**[.Net微服务实战之Kubernetes的搭建与使用](https://www.cnblogs.com/skychen1218/p/13441778.html)**

**系列文章**

* [.Net微服务实战之技术选型篇](https://www.cnblogs.com/skychen1218/p/12531412.html)
* [.Net微服务实战之技术架构分层篇](https://www.cnblogs.com/skychen1218/p/12653155.html)
* [.Net微服务实战之DevOps篇](https://www.cnblogs.com/skychen1218/p/13096784.html)
* [.Net微服务实战之负载均衡（上）](https://www.cnblogs.com/skychen1218/p/13327965.html)
* [.Net微服务实战之CI/CD](https://www.cnblogs.com/skychen1218/p/13384073.html)

**前言**

　　说到微服务就得扯到自动化运维，然后别人就不得不问你用没用上K8S。无论是概念上还是在实施搭建时，K8S的门槛比Docker Compose、Docker Swarm高了不少。我自己也经过了多次的实践，整理出一套顺利部署的流程。

　　我这次搭建花了一共整整4个工作实践与一个工作日写博客，中间有一个网络问题导致reset了集群重新搭了一次，完成后结合了Jenkins使用，还是成就感满满的。如果对大家有用，还请点个推荐与关注。

**基本概念**

**Kubectl**

kubectl用于运行Kubernetes集群命令的管理工具，Kubernetes kubectl 与 Docker 命令关系可以查看这里

http://docs.kubernetes.org.cn/70.html

**Kubeadm**

kubeadm 是 kubernetes 的集群安装工具，能够快速安装 kubernetes 集群，相关命令有以下:

kubeadm init

kubeadm join

**Kubelet**

kubelet是主要的节点代理，它会监视已分配给节点的pod，具体功能：

* 安装Pod所需的volume。
* 下载Pod的Secrets。
* Pod中运行的 docker（或experimentally，rkt）容器。
* 定期执行容器健康检查。

**Pod**

Pod是Kubernetes创建或部署的最小(最简单)的基本单位，一个Pod代表集群上正在运行的一个进程，它可能由单个容器或多个容器共享组成的资源。

一个Pod封装一个应用容器（也可以有多个容器），存储资源、一个独立的网络IP以及管理控制容器运行方式的策略选项。

Pods提供两种共享资源：网络和存储。

**网络**

每个Pod被分配一个独立的IP地址，Pod中的每个容器共享网络命名空间，包括IP地址和网络端口。Pod内的容器可以使用localhost相互通信。当Pod中的容器与Pod 外部通信时，他们必须协调如何使用共享网络资源（如端口）。

**存储**

Pod可以指定一组共享存储*volumes*。Pod中的所有容器都可以访问共享*volumes*，允许这些容器共享数据。*volumes*还用于Pod中的数据持久化，以防其中一个容器需要重新启动而丢失数据。

**Service**

一个应用服务在Kubernetes中可能会有一个或多个Pod，每个Pod的IP地址由网络组件动态随机分配（Pod重启后IP地址会改变）。为屏蔽这些后端实例的动态变化和对多实例的负载均衡，引入了Service这个资源对象。

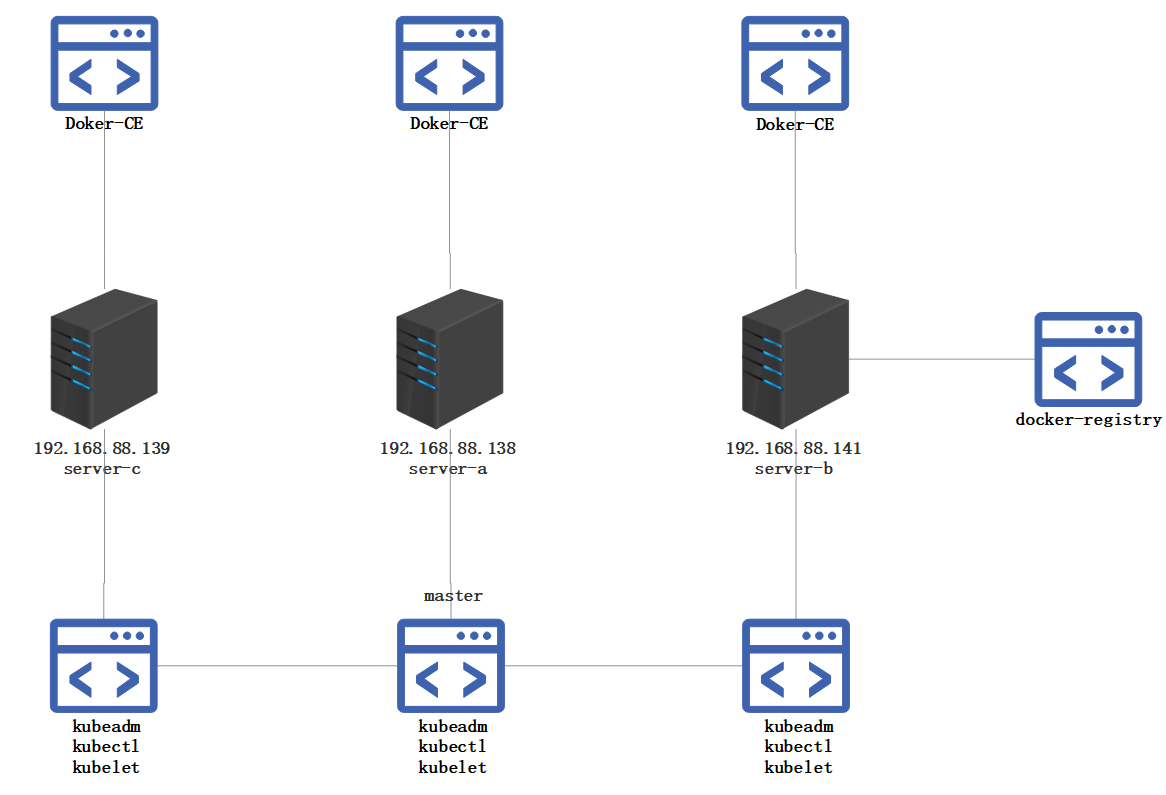
Kubernetes ServiceTypes 允许指定一个需要的类型的 Service，默认是 ClusterIP 类型。

Type 的取值以及行为如下：

* ClusterIP：通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部可以访问，这也是默认的 ServiceType。
* NodePort：通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 <NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务。
* LoadBalancer：使用云提供商的负载局衡器，可以向外部暴露服务。外部的负载均衡器可以路由到 NodePort 服务和 ClusterIP 服务。
* ExternalName：通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容（例如， foo.bar.example.com）。 没有任何类型代理被创建，这只有 Kubernetes 1.7 或更高版本的 kube-dns 才支持。

*其他详细的概念请移步到*[*http://docs.kubernetes.org.cn/227.html*](http://docs.kubernetes.org.cn/227.html)

**物理部署图**



**Docker-ce 1.19安装**

**在所有需要用到kubernetes服务器上安装docker-ce**

卸载旧版本 docker

yum remove docker docker-common docker-selinux dockesr-engine -y

升级系统软件

yum upgrade -y

安装必要的一些系统工具

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

添加docker-ce软件源

yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

更新并安装 docker-ce

yum makecache fast

yum install docker-ce-19.03.12 -y

添加docker国内镜像源

vim /etc/docker/daemon.json

{

"exec-opts": ["native.cgroupdriver=systemd"],

"registry-mirrors" : [

"http://ovfftd6p.mirror.aliyuncs.com",

"http://registry.docker-cn.com",

"http://docker.mirrors.ustc.edu.cn",

"http://hub-mirror.c.163.com"

],

"insecure-registries" : [

"registry.docker-cn.com",

"docker.mirrors.ustc.edu.cn"

],

"debug" : true,

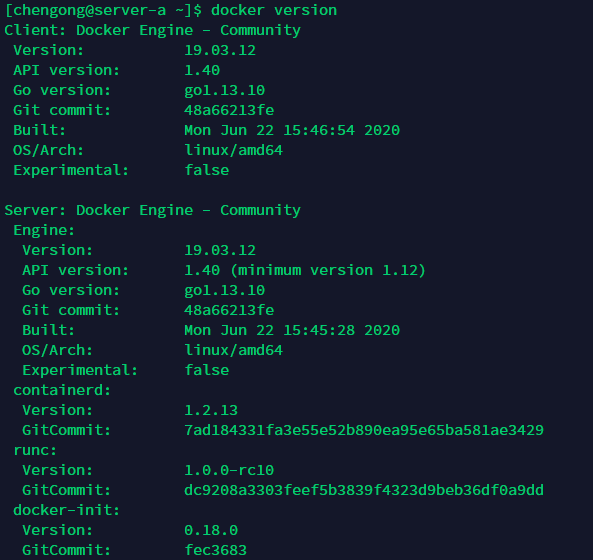
"experimental" : true

}

启动服务

systemctl start docker

systemctl enable docker



**安装kubernetes-1.18.3**

**所有需要用到kubernetes的服务器都执行以下指令。**

添加阿里kubernetes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=1

repo\_gpgcheck=1

gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

EOF

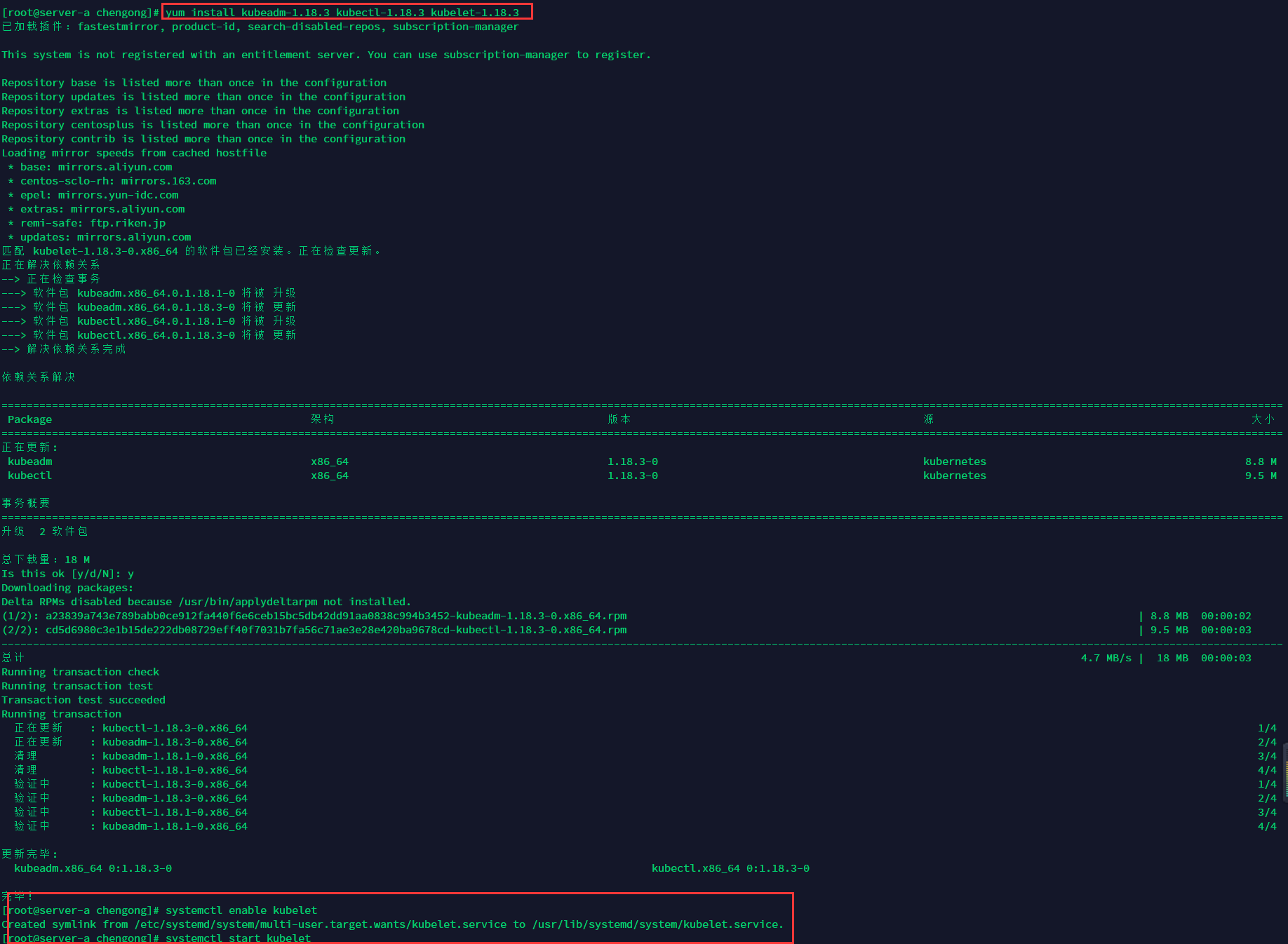
安装并启动

yum install kubeadm-1.18.3 kubectl-1.18.3 kubelet-1.18.3

启动kubelet

systemctl enable kubelet

systemctl start kubelet



在Master设置环境变量，在/etc/profile中配置

vim /etc/profile

在最后添加如下配置

export KUBECONFIG=/etc/kubernetes/admin.conf

执行命令使其起效

source /etc/profile

**初始化k8s集群**

**在master节点（server-a）进行初始化集群**

开放端口

firewall-cmd --permanent --zone=public --add-port=6443/tcp

firewall-cmd --permanent --zone=public --add-port=10250/tcp

firewall-cmd --reload

关闭swap

vim /etc/fstab

#注释swap那行， 需要重启Linux

swapoff -a

#

# /etc/fstab

# Created by anaconda on Fri Aug 25 05:49:45 2017

#

# Accessible filesystems, by reference, are maintained under '/dev/disk'

# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info

#

/dev/mapper/cl-root / xfs defaults 0 0

UUID=c23a4943-bae1-4f02-868a-0cef19d09c5d /boot xfs defaults 0 0

/dev/mapper/cl-home /home xfs defaults 0 0

#/dev/mapper/cl-swap swap swap defaults 0 0

设置iptables规则

echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables

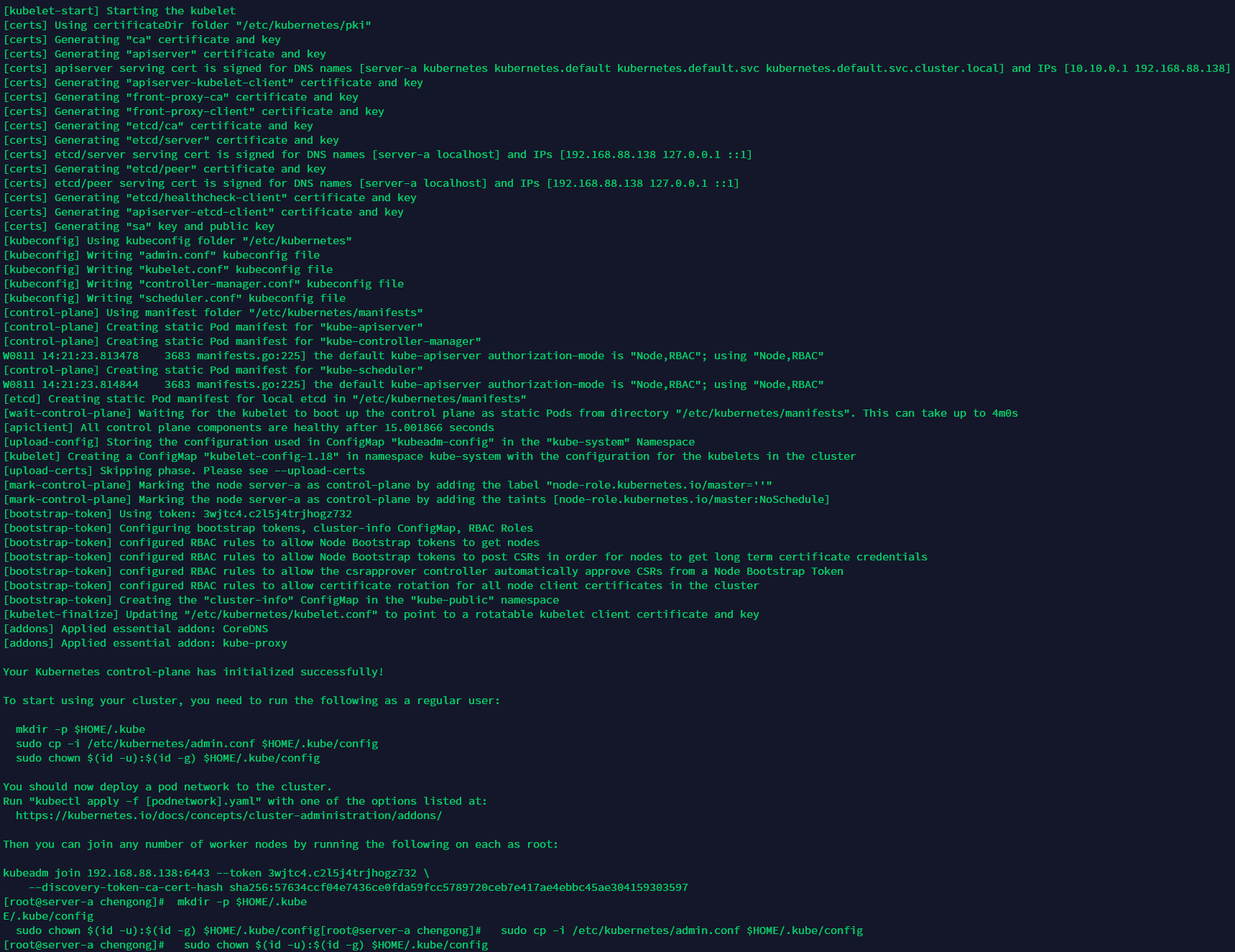
echo 1 > /proc/sys/net/bridge/bridge-nf-call-ip6tables

初始化

kubeadm init --kubernetes-version=1.18.3  --apiserver-advertise-address=10.63.207.23   --image-repository registry.aliyuncs.com/google\_containers  --service-cidr=10.10.0.0/16 --pod-network-cidr=10.122.0.0/16 --ignore-preflight-errors=Swap

　　pod-network-cidr参数的为pod网段:，apiserver-advertise-address参数为本机IP。

　　如果中途执行有异常可以通过 kubeadm reset 后重新init。



Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

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

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

You should now deploy a pod network to the cluster.

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

https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

**kubeadm join 10.63.207.23:6443 --token mhsasm.8di9rxaucfm31bb2 \**

**--discovery-token-ca-cert-hash sha256:4f515e4acebe8d80295e2ea738395b28a96e4ec0769285fd3020bf3a2bb59be9**

初始化成功执行下面指令

mkdir -p $HOME/.kube

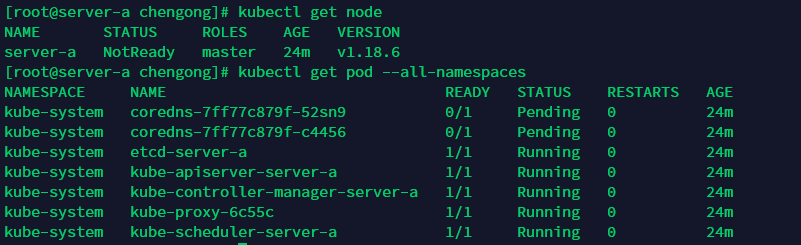
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

查看node和pod信息

kubectl get node

kubectl get pod --all-namespaces



**安装flannel组件**

**在master节点（server-a）安装flannel组件**

**方法1：**

6. 安装Pod网络插件（CNI）

$ **kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml**

#查看是否启动完毕

$ kubectl get pods -n kube-system

————————————————

版权声明：本文为CSDN博主「陈先生灬」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/a251628111/article/details/106521745>

**方法2：**

找个梯子下载kube-flannel.yml文件

https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

下载不了也没关系，我复制给到大家：

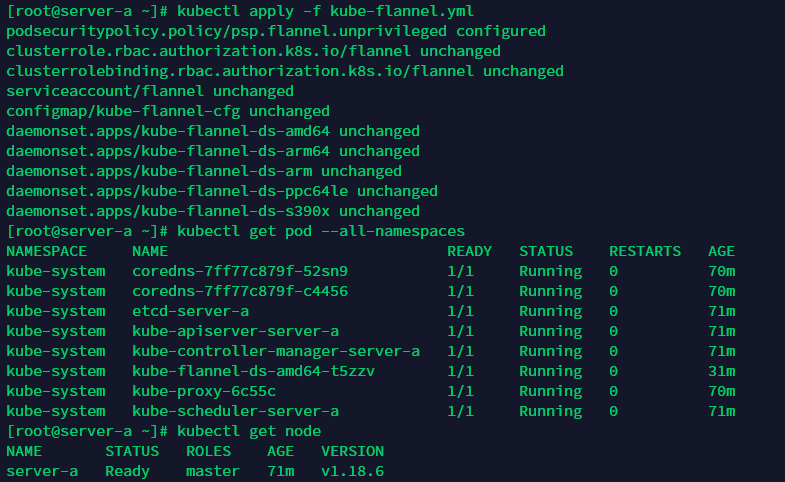
https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

先拉取依赖镜像

docker pull quay.io/coreos/flannel:v0.12.0-amd64

把上面文件保存到服务器然后执行下面命令

kubectl apply -f kube-flannel.yml



**安装dashboard**

**在master节点（server-a）安装dashboard组件**

继续用梯子下载recommended.yml文件

https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml

没梯子的可以复制下方原文件

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

# Copyright 2017 The Kubernetes Authors.

#

# 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.

apiVersion: v1

kind: Namespace

metadata:

name: kubernetes-dashboard

---

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

---

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

spec:

ports:

- port: 443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboard

---

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-certs

namespace: kubernetes-dashboard

type: Opaque

---

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-csrf

namespace: kubernetes-dashboard

type: Opaque

data:

csrf: ""

---

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-key-holder

namespace: kubernetes-dashboard

type: Opaque

---

kind: ConfigMap

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-settings

namespace: kubernetes-dashboard

---

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

rules:

# Allow Dashboard to get, update and delete Dashboard exclusive secrets.

- apiGroups: [""]

resources: ["secrets"]

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

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

# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

# Allow Dashboard to get metrics.

- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster", "dashboard-metrics-scraper"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:", "dashboard-metrics-scraper", "http:dashboard-metrics-scraper"]

verbs: ["get"]

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

rules:

# Allow Metrics Scraper to get metrics from the Metrics server

- apiGroups: ["metrics.k8s.io"]

resources: ["pods", "nodes"]

verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: kubernetes-dashboard

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: kubernetes-dashboard

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: kubernetes-dashboard

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kubernetes-dashboard

---

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

spec:

containers:

- name: kubernetes-dashboard

image: kubernetesui/dashboard:v2.0.3

imagePullPolicy: Always

ports:

- containerPort: 8443

protocol: TCP

args:

- --auto-generate-certificates

- --namespace=kubernetes-dashboard

# 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://my-address:port

volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs

# Create on-disk volume to store exec logs

- mountPath: /tmp

name: tmp-volume

livenessProbe:

httpGet:

scheme: HTTPS

path: /

port: 8443

initialDelaySeconds: 30

timeoutSeconds: 30

securityContext:

allowPrivilegeEscalation: false

readOnlyRootFilesystem: true

runAsUser: 1001

runAsGroup: 2001

volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs

- name: tmp-volume

emptyDir: {}

serviceAccountName: kubernetes-dashboard

nodeSelector:

"kubernetes.io/os": linux

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

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

effect: NoSchedule

---

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: dashboard-metrics-scraper

name: dashboard-metrics-scraper

namespace: kubernetes-dashboard

spec:

ports:

- port: 8000

targetPort: 8000

selector:

k8s-app: dashboard-metrics-scraper

---

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: dashboard-metrics-scraper

name: dashboard-metrics-scraper

namespace: kubernetes-dashboard

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: dashboard-metrics-scraper

template:

metadata:

labels:

k8s-app: dashboard-metrics-scraper

annotations:

seccomp.security.alpha.kubernetes.io/pod: 'runtime/default'

spec:

containers:

- name: dashboard-metrics-scraper

image: kubernetesui/metrics-scraper:v1.0.4

ports:

- containerPort: 8000

protocol: TCP

livenessProbe:

httpGet:

scheme: HTTP

path: /

port: 8000

initialDelaySeconds: 30

timeoutSeconds: 30

volumeMounts:

- mountPath: /tmp

name: tmp-volume

securityContext:

allowPrivilegeEscalation: false

readOnlyRootFilesystem: true

runAsUser: 1001

runAsGroup: 2001

serviceAccountName: kubernetes-dashboard

nodeSelector:

"kubernetes.io/os": linux

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

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

effect: NoSchedule

volumes:

- name: tmp-volume

emptyDir: {}

第39行修改，端口范围30000-32767

spec:

type: NodePort

ports:

- port: 443

targetPort: 8443

nodePort: 30221

selector:

k8s-app: kubernetes-dashboard

第137行开始，修改账户权限，主要三个参数，**kind: ClusterRoleBinding，roleRef-kind: ClusterRole，roleRef-name: cluster-admin**

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kubernetes-dashboard

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

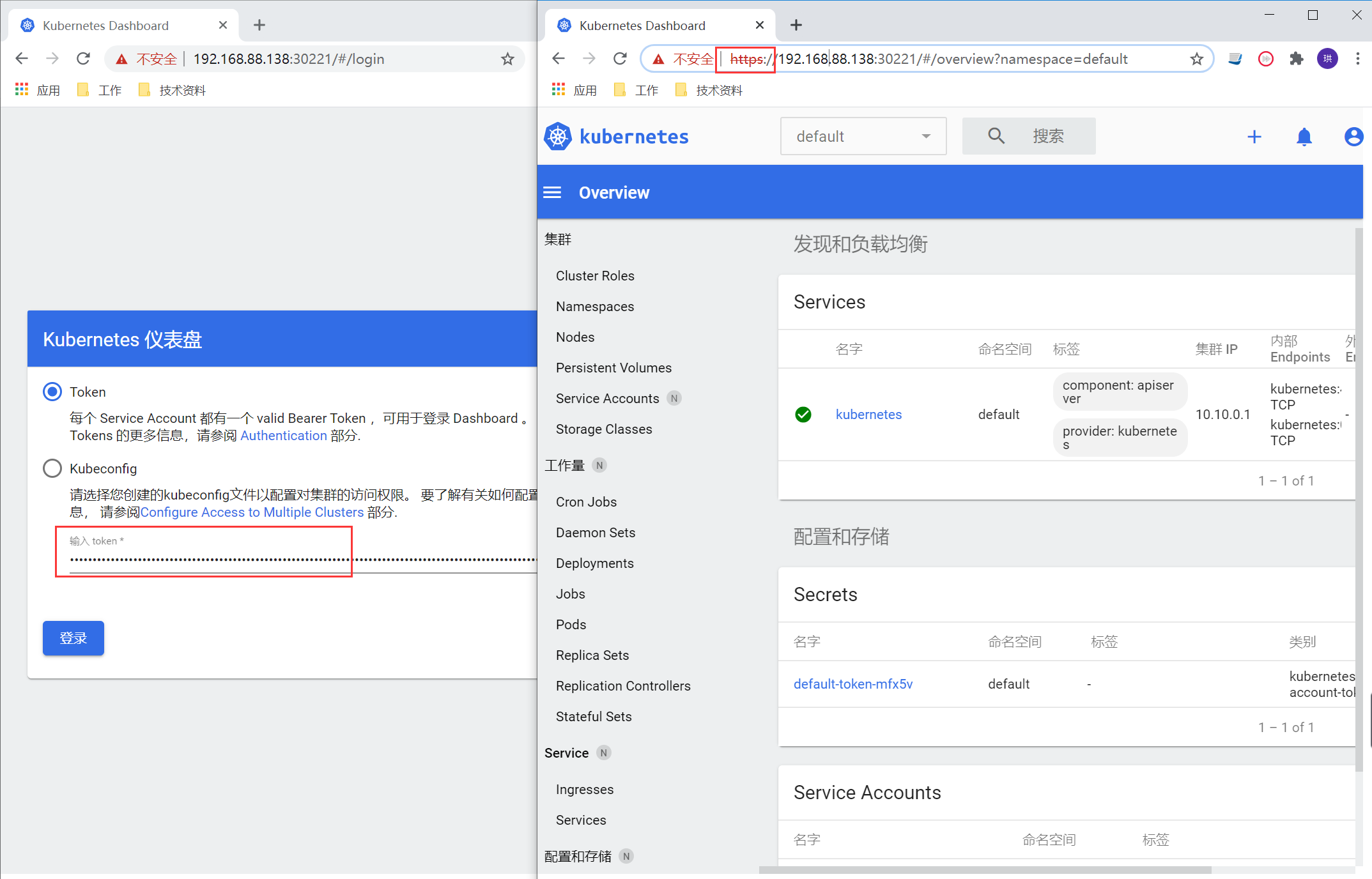
namespace: kubernetes-dashboard

---

保存到服务器后执行以下命令

kubectl apply -f recommended.yaml

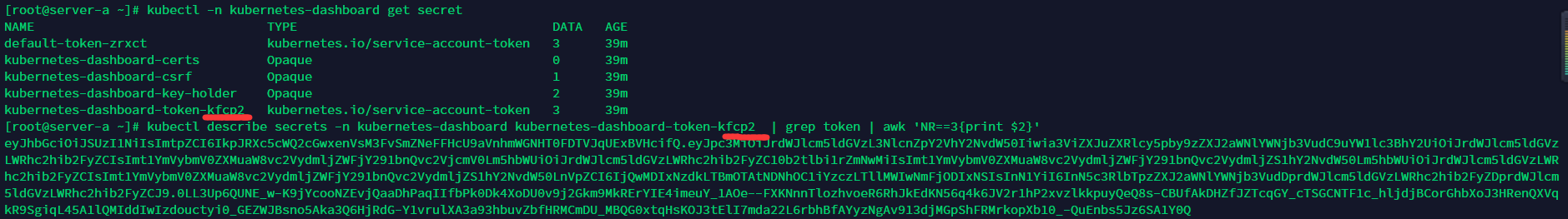
 等待一段时间启动成功后，**https://ip+nodePort**，查看UI



Token通过下面指令获取

kubectl -n kubernetes-dashboard get secret

kubectl describe secrets -n kubernetes-dashboard kubernetes-dashboard-token-kfcp2 | grep token | awk 'NR==3{print $2}'



3.部署Dashboard，并创建绑定cluster-admin角色的ServiceAccount —— admin-user (参考 [auth.yaml](https://github.com/ronwxy/ops/tree/master/k8s/installation/dashboard/kubernetes-dashboard-auth.yaml" \t "_blank))

kubectl apply -f kubernetes-dashboard.yaml

kubectl apply -f kubernetes-dashboard-auth.yaml

4.访问Dashboard

访问 [https://集群任意节点IP:30443](https://xn--ip-dh3cr99d42rvou4sez72c:30443/)，打开Dashboard登录页面，执行如下命令获取登录token

kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')

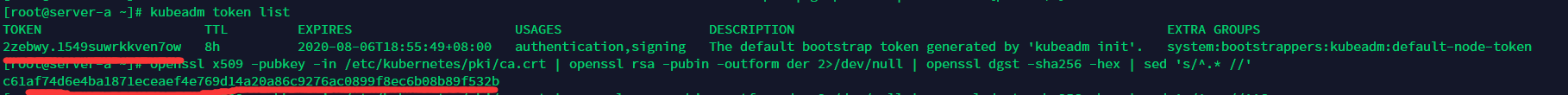
**加入Worker节点**

**在server-b与server-c执行下面操作**

把上面init后的那句join拷贝过来，如果忘记了可以在master节点执行下面指令：

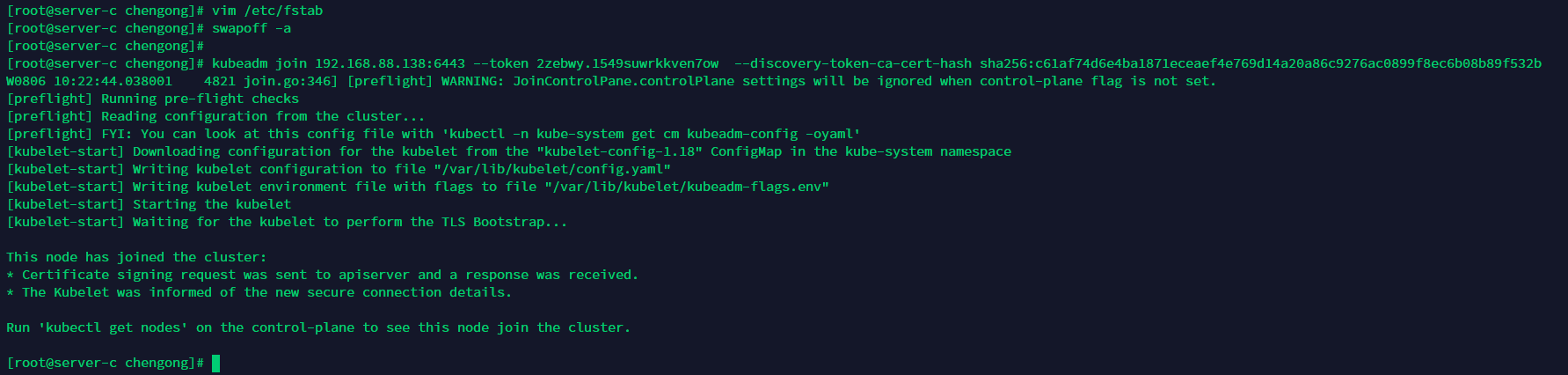
kubeadm token list

openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.\* //'



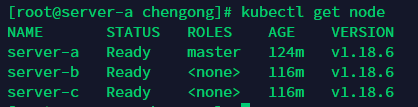
通过返回的数据拼装成下面指令

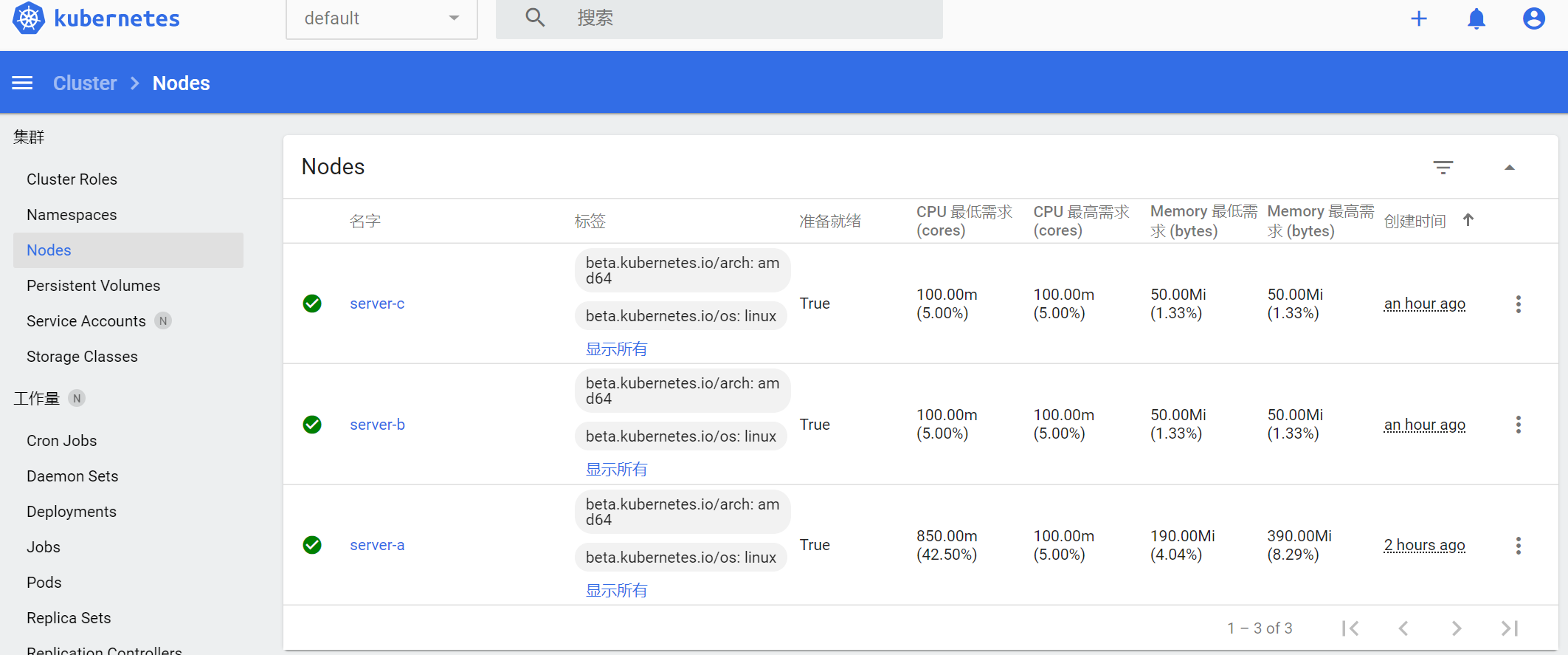
kubeadm join 192.168.88.138:6443 --token 2zebwy.1549suwrkkven7ow --discovery-token-ca-cert-hash sha256:c61af74d6e4ba1871eceaef4e769d14a20a86c9276ac0899f8ec6b08b89f532b



查看节点信息

kubectl get node





**部署Web应用**

**在master节点（sever-a）执行下面操作**

　　部署应用前建议有需要的朋友到【[.Net微服务实战之CI/CD](https://www.cnblogs.com/skychen1218/p/13384073.html)】看看如何搭建docker私有仓库，后面需要用到，搭建后私有库后执行下面指令

kubectl create secret docker-registry docker-registry-secret --docker-server=192.168.88.141:6000 --docker-username=admin --docker-password=123456789

　　docker-server就是docker私有仓库的地址

　　下面是yaml模板，注意imagePullSecrets-name与上面的命名的一致，其余的可以查看yaml里的注释

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

apiVersion: apps/v1

kind: Deployment # Deployment为多个Pod副本

metadata:

name: testdockerswarm-deployment

labels:

app: testdockerswarm-deployment

spec:

replicas: 2 # 实例数量

selector:

matchLabels: # 定义该部署匹配哪些Pod

app: testdockerswarm

minReadySeconds: 3 # 可选，指定Pod可以变成可用状态的最小秒数，默认是0

strategy:

type: RollingUpdate # 部署策略类型，使用RollingUpdate可以保证部署期间服务不间断

rollingUpdate:

maxUnavailable: 1 # 部署时最大允许停止的Pod数量

maxSurge: 1 # 部署时最大允许创建的Pod数量

template: # 用来指定Pod的模板，与Pod的定义类似

metadata:

labels: # Pod标签，与上面matchLabels对应

app: testdockerswarm

spec:

imagePullSecrets:

- name: docker-registry-secret

containers:

- name: testdockerswarm

image: 192.168.88.141:6000/testdockerswarm

imagePullPolicy: Always # Always每次拉去新镜像

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: testdockerswarm-service

labels:

name: testdockerswarm-service

spec:

selector:

app: testdockerswarm #与template-labels参数pod标签一致

ports:

- protocol: TCP

port: 80 #clusterIP开放的端口

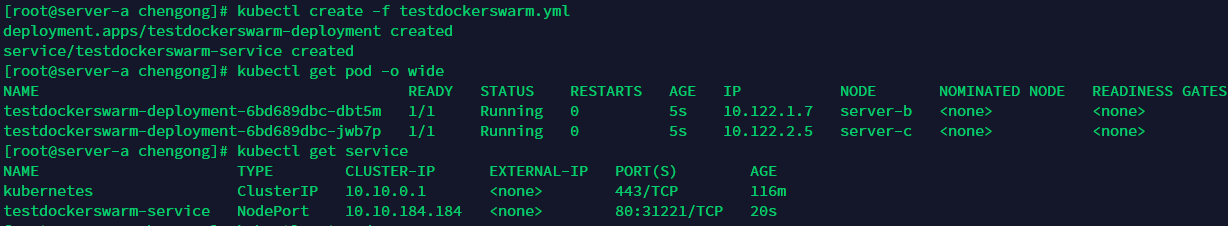
targetPort: 80 #container开放的端口，与containerPort一致

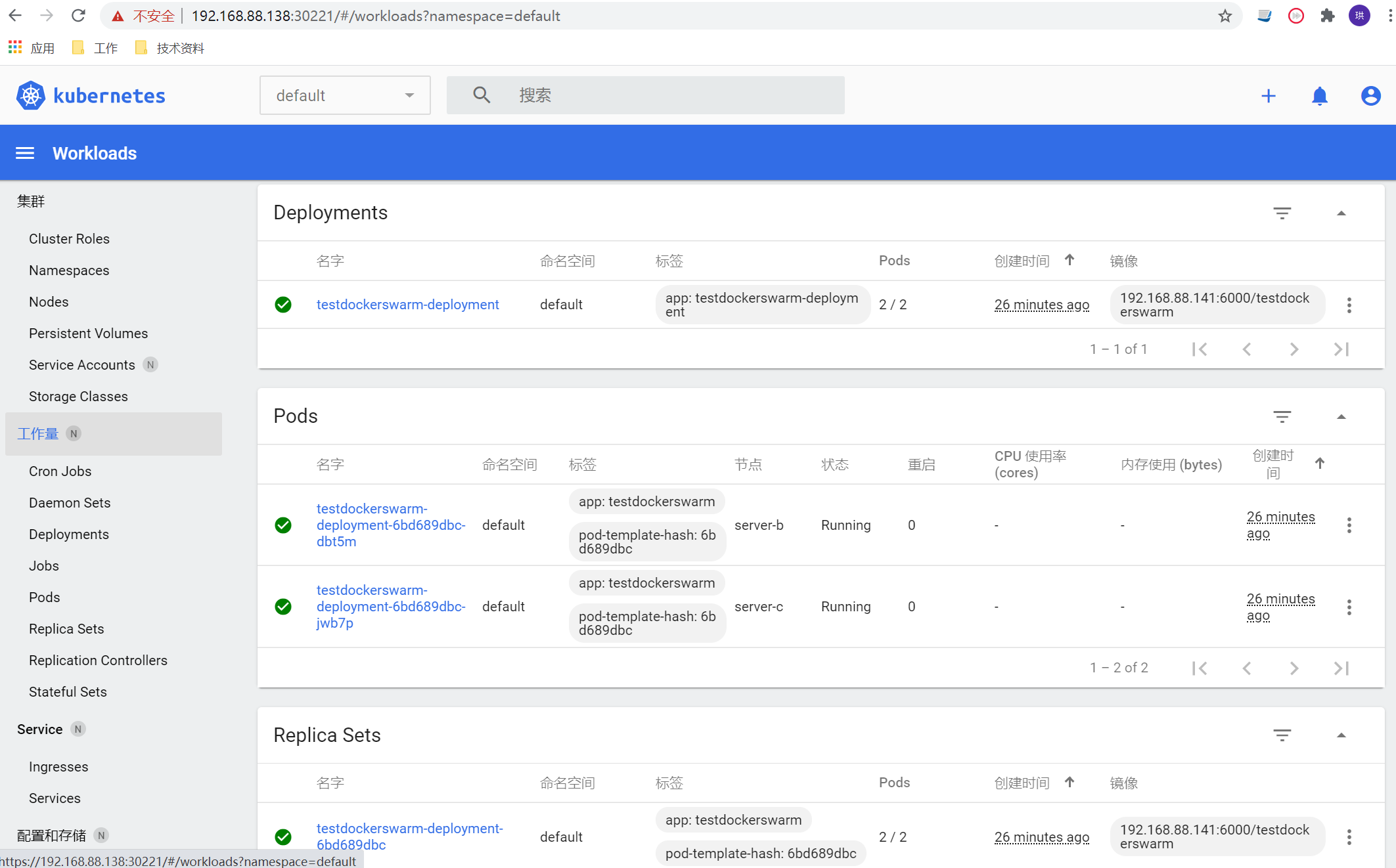
nodePort: 31221 # 所有的节点都会开放此端口，此端口供外部调用。

type: NodePort

　　把yaml文件保存到服务器后执行下面命令

kubectl create -f testdockerswarm.yml

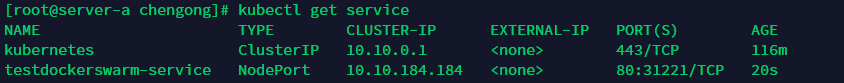




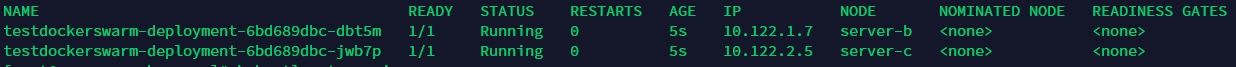
　　整个搭建部署的过程基本上到这里结束了。

**访问**

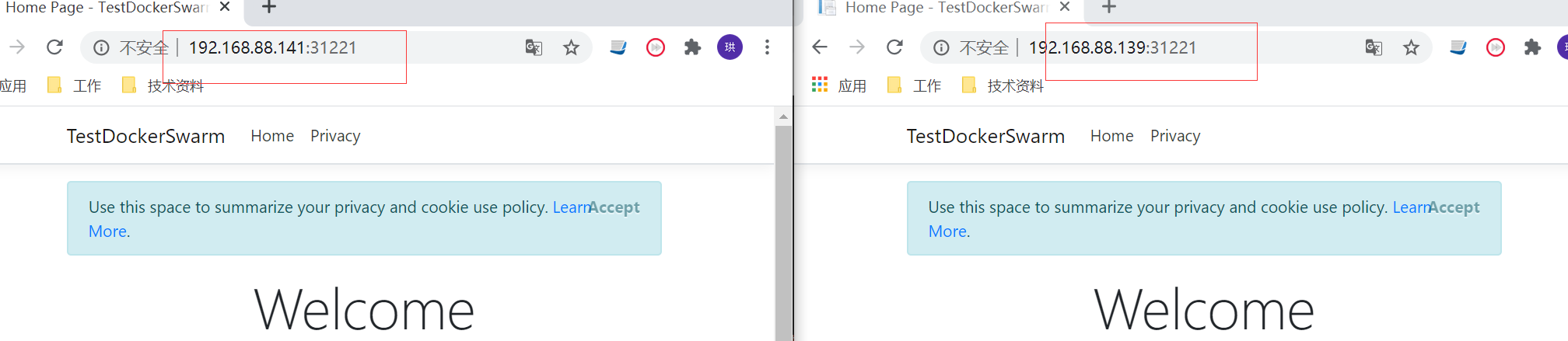
　　可以通过指令**kubectl get service**得到ClusterIP，分别在server-c和sever-b执行curl 10.10.184.184



　　也可以通过执行**kubectl get pods -o wide**得到pod ip，在server-c执行curl 10.122.2.5 和 server-b执行curl 10.122.1.7



　　也可以在外部访问 server-c和server-b的 ip + 31221



　　如果节点有异常可以通过下面指令排查

journalctl -f -u kubelet.service | grep -i error -C 500

　　如果Pod无法正常running可以通过下面指令查看

kubectl describe pod testdockerswarm-deployment-7bc647d87d-qwvzm

**测试kubernetes集群**

在Kubernetes集群中创建一个pod，验证是否正常运行：

$ kubectl create deployment nginx --image=nginx

$ kubectl expose deployment nginx --port=80 --type=NodePort

$ kubectl get pod,svc

#扩容3个nginx

$ kubectl scale deployment nginx --replicas=3