379/TCP 31m*

## 4.3创建fronted RC和Service

定义frontend的RC配置文件:frontend-controller.yaml，内容如下：

*apiVersion: v1*

*kind: ReplicationController*

*metadata:*

*name: frontend*

*labels:*

*name: frontend*

*spec:*

*replicas: 3*

*selector:*

*name: frontend*

*template:*

*metadata:*

*labels:*

*name: frontend*

*spec:*

*containers:*

*- name: fronted*

*image: docker.io/kubeguide/guestbook-php-frontend*

*env:*

*- name: GET\_HOSTS\_FROM*

*value: env*

*ports:*

*- containerPort: 80*

在容器的配置部分设置一个环境变量GET\_HOSTS\_FROM=env,从环境变量中获取redis-master和redis-slave的ip地址信息。运行kubectl create命令创建RC:

*kubectl create -f frontend-controller.yaml*

最后创建frontend Service，定义如下：

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: frontend*

*labels:*

*name: frontend*

*spec:*

*type: NodePort*

*ports:*

*- port: 80*

*nodePort: 30001*

*selector:*

*name: frontend*

通过kubectl命令创建Service

*kubectl create -f frontend-service.yaml*

主要的目的是使用service的Nodeport给Kubernets集群外的Service映射一个外网可以访问的端口。至此所有的服务都以创建完毕：

*# kubectl get services*

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

*frontend 172.12.75.63 <nodes> 80:30001/TCP 39m*

*kubernetes 172.8.0.1 <none> 443/TCP 20h*

*redis-master 172.12.118.222 <none> 6379/TCP 52m*

*redis-slave 172.11.134.47 <none> 6379/TCP 47m*

*# kubectl get pods*

*NAME READY STATUS RESTARTS AGE*

*frontend-3pxg2 1/1 Running 0 42m*

*frontend-h47q3 1/1 Running 0 42m*

*frontend-th8vq 1/1 Running 0 42m*

*my-nginx-379829228-tzgbc 1/1 Running 0 20h*

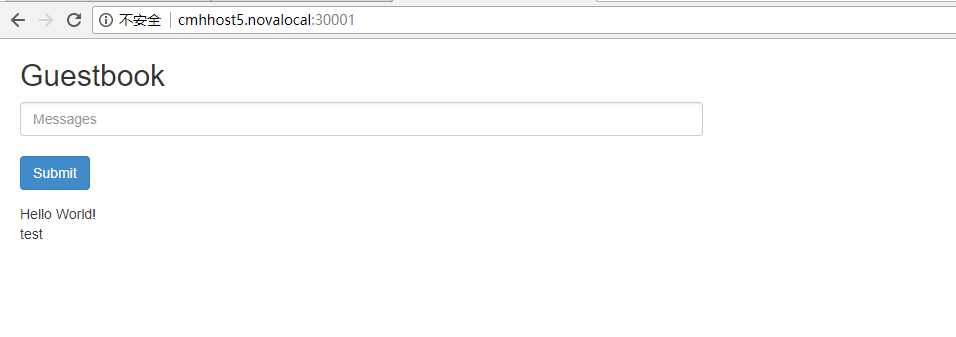
*redis-master-xt7kx 1/1 Running 0 1h*

*redis-slave-3q2tt 1/1 Running 0 49m*

*redis-slave-sr6pp 1/1 Running 0 49m*

## 4.4 通过浏览器访问frontend页面

在地址栏中输入：http://虚拟机ip:30001，打开页面如下：



输入一条留言test，将会在原留言下方显示新的留言，说明这条留言已经被加入到了Redis数据库中。

参考文献：

1. K8S安装rpm包：可以直接通过rpm包直接安装

https://dl.fedoraproject.org/pub/fedora/linux/development/rawhide/Everything/x86\_64/os/Packages/k/

https://www.csdn.net/article/2014-12-24/2823292-Docker-Kubernetes

http://10.254.0.20:8090/pages/viewpage.action?pageId=27926764

1. K8s中文文档：https://www.kubernetes.org.cn/doc-44
2. 通过kubeadm安装k8s集群：https://github.com/qianlei90/Blog/issues/28

https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

https://kubernetes.io/docs/setup/independent/install-kubeadm/

http://223.105.0.132:8090/pages/viewpage.action?pageId=30800059

1. Docker网络：<http://dockone.io/article/2599>