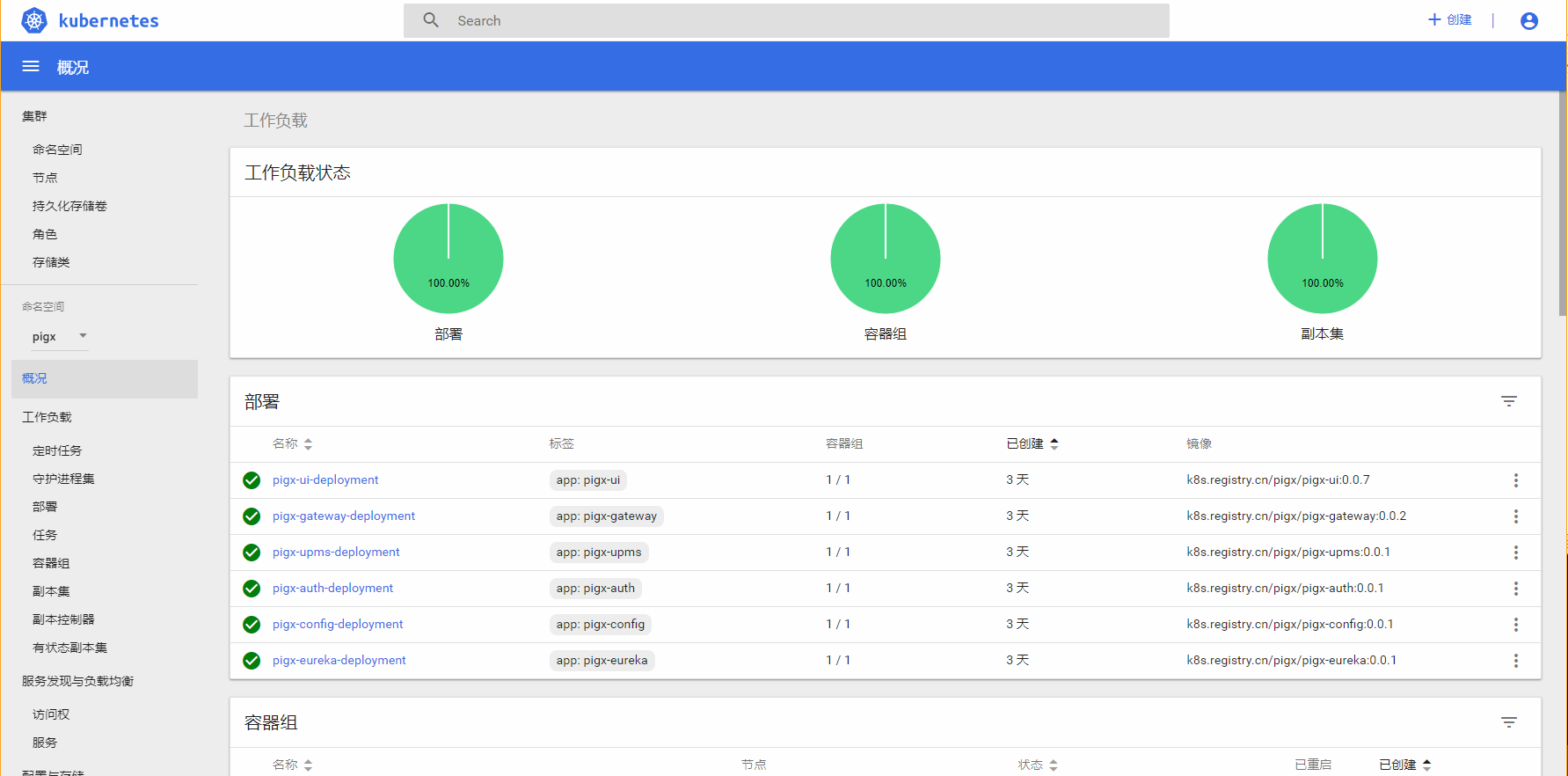
[**#**](https://pig4cloud.com/#%E5%86%99%E5%9C%A8%E5%89%8D%E9%9D%A2) **写在前面**

首先，要感谢@冷冷写了Pigx这个框架，让我学了N多的知识，在此膜拜!

本来是打算录视频的，但是由于最近工作太忙了，老是加班所以视频就不录，我把部署的相关内容写成文字分享一下自己的部署过程。本教程不面向小白，建议大家还是要看看K8S、Docker相关的内容。如果大家有什么疑问可以在pig·vip群里@githink.cn,我看到后会替你解答疑问（会的我就解答，我不会的就...）。



[**#**](https://pig4cloud.com/#%E5%87%86%E5%A4%87%E5%B7%A5%E4%BD%9C) **准备工作**

1. 搭建私服镜像仓库
2. 搭建K8S 1.13.1高可用集群
3. 准备镜像，也就是Pigx各个模块的镜像
4. 部署各个镜像

由于我自己已有外网环境的服务器，所以这里不会讲集群内如何部署MySQL、Redis、Rabbitmq等。有一个问题我也没有解决，就是验证码的问题。

[**#**](https://pig4cloud.com/#%E7%8E%AF%E5%A2%83%E6%A6%82%E8%A7%88) **环境概览**

注意：虚拟机需要设置成固定IP！！！（不会的自行百度）。自行安装Docker 18.06.1-ce。

| **IP** | **Hostname** | **CPU** | **Memory** | **disk** | **职责** |
| --- | --- | --- | --- | --- | --- |
| 192.168.1.10 | k8s.registry.cn | 1 | 1G | 50G | Docker私库 |
| 192.168.1.11 | k8s-master1 api.k8s.cn | 4 | 4G | 20G | 集群Master |
| 192.168.1.12 | k8s-master2 | 4 | 4G | 20G | 集群Master |
| 192.168.1.13 | k8s-master3 | 4 | 4G | 20G | 集群Master |
| 192.168.1.14 | k8s-slave1 | 4 | 4G | 20G | 集群node |
| 192.168.1.15 | k8s-slave2 | 4 | 4G | 20G | 集群node |

docker：18.06.1-cecentos：7.5kubernetes：1.13.1

[**#**](https://pig4cloud.com/#%E5%89%8D%E6%9C%9F%E5%87%86%E5%A4%87) **前期准备**

1.关闭所有虚拟机的防火墙（可自行配置防火墙规则）

systemctl stop firewalld && systemctl disable firewalld

2.免密登录

# 在k8s-master1机器上生成秘钥

ssh-keygen

# 将生成的秘钥分发到其他虚拟机上

ssh-copy-id root@k8s-master1

ssh-copy-id root@k8s-master2

ssh-copy-id root@k8s-master3

ssh-copy-id root@k8s-slave1

ssh-copy-id root@k8s-slave2

3.配置所有虚拟机的hosts

cat >>/etc/hosts<<EOF

192.168.1.11 k8s-master1 api.k8s.cn

192.168.1.12 k8s-master2

192.168.1.13 k8s-master3

192.168.1.14 k8s-slave1

192.168.1.15 k8s-slave2

192.168.1.10 k8s.registry.cn

EOF

4.所有虚拟机通用配置

# 新建 iptable 配置修改文件

cat <<EOF > net.iptables.k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

#关闭 swap 分区

sudo swapoff -a

#防止开机自动挂载 swap 分区，注释掉配置

sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab

#关闭 SELinux

sudo setenforce 0

#防止开机启动开启，修改 SELINUX 配置

sudo sed -i s'/SELINUX=enforcing/SELINUX=disabled'/g /etc/selinux/config

配置 iptables

sudo mv net.iptables.k8s.conf /etc/sysctl.d/ && sudo sysctl --system

#安装 wget

sudo yum install -y wget

[**#**](https://pig4cloud.com/#%E6%90%AD%E5%BB%BA%E7%A7%81%E6%9C%8D%E9%95%9C%E5%83%8F%E4%BB%93%E5%BA%93) **搭建私服镜像仓库**

[**#**](https://pig4cloud.com/#1.-%E5%AE%89%E8%A3%85docker-compose) **1. 安装Docker-compose**

# 1.安装Docker-compose

sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

#赋予Docker-compose执行权限

sudo chmod +x /usr/local/bin/docker-compose

#验证Docker-compose

docker-compose --version

[**#**](https://pig4cloud.com/#2.-%E5%AE%89%E8%A3%85harbor) **2. 安装Harbor**

# 1. 下载安装文件（可以在指定目录下载）

wget https://storage.googleapis.com/harbor-releases/harbor-online-installer-v1.5.2.tgz

# 2. 解压下载的文件

tar xvf harbor-online-installer-v1.5.2.tgz

[**#**](https://pig4cloud.com/#3.-%E9%85%8D%E7%BD%AEharbor) **3. 配置Harbor**

1. 修改Harbor的配置文件

cd harbor

vim harbor.cfg

内容如下：

# hostname设置访问地址，可以使用ip、域名，不可以设置为127.0.0.1或localhost

hostname = hub.k8s.com

# 访问协议，默认是http，也可以设置https，如果设置https，则nginx ssl需要设置on

ui\_url\_protocol = http

# mysql数据库root用户默认密码root123，实际使用时修改下

db\_password = root@1234

max\_job\_workers = 3

customize\_crt = on

ssl\_cert = /data/cert/server.crt

ssl\_cert\_key = /data/cert/server.key

secretkey\_path = /data

admiral\_url = NA

# 邮件设置，发送重置密码邮件时使用

email\_identity =

email\_server = smtp.mydomain.com

email\_server\_port = 25

email\_username = sample\_admin@mydomain.com

email\_password = abc

email\_from = admin <sample\_admin@mydomain.com>

email\_ssl = false

# 启动Harbor后，管理员UI登录的密码，默认是Harbor12345

harbor\_admin\_password = root@1234

# 认证方式，这里支持多种认证方式，如LADP、本次存储、数据库认证。默认是db\_auth，mysql数据库认证

auth\_mode = db\_auth

# LDAP认证时配置项

#ldap\_url = ldaps://ldap.mydomain.com

#ldap\_searchdn = uid=searchuser,ou=people,dc=mydomain,dc=com

#ldap\_search\_pwd = password

#ldap\_basedn = ou=people,dc=mydomain,dc=com

#ldap\_filter = (objectClass=person)

#ldap\_uid = uid

#ldap\_scope = 3

#ldap\_timeout = 5

# 是否开启自注册

self\_registration = on

# Token有效时间，默认30分钟

token\_expiration = 30

# 用户创建项目权限控制，默认是everyone（所有人），也可以设置为adminonly（只能管理员）

project\_creation\_restriction = everyone

verify\_remote\_cert = on

[**#**](https://pig4cloud.com/#4.%E5%90%AF%E5%8A%A8harbor) **4.启动Harbor**

# 1.在当前安装目录下

./install.sh

[**#**](https://pig4cloud.com/#5.%E8%AE%BF%E9%97%AEharbor%E4%BB%A5%E5%8F%8A%E4%B8%80%E4%BA%9B%E5%B8%B8%E7%94%A8%E6%93%8D%E4%BD%9C) **5.访问Harbor以及一些常用操作**

配置宿主机对Docker私有仓库镜像IP的解析，编辑 C:\WINDOWS\System32\drivers\etc\hosts 文件

添加如下内容： 192.168.1.10 k8s.registry.cn

这样我们在宿主机直接访问hub.k8s.com就可以看到Docker私库的可视化界面。

**默认用户就是admin，密码就是我们在配置文件里配置的root@1234**

[**#**](https://pig4cloud.com/#6.-%E4%BF%AE%E6%94%B9%E6%89%80%E6%9C%89%E8%99%9A%E6%8B%9F%E6%9C%BA%E7%9A%84docker%E7%9A%84daemon.json%E6%96%87%E4%BB%B6) **6. 修改所有虚拟机的Docker的daemon.json文件**

vim /etc/docker/daemon.json

# 添加如下内容：

"insecure-registries":["k8s.registry.cn"]

# 重启服务

systemctl restart docker

注意：**不要忘记添加内容之前加个逗号**

[**#**](https://pig4cloud.com/#7.-%E6%AF%8F%E5%8F%B0%E8%99%9A%E6%8B%9F%E6%9C%BA%E7%9A%84-docker-%E7%99%BB%E5%BD%95%E5%88%B0-harbor) **7. 每台虚拟机的 Docker 登录到 Harbor**

docker login k8s.registry.cn

#输入用户名密码即可（admin/root@1234）

[**#**](https://pig4cloud.com/#%E6%90%AD%E5%BB%BA%E9%9B%86%E7%BE%A4) **搭建集群**

[**#**](https://pig4cloud.com/#1.%E5%AE%89%E8%A3%85%E7%9B%B8%E5%85%B3%E8%BD%AF%E4%BB%B6) **1.安装相关软件**

# 1. 配置阿里云源

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=0

EOF

# 2. 开始安装

yum -y install kubelet-1.13.1 kubeadm-1.13.1 kubectl-1.13.1

# 3. 启动kubeadm服务

systemctl enable kubelet && systemctl start kubelet

[**#**](https://pig4cloud.com/#2.%E5%88%9D%E5%A7%8B%E5%8C%96-kubeadm-%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6) **2.初始化 kubeadm 配置文件**

1. 生成配置文件并分发到其他master节点

## 创建脚本：init.kubeadm.config.sh

#!/bin/sh

vhost=(k8s-master1 k8s-master2 k8s-master3)

vhostIP=(192.168.1.11 192.168.1.12 192.168.1.13)

domain=api.k8s.cn

## etcd 初始化 m01 m02 m03 集群配置

etcdInitCluster=(

k8s-master1=https://192.168.1.11:2380

k8s-master1=https://192.168.1.11:2380,k8s-master2=https://192.168.1.12:2380

k8s-master1=https://192.168.1.11:2380,k8s-master2=https://192.168.1.12:2380,k8s-master3=https://192.168.1.13:2380

)

## etcd 初始化时，m01 m02 m03 分别的初始化集群状态

initClusterStatus=(

new

existing

existing

)

## 2.遍历 master 主机名和对应 IP

## 生成对应的 kubeadmn 配置文件

for i in `seq 0 $((${#vhost[\*]}-1))`

do

h=${vhost[${i}]}

ip=${vhostIP[${i}]}

echo "--> $h - $ip"

## 生成 kubeadm 配置模板

cat <<EOF > kubeadm-config.$h.yaml

apiVersion: kubeadm.k8s.io/v1beta1

kind: InitConfiguration

localAPIEndpoint:

advertiseAddress: $ip

bindPort: 6443

---

apiVersion: kubeadm.k8s.io/v1beta1

kind: ClusterConfiguration

kubernetesVersion: v1.13.1

# 指定阿里云镜像仓库

imageRepository: registry.aliyuncs.com/google\_containers

# apiServerCertSANs 填所有的 masterip、lbip、其它可能需要通过它访问 apiserver 的地址、域名或主机名等，

# 如阿里fip，证书中会允许这些ip

# 这里填一个自定义的域名

apiServer:

certSANs:

- "$domain"

controlPlaneEndpoint: "$domain:6443"

## Etcd 配置

etcd:

local:

extraArgs:

listen-client-urls: "https://127.0.0.1:2379,https://$ip:2379"

advertise-client-urls: "https://$ip:2379"

listen-peer-urls: "https://$ip:2380"

initial-advertise-peer-urls: "https://$ip:2380"

initial-cluster: "${etcdInitCluster[${i}]}"

initial-cluster-state: ${initClusterStatus[${i}]}

serverCertSANs:

- $h

- $ip

peerCertSANs:

- $h

- $ip

networking:

podSubnet: "10.244.0.0/16"

EOF

echo "kubeadm-config.$h.yaml created ... ok"

## 3. 分发到其他 master 机器

scp kubeadm-config.$h.yaml root@$h:~

echo "scp kubeadm-config.$h.yaml ... ok"

done

2.创建获取镜像脚本并分享到其他master节点

## 创建脚本：kubeadm.images.sh

#!/bin/sh

vhost="k8s-master1 k8s-master2 k8s-master3"

for h in $vhost;do

echo "Pull image for $h -- begings"

sudo kubeadm config images pull --config kubeadm-config.$h.yaml

done

3.执行脚本

chmod +x kubeadm.images.sh

./kubeadm.images.sh

执行后可到其他master节点查看是否获取成功。

[**#**](https://pig4cloud.com/#%E5%AE%89%E8%A3%85k8s-master1) **安装k8s-master1**

1.初始化k8s-master1

# 安装

sudo kubeadm init --config kubeadm-config.m01.yaml

#重置

kubeadm reset --force

#命令行配置

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

# 验证

kubectl cluster-info

将安装输出的后的内容保存下来，后面安装slave节点时会用到。如： kubeadm join api.k8s.cn:6443.......这一串内容。

2.安装flannel网络插件

创建安装文件

vim k8s-flannel.yaml

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: flannel

rules:

- apiGroups:

- ""

resources:

- pods

verbs:

- get

- apiGroups:

- ""

resources:

- nodes

verbs:

- list

- watch

- apiGroups:

- ""

resources:

- nodes/status

verbs:

- patch

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: flannel

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: flannel

subjects:

- kind: ServiceAccount

name: flannel

namespace: kube-system

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: flannel

namespace: kube-system

---

kind: ConfigMap

apiVersion: v1

metadata:

name: kube-flannel-cfg

namespace: kube-system

labels:

tier: node

app: flannel

data:

cni-conf.json: |

{

"name": "cbr0",

"type": "flannel",

"delegate": {

"isDefaultGateway": true

}

}

net-conf.json: |

{

"Network": "10.244.0.0/16",

"Backend": {

"Type": "vxlan"

}

}

---

apiVersion: extensions/v1beta1

kind: DaemonSet

metadata:

name: kube-flannel-ds

namespace: kube-system

labels:

tier: node

app: flannel

spec:

template:

metadata:

labels:

tier: node

app: flannel

spec:

hostNetwork: true

nodeSelector:

beta.kubernetes.io/arch: amd64

tolerations:

- key: node-role.kubernetes.io/master

operator: Exists

effect: NoSchedule

serviceAccountName: flannel

initContainers:

- name: install-cni

image: quay.io/coreos/flannel:v0.9.1-amd64

command:

- cp

args:

- -f

- /etc/kube-flannel/cni-conf.json

- /etc/cni/net.d/10-flannel.conf

volumeMounts:

- name: cni

mountPath: /etc/cni/net.d

- name: flannel-cfg

mountPath: /etc/kube-flannel/

containers:

- name: kube-flannel

image: quay.io/coreos/flannel:v0.9.1-amd64

command: [ "/opt/bin/flanneld", "--ip-masq", "--kube-subnet-mgr" ]

securityContext:

privileged: true

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

volumeMounts:

- name: run

mountPath: /run

- name: flannel-cfg

mountPath: /etc/kube-flannel/

volumes:

- name: run

hostPath:

path: /run

- name: cni

hostPath:

path: /etc/cni/net.d

- name: flannel-cfg

configMap:

name: kube-flannel-cfg

#1.准备镜像

sudo docker pull jmgao1983/flannel:v0.10.0-amd64

sudo docker tag jmgao1983/flannel:v0.10.0-amd64 quay.io/coreos/flannel:v0.10.0-amd64

#2.kubelet apply 进行安装 flannel

kubectl apply -f ./kube-flannel.yml

[**#**](https://pig4cloud.com/#4.%E5%AE%89%E8%A3%85%E5%85%B6%E4%BB%96master%E8%8A%82%E7%82%B9) **4.安装其他Master节点**

1.将Ca证书分发到其他master节点

## 创建脚本：sync.master.ca.sh

#!/bin/sh

vhost="k8s-master2 k8s-master3"

usr=root

who=`whoami`

if [[ "$who" != "$usr" ]];then

echo "请使用 root 用户执行或者 sudo ./sync.master.ca.sh"

exit 1

fi

echo $who

# 需要从 m01 拷贝的 ca 文件

caFiles=(

/etc/kubernetes/pki/ca.crt

/etc/kubernetes/pki/ca.key

/etc/kubernetes/pki/sa.key

/etc/kubernetes/pki/sa.pub

/etc/kubernetes/pki/front-proxy-ca.crt

/etc/kubernetes/pki/front-proxy-ca.key

/etc/kubernetes/pki/etcd/ca.crt

/etc/kubernetes/pki/etcd/ca.key

/etc/kubernetes/admin.conf

)

pkiDir=/etc/kubernetes/pki/etcd

for h in $vhost

do

ssh ${usr}@$h "mkdir -p $pkiDir"

echo "Dirs for ca scp created, start to scp..."

# scp 文件到目标机

for f in ${caFiles[@]}

do

echo "scp $f ${usr}@$h:$f"

scp $f ${usr}@$h:$f

done

echo "Ca files transfered for $h ... ok"

done

执行脚本

chmod +x sync.master.ca.sh

sudo ./sync.master.ca.sh

2.安装k8s-master2

sudo kubeadm init phase certs all --config kubeadm-config.k8s-master2.yaml

sudo kubeadm init phase etcd local --config kubeadm-config.k8s-master2.yaml

sudo kubeadm init phase kubeconfig kubelet --config kubeadm-config.k8s-master2.yaml

sudo kubeadm init phase kubelet-start --config kubeadm-config.k8s-master2.yaml

将etcd加入集群

kubectl exec -n kube-system etcd-k8s-master1 -- etcdctl --ca-file /etc/kubernetes/pki/etcd/ca.crt --cert-file /etc/kubernetes/pki/etcd/peer.crt --key-file /etc/kubernetes/pki/etcd/peer.key --endpoints=https://192.168.1.11:2379 member add k8s-master2 https://192.168.1.12:2380

启动 kube-apiserver、kube-controller-manager、kube-scheduler

kubeadm init phase kubeconfig all --config kubeadm-config.k8s-master2.yaml

kubeadm init phase control-plane all --config kubeadm-config.k8s-master2.yaml

将k8s-master2加入Master节点

kubeadm init phase mark-control-plane --config kubeadm-config.k8s-master2.yaml

1. 安装k8s-master3

# 1. 配置证书、初始化 kubelet 配置和启动 kubelet

sudo kubeadm init phase certs all --config kubeadm-config.k8s-master3.yaml

sudo kubeadm init phase etcd local --config kubeadm-config.k8s-master3.yaml

sudo kubeadm init phase kubeconfig kubelet --config kubeadm-config.k8s-master3.yaml

sudo kubeadm init phase kubelet-start --config kubeadm-config.k8s-master3.yaml

# 2. 将 etcd 加入集群

kubectl exec -n kube-system etcd-k8s-master1 -- etcdctl --ca-file /etc/kubernetes/pki/etcd/ca.crt --cert-file /etc/kubernetes/pki/etcd/peer.crt --key-file /etc/kubernetes/pki/etcd/peer.key --endpoints=https://192.168.1.11:2379 member add k8s-master3 https://192.168.1.13:2380

# 3. 启动 kube-apiserver、kube-controller-manager、kube-scheduler

sudo kubeadm init phase kubeconfig all --config kubeadm-config.k8s-master3.yaml

sudo kubeadm init phase control-plane all --config kubeadm-config.k8s-master3.yaml

# 4. 将节点标记为 master 节点

sudo kubeadm init phase mark-control-plane --config kubeadm-config.k8s-master3.yaml

[**#**](https://pig4cloud.com/#5.%E5%8A%A0%E5%85%A5%E5%B7%A5%E4%BD%9C%E8%8A%82%E7%82%B9) **5.加入工作节点**

在k8s-slave1和k8s-slave2分别执行我们之前保存的命令

kubeadm join api.k8s.cn:6443 --token g391lr.e64qzy4nb57i9wsj --discovery-token-ca-cert-hash sha256:e96d87f2450978bb64fd97cf6543f660a470c2e05159eff3f588c1377e0de92c

在k8s-master1上执行kubectl get nodes:

NAME STATUS ROLES AGE VERSION

k8s-master1 Ready master 15h v1.13.1

k8s-master2 Ready master 15h v1.13.1

k8s-master3 Ready master 15h v1.13.1

k8s-slave1 Ready <none> 15h v1.13.1

k8s-slave2 Ready <none> 15h v1.13.1

[**#**](https://pig4cloud.com/#6.-%E9%83%A8%E7%BD%B2%E9%AB%98%E5%8F%AF%E7%94%A8-coredns) **6. 部署高可用 CoreDNS**

删除原来的单点CoreDNS

kubectl delete deploy coredns -n kube-system

集群部署

vim coredns-ha.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

k8s-app: kube-dns

name: coredns

namespace: kube-system

spec:

#集群规模可自行配置

replicas: 2

selector:

matchLabels:

k8s-app: kube-dns

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 1

type: RollingUpdate

template:

metadata:

labels:

k8s-app: kube-dns

spec:

affinity:

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

labelSelector:

matchExpressions:

- key: k8s-app

operator: In

values:

- kube-dns

topologyKey: kubernetes.io/hostname

containers:

- args:

- -conf

- /etc/coredns/Corefile

image: registry.cn-hangzhou.aliyuncs.com/google\_containers/coredns:1.2.6

imagePullPolicy: IfNotPresent

livenessProbe:

failureThreshold: 5

httpGet:

path: /health

port: 8080

scheme: HTTP

initialDelaySeconds: 60

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 5

name: coredns

ports:

- containerPort: 53

name: dns

protocol: UDP

- containerPort: 53

name: dns-tcp

protocol: TCP

- containerPort: 9153

name: metrics

protocol: TCP

resources:

limits:

memory: 170Mi

requests:

cpu: 100m

memory: 70Mi

securityContext:

allowPrivilegeEscalation: false

capabilities:

add:

- NET\_BIND\_SERVICE

drop:

- all

readOnlyRootFilesystem: true

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

- mountPath: /etc/coredns

name: config-volume

readOnly: true

dnsPolicy: Default

restartPolicy: Always

schedulerName: default-scheduler

securityContext: {}

serviceAccount: coredns

serviceAccountName: coredns

terminationGracePeriodSeconds: 30

tolerations:

- key: CriticalAddonsOnly

operator: Exists

- effect: NoSchedule

key: node-role.kubernetes.io/master

volumes:

- configMap:

defaultMode: 420

items:

- key: Corefile

path: Corefile

name: coredns

name: config-volume

kubectl apply -f coredns-ha.yaml

[**#**](https://pig4cloud.com/#7.%E9%83%A8%E7%BD%B2-ingress%EF%BC%8C%E6%9C%8D%E5%8A%A1%E6%9A%B4%E9%9C%B2) **7.部署 Ingress，服务暴露**

vim nginx-ingress.yaml

apiVersion: v1

kind: Namespace

metadata:

name: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: nginx-configuration

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: tcp-services

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

kind: ConfigMap

apiVersion: v1

metadata:

name: udp-services

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: nginx-ingress-clusterrole

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

rules:

- apiGroups:

- ""

resources:

- configmaps

- endpoints

- nodes

- pods

- secrets

verbs:

- list

- watch

- apiGroups:

- ""

resources:

- nodes

verbs:

- get

- apiGroups:

- ""

resources:

- services

verbs:

- get

- list

- watch

- apiGroups:

- "extensions"

resources:

- ingresses

verbs:

- get

- list

- watch

- apiGroups:

- ""

resources:

- events

verbs:

- create

- patch

- apiGroups:

- "extensions"

resources:

- ingresses/status

verbs:

- update

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: Role

metadata:

name: nginx-ingress-role

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

rules:

- apiGroups:

- ""

resources:

- configmaps

- pods

- secrets

- namespaces

verbs:

- get

- apiGroups:

- ""

resources:

- configmaps

resourceNames:

- "ingress-controller-leader-nginx"

verbs:

- get

- update

- apiGroups:

- ""

resources:

- configmaps

verbs:

- create

- apiGroups:

- ""

resources:

- endpoints

verbs:

- get

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: nginx-ingress-role-nisa-binding

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: nginx-ingress-role

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: nginx-ingress-clusterrole-nisa-binding

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: nginx-ingress-clusterrole

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

namespace: ingress-nginx

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-ingress-controller

namespace: ingress-nginx

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

spec:

replicas: 3

selector:

matchLabels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

template:

metadata:

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

annotations:

prometheus.io/port: "10254"

prometheus.io/scrape: "true"

spec:

hostNetwork: true

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- k8s-master1

- k8s-master2

- k8s-master3

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app.kubernetes.io/name

operator: In

values:

- ingress-nginx

topologyKey: "kubernetes.io/hostname"

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

serviceAccountName: nginx-ingress-serviceaccount

containers:

- name: nginx-ingress-controller

image: registry.cn-hangzhou.aliyuncs.com/google\_containers/nginx-ingress-controller:0.21.0

args:

- /nginx-ingress-controller

- --configmap=$(POD\_NAMESPACE)/nginx-configuration

- --tcp-services-configmap=$(POD\_NAMESPACE)/tcp-services

- --udp-services-configmap=$(POD\_NAMESPACE)/udp-services

- --annotations-prefix=nginx.ingress.kubernetes.io

securityContext:

capabilities:

drop:

- ALL

add:

- NET\_BIND\_SERVICE

runAsUser: 33

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

livenessProbe:

failureThreshold: 3

httpGet:

path: /healthz

port: 10254

scheme: HTTP

initialDelaySeconds: 10

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 1

readinessProbe:

failureThreshold: 3

httpGet:

path: /healthz

port: 10254

scheme: HTTP

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 1

resources:

limits:

cpu: 1

memory: 1024Mi

requests:

cpu: 0.25

memory: 512Mi

kubectl apply -f nginx-ingress.yaml

[**#**](https://pig4cloud.com/#8.%E9%83%A8%E7%BD%B2dashboard) **8.部署Dashboard**

vim kubernetes-dashboard.yaml

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-certs

namespace: kube-system

type: Opaque

---

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

---

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

rules:

- apiGroups: [""]

resources: ["secrets"]

verbs: ["create"]

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["create"]

- apiGroups: [""]

resources: ["secrets"]

resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs"]

verbs: ["get", "update", "delete"]

- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:"]

verbs: ["get"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: kubernetes-dashboard-minimal

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kube-system

---

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

spec:

containers:

- name: kubernetes-dashboard

image: registry.cn-hangzhou.aliyuncs.com/google\_containers/kubernetes-dashboard-amd64:v1.10.0

ports:

- containerPort: 8443

protocol: TCP

args:

- --auto-generate-certificates

volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs

- mountPath: /tmp

name: tmp-volume

livenessProbe:

httpGet:

scheme: HTTPS

path: /

port: 8443

initialDelaySeconds: 30

timeoutSeconds: 30

volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs

- name: tmp-volume

emptyDir: {}

serviceAccountName: kubernetes-dashboard

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

---

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

ports:

- port: 443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboard

---

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: dashboard-ingress

namespace: kube-system

annotations:

nginx.ingress.kubernetes.io/ssl-redirect: "true"

nginx.ingress.kubernetes.io/rewrite-target: /

nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

spec:

tls:

- secretName: secret-ca-k8s-cn

rules:

- host: k8s.dashboard.cn

http:

paths:

- path: /

backend:

serviceName: kubernetes-dashboard

servicePort: 443

在宿主机配置host 192.168.1.11 k8s.dashboard.cn

浏览器访问：https://k8s.dashboard.cn/

生成访问的Token：

kubectl create sa dashboard-admin -n kube-system

kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin

ADMIN\_SECRET=$(kubectl get secrets -n kube-system | grep dashboard-admin | awk '{print $1}')

DASHBOARD\_LOGIN\_TOKEN=$(kubectl describe secret -n kube-system ${ADMIN\_SECRET} | grep -E '^token' | awk '{print $2}')

echo ${DASHBOARD\_LOGIN\_TOKEN}

[**#**](https://pig4cloud.com/#%E5%87%86%E5%A4%87%E9%83%A8%E7%BD%B2%E9%9C%80%E8%A6%81%E7%9A%84%E9%95%9C%E5%83%8F) **准备部署需要的镜像**

修改好各模块配置文件里的数据库和redis地址！！！所有操作都在k8s-master1机器上进行。

[root@k8s-master1 k8s]# pwd

/project/pigx/k8s

[root@k8s-master1 k8s]# ls

auth config eureka gateway script ui upms

[root@k8s-master1 k8s]# cd eureka/

[root@k8s-master1 eureka]# ls

Dockerfile pigx-eureka.jar

可以看到k8s目录下都是各个模块的文件夹，文件夹内就是各模块构建镜像的JAR包、Dockerfile等。script是部署脚本。

1.部署pigx-eureka

Dockerfile:

FROM anapsix/alpine-java:8\_server-jre\_unlimited

MAINTAINER wangiegie@gmail.com

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN mkdir -p /pigx-eureka

WORKDIR /pigx-eureka

EXPOSE 8761

ADD ./pigx-eureka.jar ./

CMD java -Djava.security.egd=file:/dev/./urandom -jar pigx-eureka.jar

1.构建镜像

docker build -t pigx-eureka:0.0.1 .

2.标记镜像

docker tag pigx-eureka:0.0.1 k8s.registry.cn/pigx/pigx-eureka:0.0.1

3.推送镜像

docker push k8s.registry.cn/pigx/pigx-eureka:0.0.1

部署脚本：k8s-pigx-eureka.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-eureka

name: pigx-eureka

spec:

ports:

- port: 8761

protocol: TCP

targetPort: 8761

selector:

app: pigx-eureka

type: NodePort

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-eureka-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-eureka

spec:

containers:

- name: pigx-eureka

image: k8s.registry.cn/pigx/pigx-eureka:0.0.1

ports:

- containerPort: 8761

---

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

name: pigx-eureka-ingress

spec:

rules:

- host: k8s.eureka.cn

http:

paths:

- backend:

serviceName: pigx-eureka

servicePort: 8761

kubectl kubectl apply -f k8s-pigx-eureka.yaml

2.部署pigx-config

Dockerfile:

FROM anapsix/alpine-java:8\_server-jre\_unlimited

MAINTAINER maojiajiajia@gmail.com

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN mkdir -p /pigx-config

WORKDIR /pigx-config

EXPOSE 8888

ADD ./pigx-config.jar ./

CMD java -Djava.security.egd=file:/dev/./urandom -jar pigx-config.jar

构建镜像同上只需改一下名称。

部署脚本： k8s-pigx-config.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-config

name: pigx-config

spec:

ports:

- port: 8888

protocol: TCP

targetPort: 8888

selector:

app: pigx-config

type: ClusterIP

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-config-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-config

spec:

containers:

- name: pigx-config

image: k8s.registry.cn/pigx/pigx-config:0.0.1

ports:

- containerPort: 8888

kubectl kubectl apply -f k8s-pigx-config.yaml

3.部署pigx-auth

Dockerfile：

FROM anapsix/alpine-java:8\_server-jre\_unlimited

MAINTAINER maojiajiajia@gmail.com

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN mkdir -p /pigx-auth

WORKDIR /pigx-auth

EXPOSE 3000

ADD ./pigx-auth.jar ./

CMD java -Djava.security.egd=file:/dev/./urandom -jar pigx-auth.jar

部署脚本：k8s-pigx-auth.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-auth

name: pigx-auth

spec:

ports:

- port: 3000

protocol: TCP

targetPort: 3000

selector:

app: pigx-auth

type: ClusterIP

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-auth-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-auth

spec:

containers:

- name: pigx-auth

image: k8s.registry.cn/pigx/pigx-auth:0.0.1

ports:

- containerPort: 3000

kubectl kubectl apply -f k8s-pigx-auth.yaml

4.部署pigx-gateway

Dockerfile:

FROM anapsix/alpine-java:8\_server-jre\_unlimited

MAINTAINER wangiegie@gmail.com

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN mkdir -p /pigx-gateway

WORKDIR /pigx-gateway

EXPOSE 9999

ADD ./pigx-gateway.jar ./

CMD java -Djava.security.egd=file:/dev/./urandom -jar pigx-gateway.jar

部署脚本：k8s-pigx-gateway.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-gateway-service

name: pigx-gateway-service

spec:

ports:

- port: 9999

protocol: TCP

targetPort: 9999

selector:

app: pigx-gateway

type: ClusterIP

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-gateway-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-gateway

spec:

containers:

- name: pigx-gateway

image: k8s.registry.cn/pigx/pigx-gateway:0.0.2

ports:

- containerPort: 9999

kubectl kubectl apply -f k8s-pigx-gateway.yaml

5.部署pigx-upms

Dockerfile：

FROM anapsix/alpine-java:8\_server-jre\_unlimited

MAINTAINER wangiegie@gmail.com

RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN mkdir -p /pigx-upms

WORKDIR /pigx-upms

EXPOSE 4000

ADD ./pigx-upms-biz.jar ./

CMD java -Djava.security.egd=file:/dev/./urandom -jar pigx-upms-biz.jar

部署脚本：k8s-pigx-upms.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-upms

name: pigx-upms

spec:

ports:

- port: 4000

protocol: TCP

targetPort: 4000

selector:

app: pigx-upms

type: ClusterIP

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-upms-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-upms

spec:

containers:

- name: pigx-upms

image: k8s.registry.cn/pigx/pigx-upms:0.0.1

ports:

- containerPort: 4000

kubectl kubectl apply -f k8s-pigx-upms.yaml

如果不知道镜像的名称是什么可以看部署脚本里的image:标签。

6.部署pigx-ui 需要注意的是我也不知道为什么nginx部署前端静态文件时，不会代理验证码的路径，所以我就把验证码注掉了，一个是前端的登录页的，二是网关验证，大家自己找找注释掉就行。 我们需要提前把前端UI打好包，上传至虚拟机里，再去构建镜像，这样简单~

[root@k8s-master1 k8s]# cd ui/

[root@k8s-master1 ui]# ls

default.conf dist dist.zip Dockerfile

修改前端vue.config.js里的后端路径：

const url = 'http://pigx-gateway-service:9999';

打包前端项目：

npm run build

NGINX配置文件 default.conf:

server {

listen 9000;

server\_name localhost;

location / {

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri $uri/ @router;

}

location @router {

rewrite ^.\*$ /index.html last;

}

location ^~ /code/ {

proxy\_redirect 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;

proxy\_pass http://pigx-gateway-service:9999/code/;

}

location ^~ /auth/ {

proxy\_redirect 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;

proxy\_pass http://pigx-gateway-service:9999/auth/;

}

location ^~ /admin/ {

proxy\_redirect 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;

proxy\_pass http://pigx-gateway-service:9999/admin/;

}

}

Dockerfile:

FROM hub.c.163.com/library/nginx

MAINTAINER Maozk <549595297@qq.com>

RUN rm /etc/nginx/conf.d/default.conf

ADD default.conf /etc/nginx/conf.d/

COPY dist/ /usr/share/nginx/html/

部署脚本：k8s-pigx-ui.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: pigx-ui

name: pigx-ui

spec:

ports:

- port: 9000

protocol: TCP

targetPort: 9000

nodePort: 30000

selector:

app: pigx-ui

type: NodePort

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pigx-ui-deployment

spec:

replicas: 1

template:

metadata:

labels:

app: pigx-ui

spec:

containers:

- name: pigx-ui

image: k8s.registry.cn/pigx/pigx-ui:0.0.7

ports:

- containerPort: 9000

---

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

name: pigx-ui-ingress

spec:

rules:

- host: k8s.pigxui.cn

http:

paths:

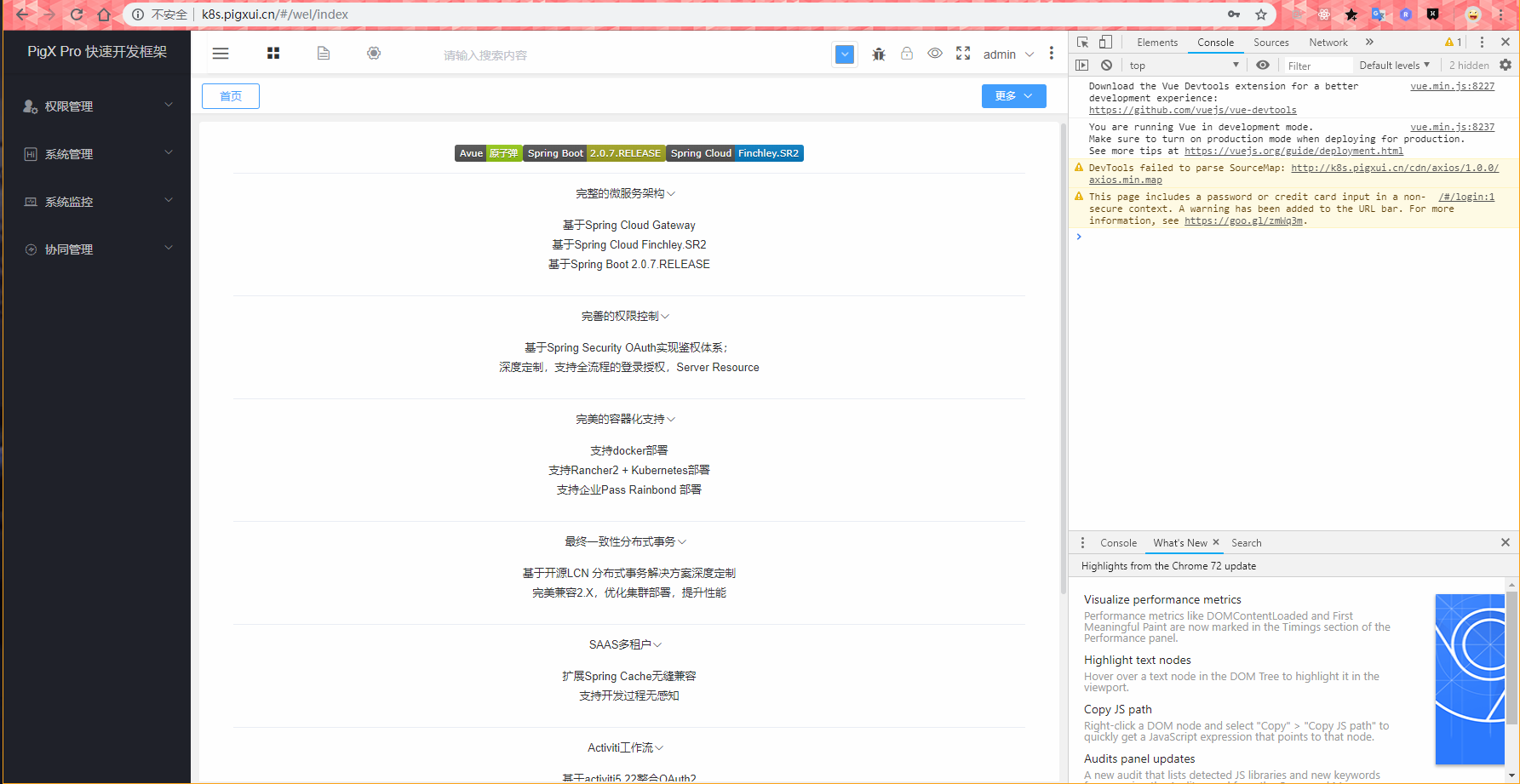
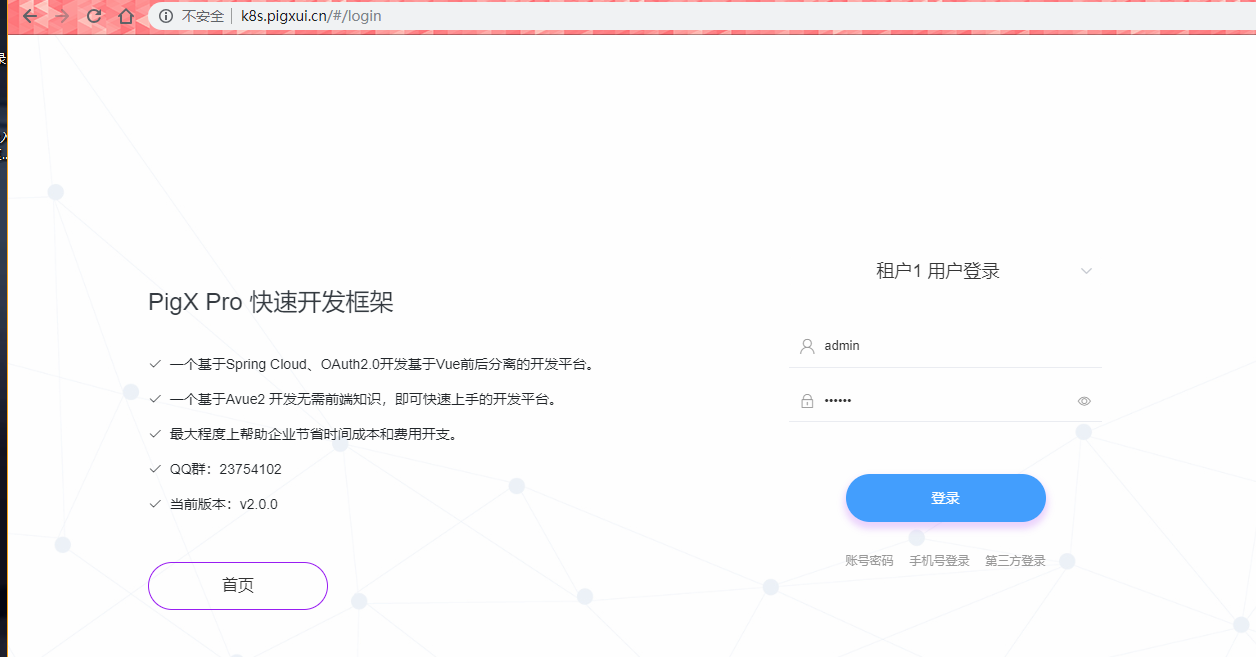
- backend:

serviceName: pigx-ui

servicePort: 9000

配置宿主机hosts：

192.168.1.11 k8s.dashboard.cn k8s.eureka.cn k8s.pigxui.cn

浏览器访问：http://k8s.pigxui.cn/#/login 

[**#**](https://pig4cloud.com/#%E6%80%BB%E7%BB%93) **总结**

本文没有对原理进行讲解，只是说了一下不说的过程。欢迎大家@我(@githink.cn)，和我一起讨论一起学习一起成长。

[**#**](https://pig4cloud.com/#%E6%9C%80%E5%90%8E) **最后**

打个广告:

1.[Kubernetes 持续集成 SpringCloud](https://gitbook.cn/gitchat/activity/5b728973a281bb5781de5f18) 本场 Chat 会从零开始教会大家如何将我们的应用持续的，分场景、环境的持续的部署到 Kubernetes 集群中，以及我们的应用如何在不同的场景下，平滑的升级，以及构建我们自己的 Docker 私有仓库、代码托管服务、Jenkins 构建服务。

包括设置固定ip，部署mysql，redis等。

2.[手把手教你使用 Kotlin 打造通用 Android MVP 架构](https://gitbook.cn/gitchat/activity/5bdfa822553a12719894583c)本场 Chat 会从零开始教会大家如何使用 Kotlin 构建一个模块化的 MVP 架构（附源码），什么是 MVP 架构、模块间的通信、用户在视图上的操作到网络请求的整个数据流向。包括像 Anko、Retrofit、Dagger2、ARouter 等一些第三方插件的使用和封装以及常见问题。最后使用框架完成注册登录流程。