# [**在ubuntu上部署Kubernetes管理docker集群**](https://segmentfault.com/a/1190000002620961)

（V1.1）

|  |  |  |
| --- | --- | --- |
| 修改日期 | 修改人 | 修改内容 |
| **2016-09-18** | 陈伟 | 初稿 |
| **2016-09-18** | 陈伟 | 修改deployAddons.sh  先查询aliregistrykey是否存在，然后添加secret |
| **2016-09-18** | 陈伟 | 在4.2.5 添加 daemonset 方式的fluentd-logging |

本文通过实际操作来演示Kubernetes的使用，因为环境有限，集群部署在本地3个ubuntu上，主要包括如下内容：

* 部署环境介绍，以及Kubernetes集群逻辑架构
* 安装部署flannel
* 安装部署Etcd和Kubernetes的各大组件
* 演示Kubernetes管理容器和服务

关于 Kubernetes 系统架构及组件介绍见[这里](http://segmentfault.com/blog/seanlook/1190000002620946)。

## **1. 部署环境及架构**

* 操作系统: ubuntu 14.04 x86\_64
* Kubernetes: v1.3.5
* Etcd版本: 2.3.1
* Docker版本: 1.12.0
* flanneld版本: 0.5.5
* bridge-utils
* 服务器信息：

| Role | Hostname | IP Address |
| --- | --- | --- |
| APIServer | k8s.master | 172.16.2.234 |
| Minion | k8s-minion1 | 172.16.2.233 |
| Minion | K8s-minion2 | 172.16.5.238 |

在详细介绍部署Kubernetes集群前，先给大家展示下集群的逻辑架构。从下图可知，整个系统分为两部分，第一部分是Kubernetes APIServer，是整个系统的核心，承担集群中所有容器的管理工作；第二部分是minion，运行Container Daemon，是所有容器栖息之地，同时在minion上运行flannel程序，负责minions之间Pod的网络通信工作。

解决dubbo等自带服务发现机制的框架需要pod通信

各节点直接的网络通信可以使用路由自动发现机制,各节点执行操作(参考<http://dockone.io/article/466>)

docker run -itd --name=router --privileged --net=host index.alauda.cn/georce/router

## **2. 准备工作**

## **2.1 设置各个节点root免密登录**

1. #每个节点的机子都要执行
2. sudo su - root
3. apt-get --assume-yes install openssh-client
4. ssh-keygen
5. #执行ssh-keygen 时无需输入，一直按enter即可
6. #将各个节点.ssh目录下的id\_rsa.pub中的值复制并添加到k8s-master中.ssh/authorized\_keys中
7. vim .ssh/authorized\_keys
8. #然后拷贝到各个节点
9. scp -r .ssh/authorized\_keys [root@172.16.2.233](mailto:root@172.16.2.233):.ssh
10. scp -r .ssh/authorized\_keys [root@172.16.5.23](mailto:root@172.16.5.23):.ssh

## **2.2 安装docker1.12+,bridge-utils**

**#安装**docker 参考(<https://docs.docker.com/v1.8/installation/ubuntulinux/>)

或者自己创建脚本，一键安装,脚本内容如下

|  |
| --- |
| vim install-docker.sh |
| #!/bin/bash  #Run as root user  echo " Checking user..."  sleep 1  if [ $UID != 0 ];then  echo "please run as root"  exit  fi  sleep 1  export APT\_INSTALL='sudo apt-get --assume-yes install'  export APT\_UPDATE='sudo apt-get --assume-yes update'  sudo chown $USER:$USER /etc/resolv.conf  echo "nameserver 223.5.5.5" >> /etc/resolv.conf  sudo chown root:root /etc/resolv.conf  $APT\_INSTALL git  #install bridge-utils $APT\_INSTALL bridge-utils $APT\_INSTALL apt-transport-https ca-certificates  apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D  echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main" > /etc/apt/sources.list.d/docker.list  $APT\_UPDATE  apt-get purge lxc-docker  apt-cache policy docker-engine  $APT\_UPDATE  $APT\_INSTALL linux-image-extra-$(uname -r) linux-image-extra-virtual  $APT\_UPDATE  $APT\_INSTALL docker-engine  service docker start  docker -v  groupadd docker  usermod -aG docker $USER  $APT\_INSTALL python-pip  sudo pip install docker-compose  sudo docker-compose -v |

#授权sudo chmow 755 install-docker.sh

#执行安装即可

./install-docker.sh

## **3. 安装Kubernetes**

### **3.1 下载安装kubernetes各组件（用我修改的1.3.5版本最好各种配置已经修改完,只需修改kubernetes/k8s-cluster/github/kubernetes/cluster/ubuntu/config-default.sh）**

可以自己从源码编译kubernetes（需要安装golang环境），也可以选择编译好的二进制版本（v1.3.5）下载，为了方便后面启动或关闭kubernetes组件，我们同时下载二进制包和源码包：

# wget

wget http://gitlab.vpclub:8022/deployment/kubernetes.git

cd kubernetes/k8s-cluster/github/kubernetes/cluster/ubuntu

#如需更新源码包，只需更新kubernetes/k8s-cluster/github/kubernetes/cluster/src　文件夹下面的相应源码包即可

＃编辑节点配置

vim config-default.sh

|  |
| --- |
| #!/bin/bash  # Copyright 2015 The Kubernetes Authors.  # Define all your cluster nodes, MASTER node comes first"  # And separated with blank space like <user\_1@ip\_1> <user\_2@ip\_2> <user\_3@ip\_3>  export nodes=${nodes:-"root@172.16.2.234 root@172.16.2.233"}  # Define all your nodes role: a(master) or i(minion) or ai(both master and minion),  # Roles must be the same order with the nodes.  roles=${roles:-"a i"}  # If it practically impossible to set an array as an environment variable  # from a script, so assume variable is a string then convert it to an array  export roles\_array=($roles)  # Define minion numbers  export NUM\_NODES=${NUM\_NODES:-1}  # define the IP range used for service cluster IPs.  # according to rfc 1918 ref: https://tools.ietf.org/html/rfc1918 choose a private ip range here.  export SERVICE\_CLUSTER\_IP\_RANGE=${SERVICE\_CLUSTER\_IP\_RANGE:-192.168.3.0/24} # formerly PORTAL\_NET  # define the IP range used for flannel overlay network, should not conflict with above SERVICE\_CLUSTER\_IP\_RANGE  export CNI\_PLUGIN\_CONF CNI\_PLUGIN\_EXES CNI\_KUBELET\_TRIGGER  CNI\_PLUGIN\_CONF=${CNI\_PLUGIN\_CONF:-""}  CNI\_PLUGIN\_EXES=${CNI\_PLUGIN\_EXES:-""}  CNI\_KUBELET\_TRIGGER=${CNI\_KUBELET\_TRIGGER:-networking}  # Flannel networking is used if CNI networking is not. The following  # variable defines the CIDR block from which cluster addresses are  # drawn.  export FLANNEL\_NET=${FLANNEL\_NET:-10.1.0.0/16}  # Optionally add other contents to the Flannel configuration JSON  # object normally stored in etcd as /coreos.com/network/config. Use  # JSON syntax suitable for insertion into a JSON object constructor  # after other field name:value pairs. For example:  # FLANNEL\_OTHER\_NET\_CONFIG=', "SubnetMin": "172.16.10.0", "SubnetMax": "172.16.90.0"'  export FLANNEL\_OTHER\_NET\_CONFIG  FLANNEL\_OTHER\_NET\_CONFIG=''  # Admission Controllers to invoke prior to persisting objects in cluster  # If we included ResourceQuota, we should keep it at the end of the list to prevent incremeting quota usage prematurely.  export ADMISSION\_CONTROL=NamespaceLifecycle,LimitRanger,ServiceAccount,SecurityContextDeny,DefaultStorageClass,ResourceQuota  #export ADMISSION\_CONTROL=NamespaceLifecycle,LimitRanger,ServiceAccount,SecurityContextDeny,ResourceQuota  # Path to the config file or directory of files of kubelet  export KUBELET\_CONFIG=${KUBELET\_CONFIG:-""}  # A port range to reserve for services with NodePort visibility  SERVICE\_NODE\_PORT\_RANGE=${SERVICE\_NODE\_PORT\_RANGE:-"30000-32767"}  # Optional: Enable node logging.  ENABLE\_NODE\_LOGGING=true  LOGGING\_DESTINATION=${LOGGING\_DESTINATION:-elasticsearch}  # Optional: When set to true, Elasticsearch and Kibana will be setup as part of the cluster bring up.  ENABLE\_CLUSTER\_LOGGING=true  ELASTICSEARCH\_LOGGING\_REPLICAS=${ELASTICSEARCH\_LOGGING\_REPLICAS:-1}  # Optional: When set to true, heapster, Influxdb and Grafana will be setup as part of the cluster bring up.  ENABLE\_CLUSTER\_MONITORING="${KUBE\_ENABLE\_CLUSTER\_MONITORING:-true}"  # Extra options to set on the Docker command line. This is useful for setting  # --insecure-registry for local registries.  DOCKER\_OPTS=${DOCKER\_OPTS:-""}  # Extra options to set on the kube-proxy command line. This is useful  # for selecting the iptables proxy-mode, for example.  KUBE\_PROXY\_EXTRA\_OPTS=${KUBE\_PROXY\_EXTRA\_OPTS:-""}  # Optional: Install cluster DNS.  ENABLE\_CLUSTER\_DNS="${KUBE\_ENABLE\_CLUSTER\_DNS:-true}"  # DNS\_SERVER\_IP must be a IP in SERVICE\_CLUSTER\_IP\_RANGE  DNS\_SERVER\_IP=${DNS\_SERVER\_IP:-"192.168.3.10"}  #  DNS\_DOMAIN=${DNS\_DOMAIN:-"cluster.local"}  DNS\_REPLICAS=${DNS\_REPLICAS:-1}  # Optional: Install Kubernetes UI  ENABLE\_CLUSTER\_UI="${KUBE\_ENABLE\_CLUSTER\_UI:-true}"  # Optional: Enable setting flags for kube-apiserver to turn on behavior in active-dev  #RUNTIME\_CONFIG=""  # Optional: Add http or https proxy when download easy-rsa.  # Add environment variable separated with blank space like "http\_proxy=http://10.x.x.x:8080 https\_proxy=https://10.x.x.x:8443"  PROXY\_SETTING=${PROXY\_SETTING:-""}  # Optional: Allows kublet/kube-api to be run in privileged mode  ALLOW\_PRIVILEGED=${ALLOW\_PRIVILEGED:-"true"}  DEBUG=${DEBUG:-"false"}  # Add SSH\_OPTS: Add this to config ssh port  SSH\_OPTS="-oPort=22 -oStrictHostKeyChecking=no -oUserKnownHostsFile=/dev/null -oLogLevel=ERROR" |

#编辑admin 权限

vim util.sh

#只需要修改 admission-control=ServiceAccount

|  |
| --- |
| function create-kube-apiserver-opts() {  cat <<EOF > ~/kube/default/kube-apiserver  KUBE\_APISERVER\_OPTS="\  --insecure-bind-address=0.0.0.0\  --insecure-port=8080\  --etcd-servers=http://127.0.0.1:4001\  --logtostderr=true\  --service-cluster-ip-range=${1}\  --service-node-port-range=${3}\  --advertise-address=${4}\  --allow-privileged=${5}\  --admission-control=ServiceAccount\  --client-ca-file=/srv/kubernetes/ca.crt\  --tls-cert-file=/srv/kubernetes/server.cert\  --tls-private-key-file=/srv/kubernetes/server.key"  EOF  } |

### **3.2 启动**

cd kubernetes/cluster

vim /etc/profile

#add line

export KUBERNETES\_PROVIDER=ubuntu

#执行k8s-cluster 启动文件

./kube-up.sh

＃查看日志-检测apiserver是否启动(一般只需看这个日志就好了，其它的日志都在此同目录下)

tail -f -n 1000 /var/log/upstart/kube-apiserver.log

#停止集群

./kube-down.sh

### **3.3 配置docker-registry config secret(此步骤我已整合到3.4一并执行)**

### (由于项目内使用镜像为公司私有阿里云镜像，需要授权)

1. ＃创建阿里云授权
2. ＃default
   * 1. kubectl create secret docker-registry aliregistrykey --docker-server=https://hub.docker.vpclub.cn --docker-username=chen.wei --docker-password=vpclub.prod [--docker-email=chen.wei@vpclub.cn](mailto:--docker-email%3Dchen.wei@vpclub.cn)
     2. #kube
     3. kubectl create secret docker-registry aliregistrykey --docker-server=https://hub.docker.vpclub.cn --docker-username=chen.wei --docker-password=vpclub.prod --docker-email=chen.wei@vpclub.cn --namespace=kube-system

### **3.4 部署dns，dashboard**

* + 1. cd kubernetes/cluster/ubuntu
    2. **./deployAddons.sh**
    3. #查看DNS and UI pods are running in the cluster

### kubectl get pods --namespace=kube-system

### **3.5 创建对外入口nginx-ingress-controller,并部署服务**

cd kubernetes/cluster/examples

**#创建tls secret 注意将create-tls.sh内的域名换成自己的**

|  |
| --- |
| #!/bin/bash  host=$1  if [ "$host" == "" ]; then  echo "please add host domain param"  exit  fi  secretname=$2  if [ "$secretname" == "" ]; then  echo "please add secretname param"  exit  fi  openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /tmp/tls.key -out /tmp/tls.crt -subj "/CN=$host"  crt=`base64 /tmp/tls.crt`  echo $crt > crt.txt  sed -i -- 's/ //g' crt.txt  crt=$(cat crt.txt)  key=`base64 /tmp/tls.key`  echo $key > key.txt  sed -i -- 's/ //g' key.txt  key=$(cat key.txt)  echo "  apiVersion: v1  kind: Secret  metadata:  name: $secretname  data:  tls.crt: $crt  tls.key: $key" > $secretname.yaml  #del useless txt  rm crt.txt key.txt  ####create secret into k8s with domain k8s.cmos.cn  kubectl create -f k8s-secret.yaml |

**＃创建nginx-ingress-controller(此步需要安装go,编译安装参考**<https://github.com/RobinsChens/kubernetes-ingress.git>)

也可直接使用demo中的nginx-ingress-rc.yaml（反正上面已经创建了aliregistrykey）

|  |
| --- |
| apiVersion: v1  kind: ReplicationController  metadata:  name: nginx-ingress-rc  labels:  app: nginx-ingress  spec:  replicas: 1  selector:  app: nginx-ingress  template:  metadata:  labels:  app: nginx-ingress  spec:  imagePullSecrets:  - name: aliregistrykey  containers:  - image: hub.docker.vpclub.cn/vpclub/nginx-ingress:0.2  imagePullPolicy: Always  name: nginx-ingress  ports:  - containerPort: 80  hostPort: 80  - containerPort: 443  hostPort: 443 |

#部署nginx-ingress-rc

kubectl create -f nginx-ingress-rc.yaml

**＃部署自己的服务，此处用我手里的项目做demo，目录结构如下**

cd **services/k8s-cluster/github/kubernetes/cluster/**examples

ls

|  |
| --- |
|  |

#创建Dockerfile(此处已经可以使用allmake 自动创建)

|  |
| --- |
| FROM hub.docker.vpclub.cn/vpclub/java:8  VOLUME /tmp  ADD app/vp-traffic-monetisation-provider-1.0.0.jar /app/vp-traffic-monetisation-provider-1.0.0.jar  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app/vp-traffic-monetisation-provider-1.0.0.jar"] |

＃然后build镜像，并上传至自己的私有镜像库，推荐有公网域名，自己搭建docker-registry

参考文档services/document下面的private-install-docker-registry

#创建服务provider的rc,svc（此处已经可以使用allmake 自动创建deployk8s.yaml来代替）

vim traffic-provider-rc.yaml

|  |
| --- |
| apiVersion: v1  kind: ReplicationController  metadata:  name: traffic-provider  spec:  replicas: 2  template:  metadata:  labels:  app: traffic  spec:  containers:  - name: vp-traffic-monetisation-provider  image: hub.docker.vpclub.cn/vpclub\_containers/vp-traffic-monetisation-provider:1.0.0  ports:  - containerPort: 10010  # - port: 10010  imagePullSecrets:  - name: aliregistrykey |

vim traffic-provider-svc.yaml

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: traffic-svc  labels:  app: traffic  spec:  ports:  - port: 80  targetPort: 10010  protocol: TCP  name: http  selector:  app: traffic |

#同理创建consumer的相关yaml

vim traffic-consumer-rc.yaml

|  |
| --- |
| apiVersion: v1  kind: ReplicationController  metadata:  name: traffic-consumer  spec:  replicas: 1  template:  metadata:  labels:  app: traffic-con  spec:  containers:  - name: vp-traffic-monetisation-consumer  image: hub.docker.vpclub.cn/vpclub\_containers/vp-traffic-monetisation-consumer:1.0.0  ports:  - containerPort: 10020  # - port: 10010  imagePullSecrets:  - name: aliregistrykey |

vim traffic-consumer-svc.yaml

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: trafficcon-svc  labels:  app: traffic-con  spec:  ports:  - port: 80  targetPort: 10020  protocol: TCP  name: http  selector:  app: traffic-con |

#创建自己的ingress后台consumer服务restapi路由规则(此为demo可以自己改)

vim k8s-ingress.yaml

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: k8s-ingress  spec:  tls:  - hosts:  - k8s.cmos.cn  secretName: k8s-secret  rules:  - host: k8s.cmos.cn  http:  paths:  - path: /traffic  backend:  serviceName: trafficcon-svc ##需要部署此name的service服务  servicePort: 80 |

#部署服务，并部署ingress

kubectl create -f vp-traffic-monetisation-provider/traffic-provider-rc.yaml

kubectl create -f vp-traffic-monetisation-provider/traffic-provider-svc.yaml

kubectl create -f vp-traffic-monetisation-consumer/traffic-consumer-rc.yaml

kubectl create -f vp-traffic-monetisation-consumer/traffic-consumer-svc.yaml

kubectl create -f k8s-ingress.yaml

#Find out the external IP address of the node where the controller is running:

#查询nginx-controller 部署的nodeip 地址，（内网域名测试需要配置host）

kubectl get pods -o wide

## **4. 安装EFK日志**

### **4.1 生成fluentd,elasticsearch,kibana镜像**

* + 1. **(参考部署文件目录k8s-cluster/github/kubernetes/cluster/addons/fluentd-elasticsearch)**

### **4.2 创建创建部署yaml并部署EFK**

* + 1. **(参考部署文件目录k8s-cluster/github/kubernetes/cluster/addons/efk)**

# 1.elasticsearch 在kubernetes中的账户

# service-account.yaml

|  |
| --- |
| apiVersion: v1  kind: ServiceAccount  metadata:  name: elasticsearch  namespace: kube-system |

# ２.elasticsearch-logging配置yaml

elasticsearch-logging.yaml

|  |
| --- |
| #  # Copyright (C) 2015 Red Hat, Inc.  #  # Licensed under the Apache License, Version 2.0 (the "License");  # you may not use this file except in compliance with the License.  # You may obtain a copy of the License at  #  # http://www.apache.org/licenses/LICENSE-2.0  #  # Unless required by applicable law or agreed to in writing, software  # distributed under the License is distributed on an "AS IS" BASIS,  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  # See the License for the specific language governing permissions and  # limitations under the License.  #  kind: "List"  apiVersion: "v1"  items:  - apiVersion: "v1"  kind: "Service"  metadata:  name: "elasticsearch"  namespace: "kube-system"  labels:  component: "elasticsearch"  provider: "fabric8"  spec:  ports:  - port: 9200  targetPort: 9200  selector:  component: "elasticsearch"  type: "client"  provider: "fabric8"  type: "LoadBalancer"  - apiVersion: "v1"  kind: "Service"  metadata:  name: "elasticsearch-masters"  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "master"  provider: "fabric8"  spec:  clusterIP: "None"  ports:  - port: 9300  targetPort: 9300  selector:  provider: "fabric8"  type: "master"  component: "elasticsearch"  - apiVersion: "v1"  kind: "ReplicationController"  metadata:  name: "elasticsearch-data"  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "data"  provider: "fabric8"  spec:  replicas: 1  selector:  component: "elasticsearch"  type: "data"  provider: "fabric8"  template:  metadata:  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "data"  provider: "fabric8"  spec:  serviceAccount: elasticsearch  serviceAccountName: elasticsearch  imagePullSecrets:  - name: aliregistrykey  containers:  - env:  - name: "SERVICE"  value: "elasticsearch-masters"  - name: "KUBERNETES\_NAMESPACE"  valueFrom:  fieldRef:  fieldPath: "metadata.namespace"  - name: "NODE\_DATA"  value: "true"  - name: "NODE\_MASTER"  value: "false"  - name: "TZ"  value: "Asia/Shanghai"  image: "hub.docker.vpclub.cn/vpclub\_containers/docker-elasticsearch-kubernetes:2.3.5"  name: "elasticsearch"  ports:  - containerPort: 9300  name: "transport"  volumeMounts:  - mountPath: "/usr/share/elasticsearch/data"  name: "elasticsearch-data"  readOnly: false  volumes:  - emptyDir:  medium: ""  name: "elasticsearch-data"  - apiVersion: "v1"  kind: "ReplicationController"  metadata:  name: "elasticsearch-master"  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "master"  provider: "fabric8"  spec:  replicas: 1  selector:  component: "elasticsearch"  type: "master"  provider: "fabric8"  template:  metadata:  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "master"  provider: "fabric8"  spec:  serviceAccount: elasticsearch  serviceAccountName: elasticsearch  imagePullSecrets:  - name: aliregistrykey  containers:  - env:  - name: "SERVICE"  value: "elasticsearch-masters"  - name: "KUBERNETES\_NAMESPACE"  valueFrom:  fieldRef:  fieldPath: "metadata.namespace"  - name: "NODE\_DATA"  value: "false"  - name: "NODE\_MASTER"  value: "true"  - name: "TZ"  value: "Asia/Shanghai"  image: "hub.docker.vpclub.cn/vpclub\_containers/docker-elasticsearch-kubernetes:2.3.5"  name: "elasticsearch"  ports:  - containerPort: 9300  name: "transport"  - apiVersion: "v1"  kind: "ReplicationController"  metadata:  name: "elasticsearch-client"  namespace: "kube-system"  labels:  component: "elasticsearch"  type: "master"  provider: "fabric8"  spec:  replicas: 1  selector:  component: "elasticsearch"  type: "client"  provider: "fabric8"  template:  metadata:  labels:  component: "elasticsearch"  type: "client"  provider: "fabric8"  spec:  serviceAccount: elasticsearch  serviceAccountName: elasticsearch  imagePullSecrets:  - name: aliregistrykey  containers:  - env:  - name: "SERVICE"  value: "elasticsearch-masters"  - name: "KUBERNETES\_NAMESPACE"  valueFrom:  fieldRef:  fieldPath: "metadata.namespace"  - name: "NODE\_DATA"  value: "false"  - name: "NODE\_MASTER"  value: "false"  - name: "TZ"  value: "Asia/Shanghai"  image: "hub.docker.vpclub.cn/vpclub\_containers/docker-elasticsearch-kubernetes:2.3.5"  name: "elasticsearch"  ports:  - containerPort: 9200  name: "http"  - containerPort: 9300  name: "transport" |

# 3. fluentd的配置yaml

fluentd-es-logging.yaml

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: fluentd-logging  namespace: kube-system  labels:  k8s-app: fluentd-logging  spec:  imagePullSecrets:  - name: aliregistrykey  dnsPolicy: "ClusterFirst"  containers:  - name: fluentd-logging  image: hub.docker.vpclub.cn/vpclub\_containers/fluentd-elasticsearch:1.21  resources:  limits:  memory: 200Mi  requests:  cpu: 100m  memory: 200Mi  env:  - name: "ELASTICSEARCH\_URL"  value: "${ELASTICSEARCH\_HOST}"  - name: TZ  value: Asia/Shanghai  volumeMounts:  - name: varlog  mountPath: /var/log  - name: varlibdockercontainers  mountPath: /var/lib/docker/containers  readOnly: true  terminationGracePeriodSeconds: 30  volumes:  - name: varlog  hostPath:  path: /var/log  - name: varlibdockercontainers  hostPath:  path: /var/lib/docker/containers |

# 4.　kibana的配置yaml

kibana-controller.yaml

|  |
| --- |
| apiVersion: v1  kind: ReplicationController  metadata:  name: kibana-logging-v1  # namespace: kube-system  labels:  k8s-app: kibana-logging  version: v1  kubernetes.io/cluster-service: "true"  spec:  replicas: 1  selector:  k8s-app: kibana-logging  version: v1  template:  metadata:  labels:  k8s-app: kibana-logging  version: v1  kubernetes.io/cluster-service: "true"  annotations:  scheduler.alpha.kubernetes.io/critical-pod: ''  scheduler.alpha.kubernetes.io/tolerations: '[{"key":"CriticalAddonsOnly", "operator":"Exists"}]'  spec:  imagePullSecrets:  - name: aliregistrykey  containers:  - name: kibana-logging  image: hub.docker.vpclub.cn/vpclub\_containers/kibana:1.5-1  resources:  # keep request = limit to keep this container in guaranteed class  limits:  cpu: 100m  requests:  cpu: 100m  env:  - name: "ELASTICSEARCH\_URL"  value: "http://${ELASTICSEARCH\_HOST}:9200"  - name: TZ  value: Asia/Shanghai  ports:  - containerPort: 5601  name: ui  protocol: TCP |

#配置kibana的service是为了单独配置内网域名的log查询入口,配合nginx-ingress

kibana-service.yaml

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: kibana-logging  # namespace: kube-system  labels:  k8s-app: kibana-logging  kubernetes.io/cluster-service: "true"  kubernetes.io/name: "Kibana"  spec:  ports:  - port: 80  targetPort: 5601  # - port: 5601  protocol: TCP  # targetPort: ui  selector:  k8s-app: kibana-logging |

# 5.　创建部署脚本(add daemonset 方式的fluentd-logging )

若想使用daemonset去掉#use daemonset下面所有行的注释即可

create-efk.sh

|  |
| --- |
| #!/bin/bash  kubectl create -f service-account.yaml  kubectl create -f kibana-service.yaml  kubectl create -f elasticsearch-logging.yaml  ELASTICSEARCH\_HOST=$(kubectl --namespace=kube-system get svc elasticsearch -o template --template={{.spec.clusterIP}})  echo "host is $ELASTICSEARCH\_HOST"  cp fluentd-es-logging.yaml fluentd-es-logging-actual.yaml  cp kibana-controller.yaml kibana-controller-actual.yaml  cp fluentd-daemonset.yaml fluentd-daemonset-actual.yaml  #replace  sed -i -- "s/\${ELASTICSEARCH\_HOST}/${ELASTICSEARCH\_HOST}/g" fluentd-es-logging-actual.yaml  sed -i -- "s/\${ELASTICSEARCH\_HOST}/${ELASTICSEARCH\_HOST}/g" kibana-controller-actual.yaml  kubectl create -f fluentd-es-logging-actual.yaml  kubectl create -f kibana-controller-actual.yaml  rm fluentd-es-logging-actual.yaml kibana-controller-actual.yaml  #use daemonset  #sed -i -- "s/\${ELASTICSEARCH\_HOST}/${ELASTICSEARCH\_HOST}/g" fluentd-daemonset-actual.yaml  #kubectl create -f fluentd-daemonset-actual.yaml  #rm fluentd-daemonset-actual.yaml |

# 6.　配置log.k8s.cmos.cn（域名自定义的访问需要配置hosts） 的ingress,并定位 nginx-ingress的pod的nodeip（可以通过命令方式获得所有pod详细信息就可以看到）

6.1执行上面创建好的create-tls.sh 参数为log域名

./create-tls.sh log.k8s.cmos.cn k8s-log

6.2创建create-ingress.sh ,为kibana的service做nginx代理

vim create-ingress.sh

|  |
| --- |
| #!/bin/bash  host=$1  if [ "$host" == "" ]; then  echo "please add first param host domain "  exit  fi  secretname=$2  if [ "$secretname" == "" ]; then  echo "please add second param secretname"  exit  fi  appsvc=$3  if [ "$secretname" == "" ]; then  echo "please add third param appsvc"  exit  fi  echo "  apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: $secretname-ingress  spec:  tls:  - hosts:  - $host  secretName: $secretname  rules:  - host: $host  http:  paths:  - path: /  backend:  serviceName: $appsvc  servicePort: 80" > $secretname-ingress.yaml  #deploy log ingress  kubectl create -f $secretname-ingress.yaml |

＃授执行权限

sudo chmod 755 create-ingress.sh

#执行第一,二个参数与 create-tls.sh相同,第三个参数为需要代理的app svc，此处为kibana-logging

./create-ingress.sh log.k8s.cmos.cn k8s-log kibana-logging

6.3 查询nginx-ingress 的pod 的nodeip

kubectl get pods -o wide

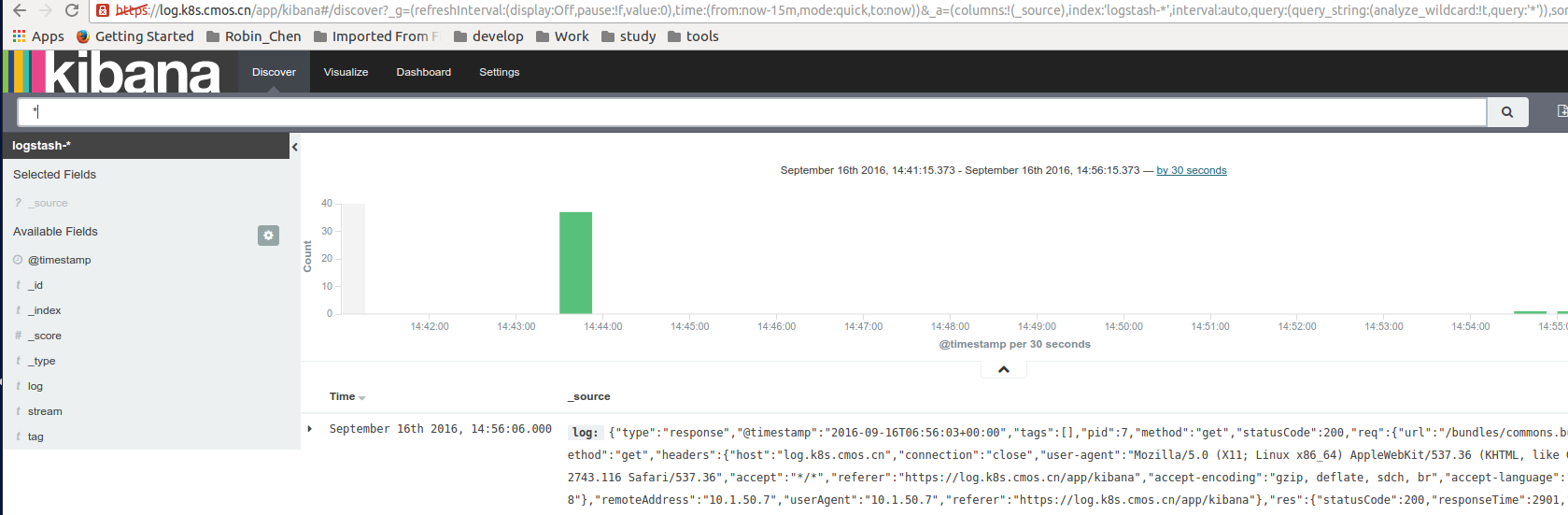


看到nginx-ingress-rc开头的pod ，对应NODE 的值，即为自定义域名对应的宿主机.

此刻配置/etc/hosts　文件,加上如下配置

172.16.2.233 log.k8s.cmos.cn

然后访问https://log.k8s.cmos.cn即可



## **5. 利用allmake批量工作**

### **5.1 批量生成Dockerfile,deployk8s.yaml，以及Makefile**

### **首先在master主机安装allmake**

### **参考（**[**https://github.com/allmake/allmake.git**](https://github.com/allmake/allmake.git)**）**

### **demo参考目录services/k8s-cluster/github/kubernetes/cluster/examples**

# １．编辑部署目录（如果已经按照运维文档建立过就无需此步）

1. mkdir -p deployment/backservices/
2. cd deployment/backservices/
3. ①创建（更新）部署目录 执行 deploy.sh脚本(vim deploy.sh 并授执行权限sudo chmod 755 deploy.sh)　(６个参数)

|  |
| --- |
| #!/bin/bash  inet=$1  if [ "$inet" == "" ]; then  inet=eth0  fi  profile=$2  if [ "$profile" == "" ]; then  profile=test  fi  tmp=$3  if [ "$tmp" == "" ]; then  tmp=java  fi  jmx=$4  if [ "$jmx" == "" ]; then  jmx=128m  fi  ver=$5  if [ "$ver" == "" ]; then  ver=1.0.0  fi  logip=$6  if [ "$logip" == "" ]; then  logip=172.16.45.3:5000  fi  function addNewApp()  {  allmake add app=$1 ver=$ver profile=$profile ports="$2" template=$tmp inet=$inet jmx=$4 logIpAddress=$logip  }  mkdir -p 01-provider 02-consumer  pushd 01-provider  addNewApp vp-traffic-monetisation-provider "10010 23010 33010" "" 128m  popd  pushd 02-consumer  addNewApp vp-traffic-monetisation-consumer "10020 23020 33020" "" 128m  popd |

1. #执行脚本创建服务器目录（自动创建Mockerfile,deployk8s.yaml,Makefile)
   * 1. ./deploy.sh eth0 dev dockerfile

#此时创建k8s所需镜像,依次执行各个app下面的Makefile即sudo make 即可

* + 1. 注意这里的Makefile都是按照本公司的docker registry配置
    2. sudo make all

#这个批量执行我还没写。。。

allmake cmd="sudo make"

allmake cmd="kubectl create -f deployk8s.yaml"

#发布到k8s－各个项目deployk8s.yaml路径下执行

kubectl create -f deployk8s.yaml

#配置consumer的svc到nginx-ingress 此处与上面做法一致

如下

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: k8s-ingress  spec:  tls:  - hosts:  - k8s.cmos.cn  secretName: k8s-secret  rules:  - host: k8s.cmos.cn  http:  paths:  - path: /traffic  backend:  serviceName: trafficcon-svc ##需要部署此name的service服务  servicePort: 80 |

＃不同的consumer的svc自己添加即可，目前还没有全部自动化

## **6. 添加工作节点work node**

### **6.1 拷贝原先work节点中/root/kube 文件夹至需新增的工作节点中/root/kube**

* + 1. **kube目录可以参考（k8s-cluster/github/kubernetes/kube）**
    2. scp -r work1:/root/kube work2:/root
    3. ssh work2
    4. cd kube

# **1.编辑添加work节点脚本**

* + 1. vim kube-work.sh

|  |
| --- |
| #!/bin/bash  KUBE\_ROOT=$(dirname "${BASH\_SOURCE}")/  DEFAULT\_KUBECONFIG="${HOME}/.kube/config"  source ~/kube/util.sh  export MASTER\_IP=$1  if [ "$MASTER\_IP" == "" ]; then  echo "please add MASTER\_IP first param"  exit  fi  if [ "$host" == "" ]; then  host=$( ifconfig eth0 | grep "inet addr" | cut -d ':' -f 2 | cut -d ' ' -f 1 )  echo "current node is $host"  fi  source ~/kube/config-default.sh  if [ -z "$CNI\_PLUGIN\_CONF" ] || [ -z "$CNI\_PLUGIN\_EXES" ]; then  # Prep for Flannel use: copy the flannel binaries and scripts, set reconf flag  NEED\_RECONFIG\_DOCKER=true  CNI\_PLUGIN\_CONF=''  else  # Prep for CNI use: copy the CNI config and binaries, adjust upstart config, set reconf flag  NEED\_RECONFIG\_DOCKER=false  fi  function create-kubelet-opts() {  if [ -n "$7" ] ; then  cni\_opts=" --network-plugin=cni --network-plugin-dir=/etc/cni/net.d"  else  cni\_opts=""  fi  cat <<EOF > ~/kube/default/kubelet  KUBELET\_OPTS="\  --hostname-override=${1} \  --api-servers=http://${2}:8080 \  --logtostderr=true \  --cluster-dns=${3} \  --pod-infra-container-image=docker.io/kubernetes/pause:latest \  --cluster-domain=${4} \  --config=${5} \  --allow-privileged=${6}  $cni\_opts"  EOF  }  # Create ~/kube/default/kube-proxy with proper contents.  # $1: The hostname or IP address by which the node is identified.  # $2: The one hostname or IP address at which the API server is reached (insecurely).  function create-kube-proxy-opts() {  cat <<EOF > ~/kube/default/kube-proxy  KUBE\_PROXY\_OPTS="\  --hostname-override=${1} \  --master=http://${2}:8080 \  --logtostderr=true \  ${3}"  EOF  }  # Create ~/kube/default/flanneld with proper contents.  # $1: The one hostname or IP address at which the etcd leader listens.  # $2: The IP address or network interface for the local Flannel daemon to use  function create-flanneld-opts() {  cat <<EOF > ~/kube/default/flanneld  FLANNEL\_OPTS="--etcd-endpoints=http://${1}:4001 \  --ip-masq \  --iface=${2}"  EOF  }  function proxy-starts() {  cp ~/kube/default/\* /etc/default/  cp ~/kube/init\_conf/\* /etc/init/  cp ~/kube/init\_scripts/\* /etc/init.d/  mkdir -p /opt/bin/  cp ~/kube/minion/\* /opt/bin  #restart Docker#config docker config  echo 'NEED\_RECONFIG\_DOCKER is' ${NEED\_RECONFIG\_DOCKER}  if ${NEED\_RECONFIG\_DOCKER}; then  echo ${DOCKER\_OPTS} is \"${DOCKER\_OPTS}\"  KUBE\_CONFIG\_FILE=\"${KUBE\_CONFIG\_FILE}\"  DOCKER\_OPTS=\"${DOCKER\_OPTS}\"  ~/kube/reconfDocker.sh i;  fi  service flanneld start  service kubelet start  service kube-proxy start  }  function create-proxy-config() {  #statements  echo "ALLOW\_PRIVILEGED is ${ALLOW\_PRIVILEGED} "  create-kubelet-opts \  ${host} \  ${MASTER\_IP} \  ${DNS\_SERVER\_IP} \  ${DNS\_DOMAIN} \  '${KUBELET\_CONFIG}' \  ${ALLOW\_PRIVILEGED} \  ${CNI\_PLUGIN\_CONF}  create-kube-proxy-opts \  ${host} \  ${MASTER\_IP} \  ${KUBE\_PROXY\_EXTRA\_OPTS}  create-flanneld-opts ${MASTER\_IP} ${host}  }  #first config proxy  create-proxy-config  # start proxy kubelet  proxy-starts |

### 

# **2授权并执行-参数为master-ip**

* + 1. chmod 755 kube-work.sh
    2. ./kube-work.sh 172.16.2.234