1 Kubernetes集群机器信息

1.1 机器信息

Kubernetes集群机器编号、IP信息及原机器归属信息如下：

编号	IP	原归属	机器型号	CPU+Mem	备注
1	192.168.1.107	原KV集群	Dell R740	Intel Xeon Gold 5115 CPU@2.4GHz，2NUMA节点，每节点10个核，开启HT后40 processors 64G Mem	
2	192.168.1.104	王千里	曙光I620-G20	Intel Xeon E5-2650 CPU v4@2.2GHz，2NUMA节点，每节点12个核，开启HT后48 processors 64G Mem	
3	192.168.1.118	董健	曙光I620-G20	Intel Xeon E5-2650 CPU v4@2.2GHz，2NUMA节点，每节点12个核，开启HT后48 processors 64G Mem	
4	192.168.1.119	原KV集群	Dell R740	Intel Xeon Gold 5115 CPU@2.4GHz，2NUMA节点，每节点10个核，开启HT后40 processors 64G Mem	
5	192.168.1.114	原KV集群	Dell R740	Intel Xeon Gold 5115 CPU@2.4GHz，2NUMA节点，每节点10个核，开启HT后40 processors 64G Mem	
6	192.168.1.109	原KV集群	Dell R740	Intel Xeon Gold 5115 CPU@2.4GHz，2NUMA节点，每节点10个核，开启HT后40 processors 64G Mem	
7	192.168.1.134	林帅	Dell R720	Intel Xeon E5-2650 CPU v2@2.6GHz，2NUMA节点，每节点8个核，开启HT后32 processors 64G Mem	
8	192.168.1.113	左泽	Dell R730	Intel Xeon E5-2650 CPU v4@2.2GHz，2NUMA节点，每节点12个核，开启HT后48 processors 64G Mem	
9	192.168.1.115	左泽	Dell R730	Dell R720Intel Xeon E5-2650 CPU v4@2.2GHz，2NUMA节点，每节点12个核，开启HT后48 processors 64G Mem	
10	192.168.1.51	吴加禹	Dell R730	Intel Xeon E5-2650 CPU v4@2.2GHz，1NUMA节点，该节点共12个核，开启HT后24 processors 64G Mem	
11	192.168.1.121	汪睿(塔式机器)	Dell T620	Intel Xeon E5-2609 CPU v2@2.5GHz，1NUMA节点，该节点共4个核，好像不能开启HT 64G Mem	好像不能开启HT

1.2 软件信息

```
#1.操作系统版本
OS版本：Ubuntu 16.04.6 Server
内核版本:4.4.0-142
#2.kubernetes软件相关版本
kubeadm v1.18.3
kubectl v1.18.3
kubelet v1.18.3
kube-apiserver v1.18.3
kube-scheduler v1.18.3
kube-controller-manager v1.18.3
etcd 3.4.3
kube-prometheus v0.5+
```

2 Kubernetes安装过程

2.1 集群初始化设置

```
1. 在所有机器上生成ssh key，便于集群内各个机器间通过ssh无密码访问，命令如下，
ssh-keysgen
ssh-copy-id localhost
#将密钥拷贝到各个node，方便进行免密登录
for i in `seq 1 11`;do
    scp -r ~/.ssh k8s@node$i:.
done

2. 配置各个机器配置别名，直接通过别名访问，而不必输入IP
#在当前机器的/etc/hosts文件中追加如下内容，这样本机器访问其他机器直接通过名称而不是IP
192.168.1.107   node1 master
192.168.1.104   node2
192.168.1.118   node3
192.168.1.119   node4
192.168.1.114   node5
192.168.1.109   node6
192.168.1.134   node7
192.168.1.113   node8
192.168.1.115   node9
192.168.1.51    node10
192.168.1.121   node11

3. 配置sudo免密
#1.以root身份创建一个文件为k8s，位置为/etc/sudoers.d/k8s，内容如下：
k8s ALL=( root ) NOPASSWD:ALL
#然后拷贝一份到普通用户目录下并修改权限
cp /etc/sudoers.d/k8s ~
cd ~ && chown k8s:k8s k8s
#把k8s文件广播到其他节点，这样其他节点执行sudo时也无需输入密码
for i in `seq 2 11`;do
     scp k8s node$i:.
done

4. 给机器设置主机名，便于识别机器
#进入到其他各个节点，分别执行以下命令，从而给设置其他机器的主机名，以便知道当前是在那个node上
```

```
for i in `seq 2 11`;do
     ssh node$i sudo hostnamectl set-hostname node$i
done

5. 通过node名称来访问各个主机
#拷贝hosts文件
for i in `seq 2 11`;do
     scp /etc/hosts node$i:.
     ssh node$i sudo chown root:root hosts
     ssh node$i sudo mv ~/hosts /etc/hosts
done
6. 修改apt源
新的源的内容：https://developer.aliyun.com/mirror/ubuntu
for i in `seq 2 11`;do
     ssh node$i sudo cp /etc/apt/sources.list /etc/apt/sources.list_origin
     scp /etc/apt/sources.list node$i:.
     ssh node$i sudo chown root:root ~/sources.list
     ssh node$i sudo mv ~/sources.list /etc/apt/sources.list
done

7. 测试上面的效果
#测试sudo无需输入密码，可以运行指令sudo date
#测试ssh访问其他机器，可以运行指令ssh node3

8. 设置密钥登陆，不用密码登录，防止挖矿
#第1步：在ssh远程登录工具，如Mobaxetem中的Tools中SSH Key Generator生成密钥。
#然后追加公钥到服务器.ssh目录中的authorized_keys，具体可以参见3.4节。
#第2步：在服务器上关闭集群的密码访问方式，只允许通过密钥访问。
#第2.1步，执行sudo su命令
#第2.2步，修改/etc/ssh/sshd_config文件，将#PasswordAuthentication yes 中的#删去，同时
将yes改成no，并保存退出
#第2.3步，执行systemctl restart sshd
```

2.1 Kubernetes安装教程

2.1.1 准备阶段

注意准备阶段的操作是在集群的所有机器上都要执行的

```
#1.将机器上所有的kubernetes/docker软件都卸载掉，这里可以通过MobaXterm的MultiExec功能完
成，也可以像前面使用的for i in `seq 1 11`这种脚本
#MultiExec模式
sudo apt purge docker*
sudo apt purge kubeadm kubectl kubelet
sudo rm -rf /usr/bin/kube*
sudo apt autoremove
#shell脚本，下面类似
for i in `seq 1 11`;do
    ssh node$i sudo apt purge docker*
    ssh node$i sudo apt purge kubeadm kubectl kubelet
    ssh node$i sudo rm -rf /usr/bin/kube*
    ssh node$i sudo apt autoremove
done

#2.安装docker
curl -fsSL get.docker.com -o get-docker.sh
sudo sh get-docker.sh --mirror Aliyun
```

```
#3.启动docker ce
sudo systemctl enable docker
sudo systemctl start docker

#4.建立docker用户组
sudo groupadd docker
sudo usermod -aG docker $USER

#5.使用镜像加速，在/etc/docker/daemon.json中写入如下内容
sudo mkdir -p /etc/docker
sudo vim docker.json
#按下i键，将下面内容粘贴到vim打开的文件中
{
  "registry-mirrors": [
    "https://registry.docker-cn.com"
  ]
}

#6.重新启动服务
sudo systemctl daemon-reload
sudo systemctl restart docker

#7.禁用swap，禁用swap原因：https://github.com/kubernetes/kubernetes/issues/53533
sudo swapoff -a

#8.使用aliyun的源，注意从此之后的步骤最好使用sudo su切换到root模式，否则可能会出现各种问题
sudo su
apt-get update && apt-get install -y apt-transport-https curl
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main
EOF #注意从cat到EOF这是一个shell命令，这一步会在/etc/apt/source.list.d/目录下生成一个kubernetes的文件，文件内容就是deb https这一行。
apt-get update

#9.安装kubeadm/kubelet/kubectl
apt-cache policy #查看历史版本k8s，最新的kubenetes为1.18.4，但由于无法从阿里云和docker hub拉取kube-apiserver等镜像，所以这里安装k8s v1.18.3
apt-get install -y kubeadm=1.18.3-00 kubelet=1.18.3-00 kubectl=1.18.3-00 #可以选择其他k8s版本，还有就是kubernetes-cni可以先不安装，如果报错则执行apt-get install kubernetes-cni=0.7.5-00以便解决无法找到cni的问题
apt-mark hold kubelet kubeadm kubectl
#获取当前版本kubeadm需要启动的镜像
kubeadm config images list
exit #切回到普通用户
#结果类似与下面的结果
#k8s.gcr.io/kube-apiserver:v1.18.3
#k8s.gcr.io/kube-controller-manager:v1.18.3
#k8s.gcr.io/kube-scheduler:v1.18.3
#k8s.gcr.io/kube-proxy:v1.18.3
#k8s.gcr.io/pause:3.2
#k8s.gcr.io/etcd:3.4.3-0
#k8s.gcr.io/coredns:1.6.7

#10.编写下载镜像的脚本install-k8s.sh，提前来取好镜像脚本内容如下：
#----------------------------install-k8s.sh开始-----------------
#! /bin/bash
images=(
```

```
    kube-apiserver:v1.18.3
    kube-controller-manager:v1.18.3
    kube-scheduler:v1.18.3
    kube-proxy:v1.18.3
    pause:3.2
    etcd:3.4.3
    coredns:1.6.7
)

for imageName in ${images[@]} ; do
    docker pull registry.cn-hangzhou.aliyuncs.com/google_container
        s/$imageName
    docker tag registry.cn-hangzhou.aliyuncs.com/google_containers
        /$imageName k8s.gcr.io/$imageName
done
#----------------------------install-k8s.sh结束------------------

#11.执行install-k8s.sh脚本
chmod +x install-k8s.sh
sudo ./install-k8s.sh
#注意上面的脚本执行完成后，可能需要将etcd的镜像重新打一下tag，执行如下指令
sudo docker tag k8s.gcr.io/etcd:3.4.3 k8s.gcr.io/etcd:3.4.3-0

#12.下载prometheus相关容器镜像，为后期搭建prometheus做准备
sudo docker pull quay.io/coreos/kube-state-metrics:v1.9.5
sudo docker pull quay.io/coreos/kube-rbac-proxy:v0.4.1
sudo docker pull quay.io/prometheus/alertmanager:v0.20.0
sudo docker pull quay.io/coreos/k8s-prometheus-adapter-amd64:v0.5.0
sudo docker pull quay.io/prometheus/node-exporter:v0.18.1
sudo docker pull quay.io/prometheus/prometheus:v2.15.2
sudo docker pull quay.io/coreos/prometheus-operator:v0.38.1
sudo docker pull grafana/grafana:6.6.0
#注意prometheus相关的镜像名称可以从kube-prometheus的manifests和manifests/setup文件夹中
找到
```

2.1.2 安装Master节点

```
#1.配置cgroup dirver
sudo docker info | grep -i cgroup
#上面的指令执行结果为： Cgroup Driver: cgroupfs
#因为kubelet使用的cgroupfs为system，和上述不一致，所以需要有如下修正：
sudo  vim /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
#然后再vim打开的文件中添加如下配置：
Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs"
#或者下面的内容也可以： Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs --
pod-infra-container-image=registry.cn-
hangzhou.aliyuncs.com/google_containers/pause-amd64:3.1"

#2.重启kubelet，实际上此时kubelet启动是失败的
sudo systemctl daemon-reload
sudo systemctl restart kubelet

#3.解除防火墙限制
sudo vim /etc/sysctl.conf
#在打开的文件末尾添加如下内容：
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.bridge.bridge-nf-call-iptables = 1
#添加完上面的内容后，执行如下指令：
sudo sysctl -p

#4.初始化kubeadm
sudo kubeadm init --kubernetes-version=1.18.3 --apiserver-advertise-
address=192.168.1.107 --pod-network-cidr=10.244.0.0/16 --service-
cidr=10.96.0.0/12

#注意:kubeadm init常用主要参数如下：
#--kubernetes-version: 指定Kubenetes版本，如果不指定该参数，会从google网站下载最新的版本信息。
#--pod-network-cidr: 指定pod网络的IP地址范围，它的值取决于你在下一步选择的哪个网络网络插件，如果使用的是Calico网络，需要指定为192.168.0.0/16，如果使用的是Flannel则需要指定为10.244.0.0/16。其他CNI网络教程请参看：https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

#--apiserver-advertise-address: 指定master服务发布的Ip地址，如果不指定，则会自动检测网络接口，通常是内网IP。
#--feature-gates=CoreDNS: 是否使用CoreDNS，值为true/false，CoreDNS插件在1.10中提升到了Beta阶段，最终会成为Kubernetes的缺省选项。

#5.kubeadm init指令执行成功后会显示出下面需要执行的三条指令，指令如下，
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
#注意kubeadm init指令执行完后也会出现kubeadm join 192.168.1.107:6443 --token
tq9yri.dqod9td3fhiatq02  --discovery-token-ca-cert-hash
sha256:4616b79191ff1161ea1b12d35645a0534fd485f9718488cbf9774cd55dcd85ea  这一句话，要记住这句话，后面将其他节点加入集群的时候就直接在要加入k8s集群的slave节点中执行这句话就可以了，执行成功后会看到This node has joined the cluster的结果

#6.执行完第5步的三条指令后，进行测试，执行下面2条指令中的一条即可
curl https://127.0.0.1:6443 -k 或者 curl https://<master-ip>:6443 -k
#执行了上述指令后得到的回应如下：
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}

#7.查看节点状态
kubectl get nodes #会发现nodes处于NotReady状态，这是正确的结果，等安装完成CNI网络插件后状态才会变为Ready。


#7.安装网络插件，此处以flannel为例，参考：
https://www.cnblogs.com/winstom/p/11159165.html,
https://github.com/coreos/flannel
```

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

#8.查看pod启动状况
kubectl get pods --all-namespaces #如果看到pod都处于running状态，则可以添加slave节点
kube-system   calico-kube-controllers-76d4774d89-5dk5q   1/1     Running   0
     41h
kube-system   calico-node-2n6pd                          1/1     Running   0
     40h
kube-system   calico-node-hm6hh                          1/1     Running   0
     41h
kube-system   calico-node-x46md                          1/1     Running   0
     40h
kube-system   coredns-66bff467f8-278w7                   1/1     Running   0
     42h
kube-system   coredns-66bff467f8-7w5mp                   1/1     Running   0
     42h
kube-system   etcd-node1                                 1/1     Running   0
     42h
kube-system   kube-apiserver-node1                       1/1     Running   0
     42h
kube-system   kube-controller-manager-node1              1/1     Running   0
     42h
kube-system   kube-proxy-2wff5                           1/1     Running   0
     40h
kube-system   kube-proxy-9z8rt                           1/1     Running   0
     40h
kube-system   kube-proxy-mnjlr                           1/1     Running   0
     42h
kube-system   kube-scheduler-node1                       1/1     Running   0
     42h
```

2.1.3 加入Slave节点

注意：默认情况下，集群不会将Pod调度到Master节点，如果想调度Pod到Master节点，需要执行指令：`kubectl taint nodes --all node-role.kubernetes.io/master-`，输出的结果类似与下面：

```
node "test-01" untainted
taint "node-role.kubernetes.io/master:" not found
taint "node-role.kubernetes.io/master:" not found
```

在想要作为集群Slave节点的机器中执行如下指令：

```
#注意：在Master节点中执行kubeadm init指令时实际上就给出了加入k8s集群的完整指令，如果slave是
在一开始就加入集群的话，直接执行kubeadm init给出的指令即可。而如果超过了24个小时，才将Slave加
入集群的话，可以参考下面的方式。

sudo su

kubeadm join --token <token> <control-plane-host>:<control-plane-port> --
discovery-token-ca-cert-hash sha256:<hash>


#kubeadm join指令相关说明：
#1.<token>的值可以通过在Master节点中执行下面的指令获取：
```

```
kubeadm token list
#2.输出的结果类似于下面：
TOKEN                    TTL  EXPIRES              USAGES           DESCRIPTION
           EXTRA GROUPS
8ewj1p.9r9hcjoqgajrj4gi  23h  2018-06-12T02:51:28Z authentication,  The default
bootstrap  system:
                                                   signing          token
generated by     bootstrappers:
                                                                    'kubeadm
init'.        kubeadm:

          default-node-token                    #3.注意：token的有效期为24小时，过了
24小时在使用kubeadm token list就看不到结果，此时如果想将新的机器加入到Kubernetes集群中，需
要重新创建token，创建token的指令如下：
kubeadm token create  #输出的结果类似于5didvk.d09sbcov8ph2amjw这种
#4.<control-plane-host>:<control-plane-port>指的是Master的IP地址和API server的端口，
如：192.168.1.107:6443，kubeadm方式安装时通常是6443端口
#5.<hash>的值可以通过在Master中运行如下指令获取，输出的结果类似与下面：
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -
outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'

#6.在Slave节点中执行完成上述指令后，等待一段时间，在Master中执行kubectl get nodes就可以看
到slave节点处于Ready状态了
```

2.1.4 移除Slave节点

```
#1.在Master节点中执行下面的指令：
kubectl get nodes #获取Kubernetes集群所有的node列表
kubectl drain <node name> --delete-local-data --force --ignore-daemonsets
kubectl delete node <node name>
#2.在被删除的Slave节点上，执行如下指令重置所有kubeadm的安装状态
sudo kubeadm reset
```

2.1.5 移除Master节点

```
#1.在Master节点中执行如下指令
kubectl get nodes #获取Kubernetes集群所有的node列表
kubectl drain <master node name> --delete-local-data --force --ignore-daemonsets
kubectl delete node <master node name>
sudo kubeadm reset
```

2.1.6 验证集群功能

```
#1.检查节点状态
kubectl get nodes #正常情况下所有节点都处于Ready状态
#2.创建测试文件 nginx-ds.yml，其内容如下：
apiVersion: v1
kind: Service
metadata:
  name: nginx-ds
  labels:
    app: nginx-ds
spec:
  type: NodePort
```

```
  selector:
    app: nginx-ds
  ports:
  - name: http
    port: 80
    targetPort: 80
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-ds
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      app: nginx-ds
  template:
    metadata:
      labels:
        app: nginx-ds
    spec:
      containers:
      - name: my-nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80

#3.执行测试
kubectl create -f nginx-ds.yml #结果应该是每一个Slave节点都有一个nginx-ds的Pod在运行
#4.检查各节点的Pod IP连通性，编写测试脚本ping-test.sh，脚本内容如下：
#!/bin/bash
podIPs=`kubectl get pods -o wide -l app=nginx-ds | awk '{print  $6}' |grep -v
IP`
count=0
for podIp in ${podIPs[@]}
do
  let count=0
  for i in `seq 1 11` ;do
    if ssh node$i  ping -c 1 $podIp |grep '100% packet loss' ;
    then
           echo "node$i ping $podIp error" ;
    else
           let count+=1
    fi
  done
  if [ $count == 11 ];then
    echo "all Slave node ping $podIp successful!";
  else
    echo "ping $podIp failed"
  fi
done
#5.执行脚本
chmod +x ping-test.sh
./ping-test.sh
#6.如果ping-test.sh成功执行则测试服务IP和端口可达性。如果ping-test.sh执行后发现部分部分
ping测试失败，则可以再执行一次，第一次可能会因为需要做一些准备工作而失败，再次执行后一般会成功。
#6.1测试服务IP可达性，编写脚本svc-test.sh，其内容如下：
```

```
#!/bin/bash
clusterIP=`kubectl get svc -l app=nginx-ds | awk '{print  $3}' |grep -v IP`
echo "clusterIP=$clusterIP"
for i in `seq 1 11` ; do
  ssh node$i curl -s $clusterIP | grep "<title>Welcome to nginx"
done
#7.执行脚本svc-test.sh
chmod +x svc-test.sh
./svc-test.sh #如果输出了11个<title>Welcome to nginx!</title>则表示服务IP是可达的
#8.测试端口可达性，编写脚本port-test.sh，其内容如下：
#!/bin/bash
count=0
nodePort=`kubectl get svc -l app=nginx-ds | awk '{print  $5}' |grep -v PORT |
awk -F ':' '{print $2}' |awk -F '/' '{print $1}'`
for i in `seq 1 11`; do
  ssh node$i curl -s node$i:$nodePort | grep "<title>Welcome to nginx"
  let count+=1
done
echo "total count=$count"

#9.执行port-test.sh
chmod +x port-test.sh
./port-test.sh #如果输出了11个<title>Welcome to nginx!</title>以及最后输出的total
count=11则表示NodePort也是可达的

#10.至此，说明集群搭建成功
```

相关参考：

1. 使用kubeadm 部署 Kubernetes(国内环境)
2. 在Ubuntu 18.04 中安装 Kubernetes
3. Creating a single control-plane cluster with kubeadm
4. kubeadm安装kubernetes
5. kubeadm初始化Kubernetes集群
6. Kubernetes-1.18.0高可用集群部署
7. 使用kubeadm安装kubernetes_v1.18.x

2.1.6 其它搭建Kubernetes集群方式

- 和我一步步部署 KUBERNETES 集群
- Kubernetes ubuntu二进制包安装
- k8s1.12优化 二进制安装（非常方便维护）
- kubernetes1.12二进制离线安装总结

3 部署kube-prometheus

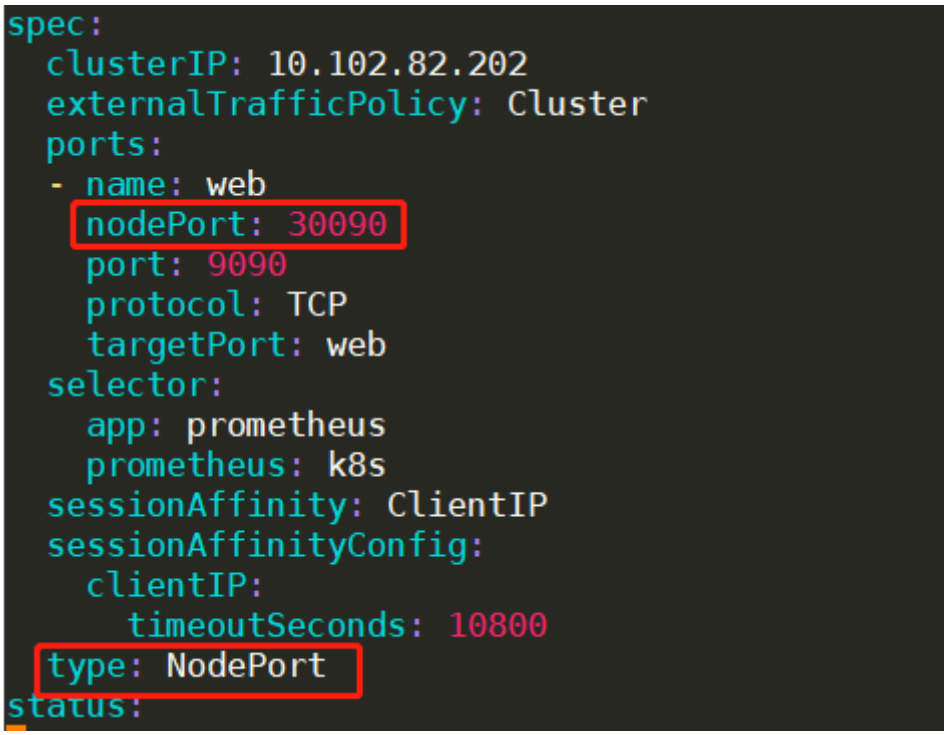
3.1 部署kube-prometheus

```
#1.下载kube-prometheus
git clone https://github.com/coreos/kube-prometheus.git
#2.安装kube-prometheus
kubectl create -f manifests/setup
kubectl create -f manifests/
#注意此时执行kubectl get servicemonitors --all-namespaces 会得到No Resources found
的结果
```

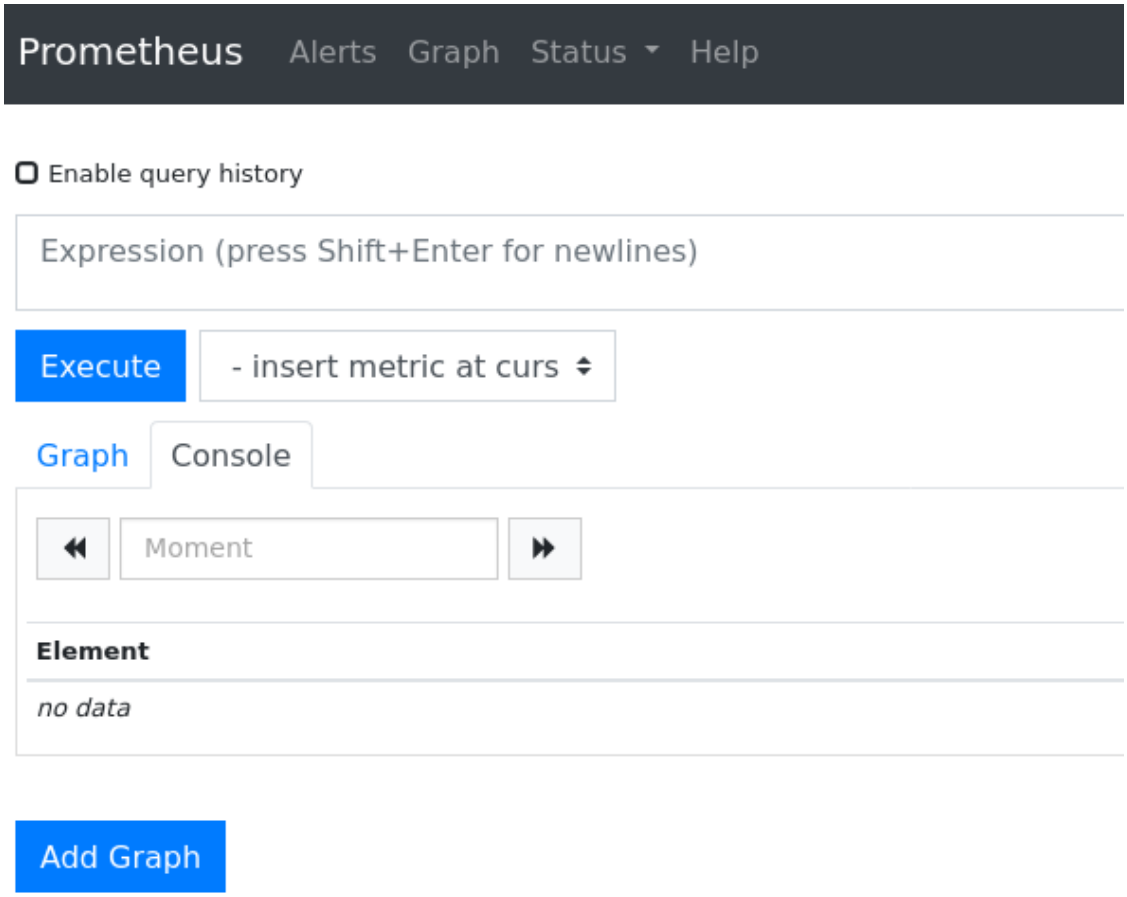
```
#3.一直等待知道出现下面的结果，即monitoring namespace下所有的Pod都处于running状态，则表示prometheus可用，使用kubectl get pod -n monitoring命令查看
NAME                                   READY   STATUS    RESTARTS   AGE
alertmanager-main-0                    2/2     Running   0          55s
alertmanager-main-1                    2/2     Running   0          55s
alertmanager-main-2                    2/2     Running   0          55s
grafana-5c55845445-nxvw8               1/1     Running   0          53s
kube-state-metrics-957fd6c75-8vm26     3/3     Running   0          53s
node-exporter-58bww                    2/2     Running   0          53s
node-exporter-65tqk                    2/2     Running   0          53s
node-exporter-7v7m5                    2/2     Running   0          53s
node-exporter-7w8mn                    2/2     Running   0          53s
node-exporter-9gqtd                    2/2     Running   0          53s
node-exporter-9sc28                    2/2     Running   0          53s
node-exporter-jj95q                    2/2     Running   0          53s
node-exporter-sjqhq                    2/2     Running   0          53s
node-exporter-x8bhx                    2/2     Running   0          53s
node-exporter-xm8pq                    2/2     Running   0          53s
prometheus-adapter-fb65f8f5f-4wgrs     1/1     Running   0          53s
prometheus-k8s-0                       3/3     Running   1          54s
prometheus-k8s-1                       3/3     Running   1          54s
prometheus-operator-5b96bb5d85-s5kc8   2/2     Running   0          62s

#4.如果出现上面的结果则表示prometheus安装成功，下面更改prometheus的访问模式，以便从集群外访问prometheus
## 4.1 修改

#4.请参考博客：https://blog.csdn.net/Happy_Sunshine_Boy/article/details/105708921
```

3.2 更改Prometheus访问模式

- 修改配置文件`kubectl edit svc/prometheus-k8s -n monitoring`，修改该文件中的nodePort和type，如下图：



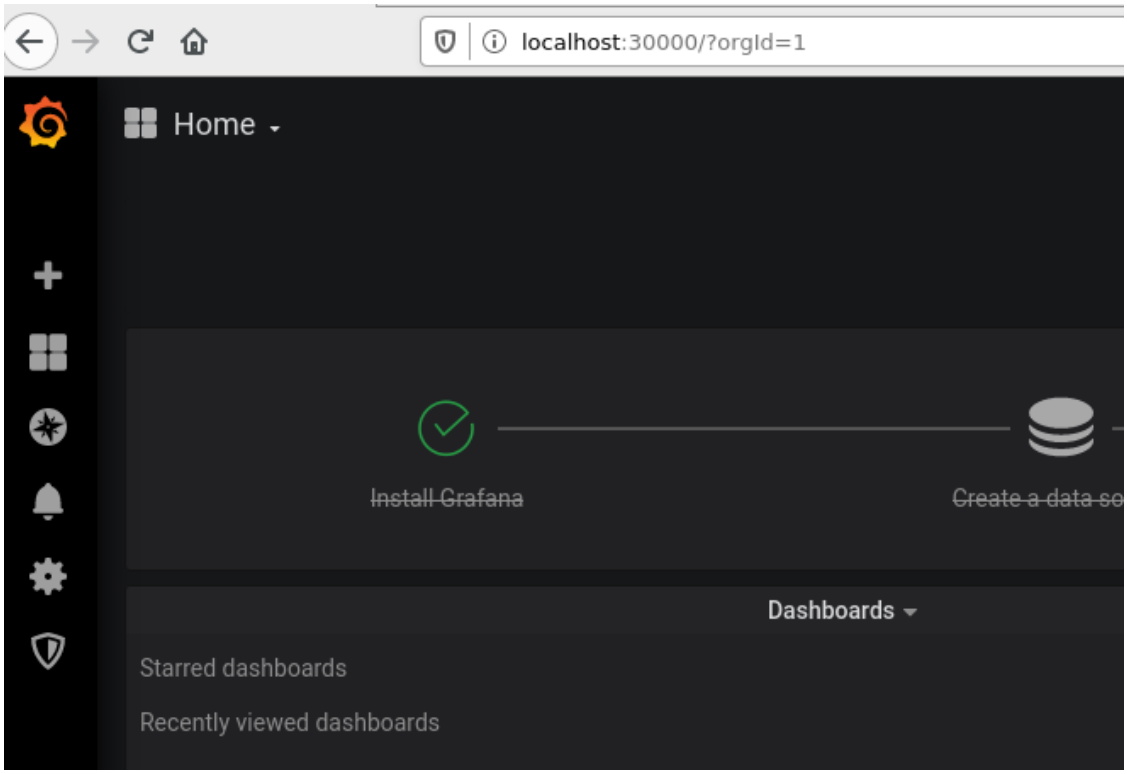
注意：nodePort是新增的，端口号自己指定，type改为NodePort，这样通过机器的外网IP+30090端口就可以访问prometheus了，访问结果类似与下面：



- 修改配置文件`kubectl edit svc/grafana -n monitoring`，修改该文件中的nodePort和type，如下图：

```
spec:
  clusterIP: 10.110.136.177
  externalTrafficPolicy: Cluster
  ports:
  - name: http
    nodePort: 30000
    port: 3000
    protocol: TCP
    targetPort: http
  selector:
    app: grafana
  sessionAffinity: None
  type: NodePort
status:
```

访问结果类似与下面，注意：Granafa的登录账户和密码都是admin



- 修改配置文件`kubectl edit svc/alertmanager-main -n monitoring`，修改该文件中的nodePort和type，过程与上面类似。

3.3 Prometheus问题解决

通过IP+30090端口或者localhost+30090端口，即上面prometheus修改的nodePort，查看prometheus结果中，发现monitoring/kube-controller-manager和monitoring/kube-scheduler没有对应的监控目标，即它们都是0/0up。解决方案如下：

- 首先进入下载的kube-prometheus目录，`cd ~/k8s_install_files/kube-prometheus/manifests`，然后执行`vim prometheus-kubeControllerManagerService.yaml`，注意这里实际上是新增了该文件，新增内容如下：

```
apiVersion: v1
kind: Service
metadata:
  namespace: kube-system
  name: kube-controller-manager
  labels:
    k8s-app: kube-controller-manager
spec:
  selector:
    component: kube-controller-manager
  type: ClusterIP
  clusterIP: None
  ports:
  - name: http-metrics
    port: 10252
    targetPort: 10252
    protocol: TCP
---
apiVersion: v1
kind: Endpoints
metadata:
  labels:
```

```
    k8s-app: kube-controller-manager
  name: kube-controller-manager
  namespace: kube-system
subsets:
- addresses:
  - ip: 192.168.1.107
  ports:
  - name: http-metrics
    port: 10252
    protocol: TCP
```

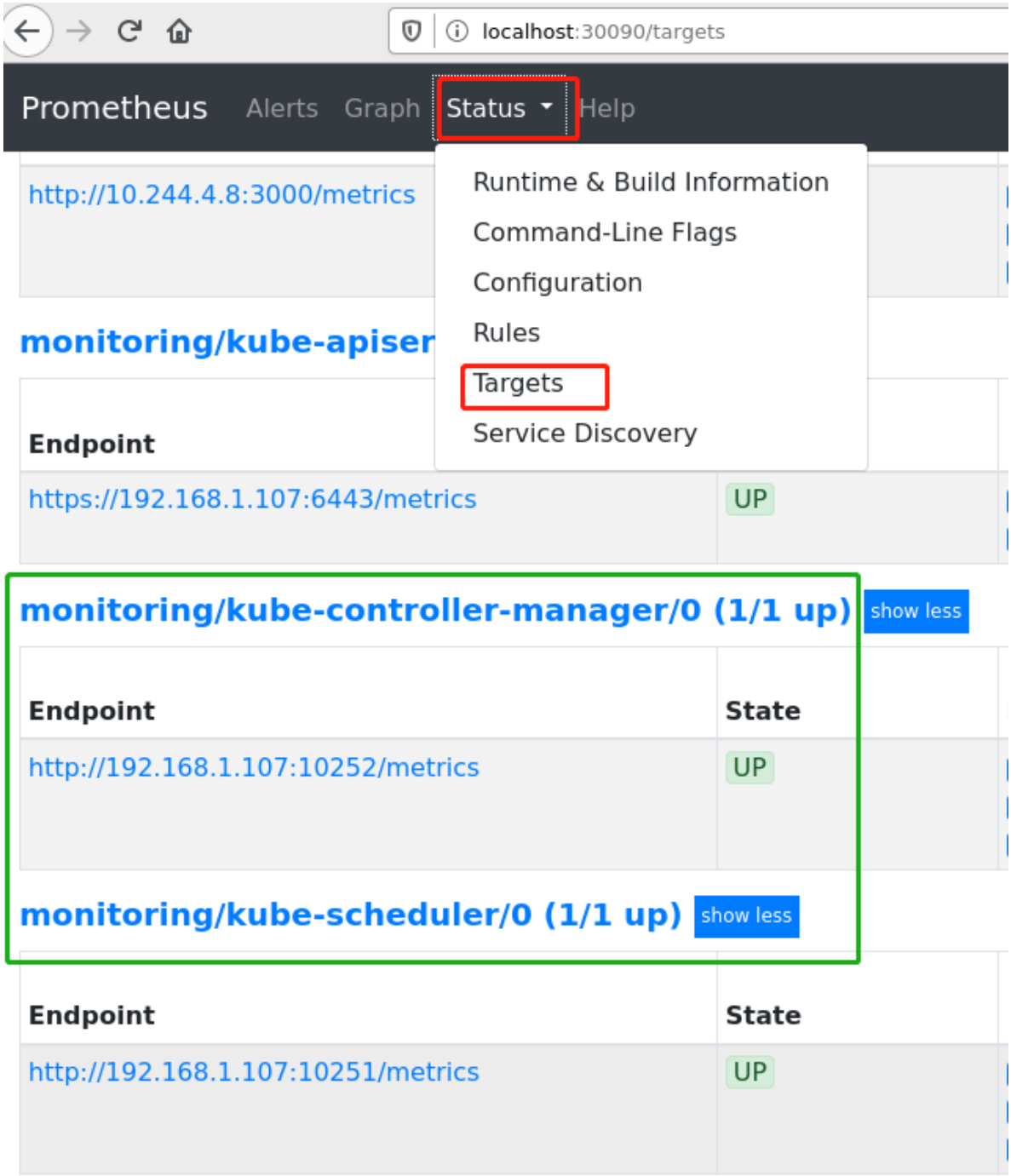
注意：上述IP指的是Master所在的IP，也可以在status->Targets中看monitoring/kube-apiserver的Labels列中的instance中的IP

- 执行`kubectl apply -f prometheus-kubeControllerManagerService.yaml`
- 继续在上述目录下添加另一个文件，使用指令`vim prometheus-kubeSchedulerService.yaml`，该文件内容如下：

```
apiVersion: v1
kind: Service
metadata:
  namespace: kube-system
  name: kube-scheduler
  labels:
    k8s-app: kube-scheduler
spec:
  type: ClusterIP
  clusterIP: None
  ports:
  - name: port
    port: 10251
    protocol: TCP
---
apiVersion: v1
kind: Endpoints
metadata:
  labels:
    k8s-app: kube-scheduler
  name: kube-scheduler
  namespace: kube-system
subsets:
- addresses:
  - ip: 192.168.1.107
  ports:
  - name: http-metrics
    port: 10251
    protocol: TCP
```

注意IP

- 接着执行指令`kubectl apply -f prometheus-kubeSchedulerService.yaml`。最后的结果如下：



3.4 删除kube-prometheus

只需执行下面的指令：

```
kubectl delete --ignore-not-found=true -f manifests/ -f manifests/setup
```

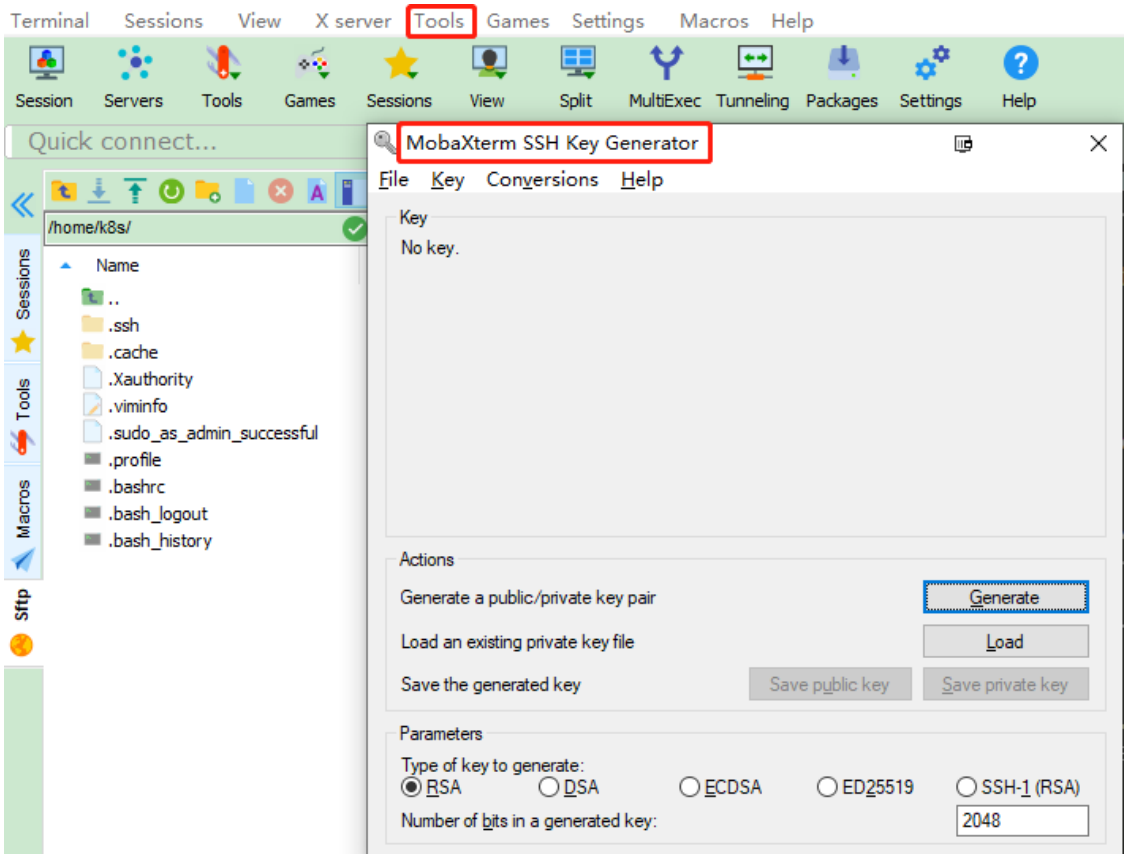
相关参考：

- Kubernetes-v1.17.4部署kube-prometheus-v0.5.0
- kube-prometheus github网址

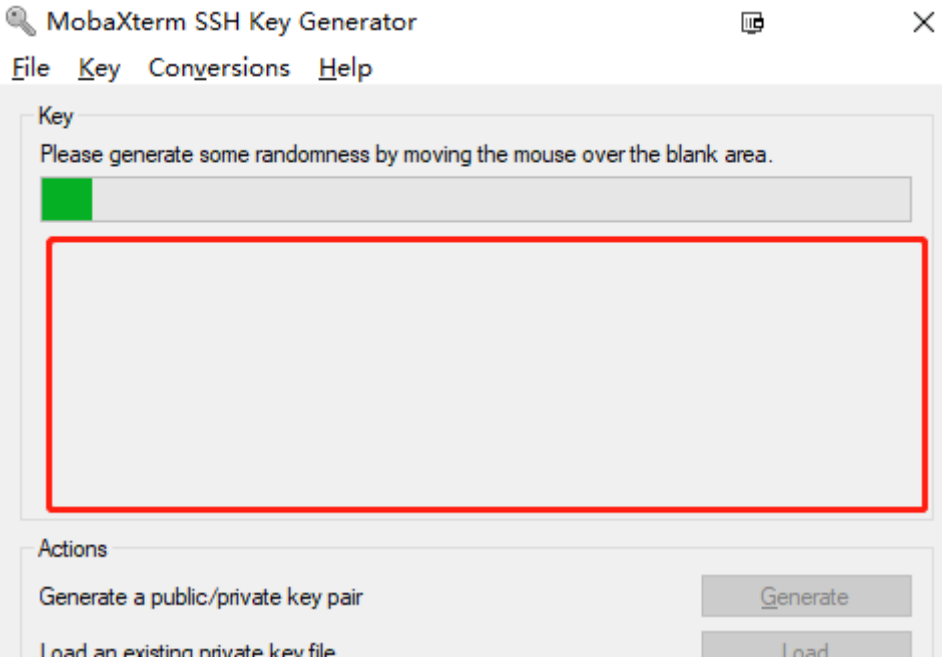
4 使用SSH密钥方式登录Kubernetes集群

- 此处以MobaXterm远程登录工具为例：

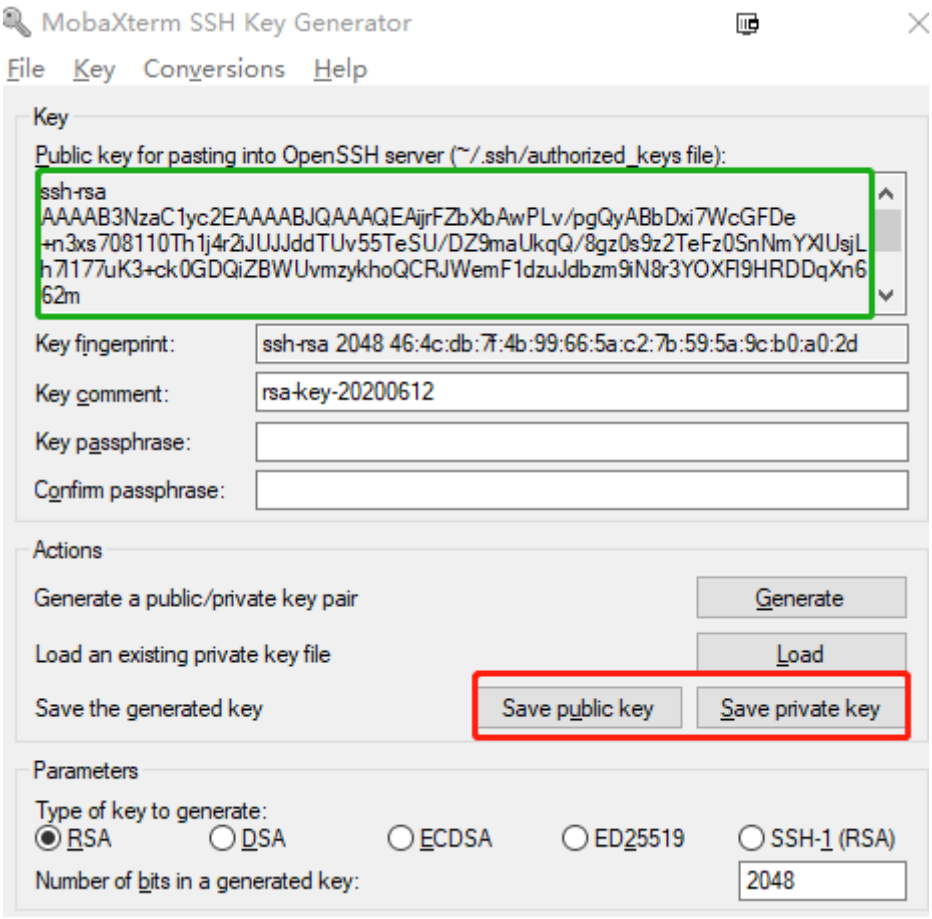
1. 点击MobaXterm的工具栏Tools下的MobaKeyGen(SSH key generator)，点开后界面如下：



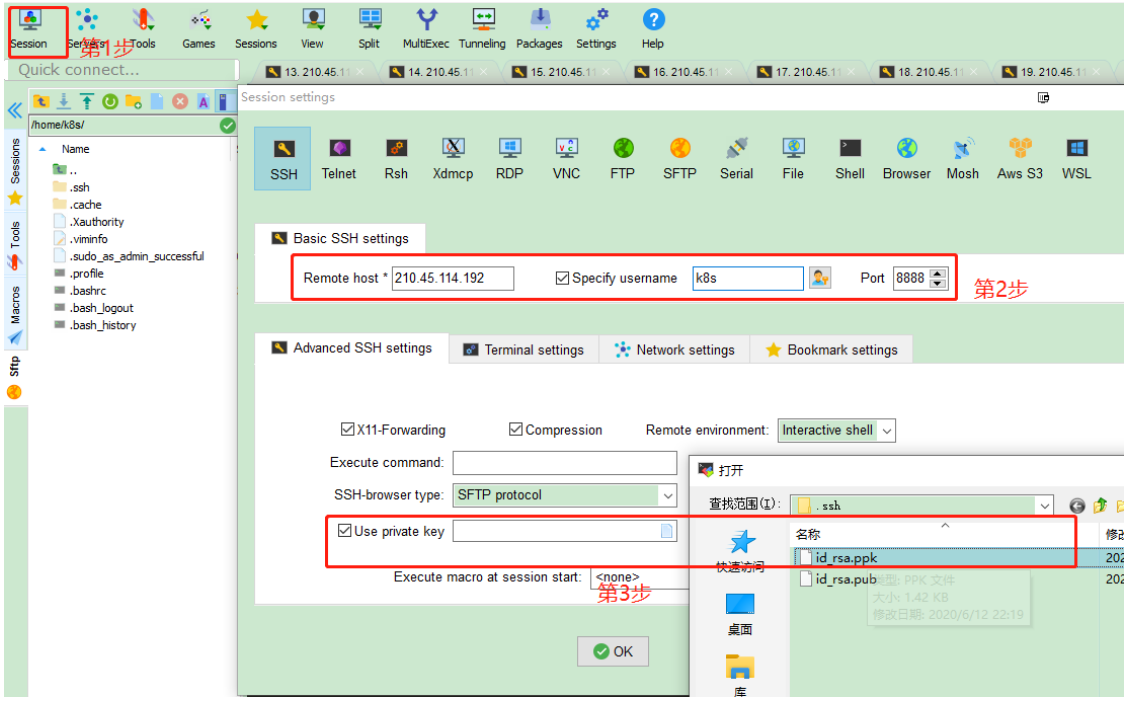
2. 点击Generate按钮，然后会开始生成公钥/私钥，**注意在生成的过程中需要在下图红框中用鼠标滑动**，否则生成公钥/私钥进度条不会动。



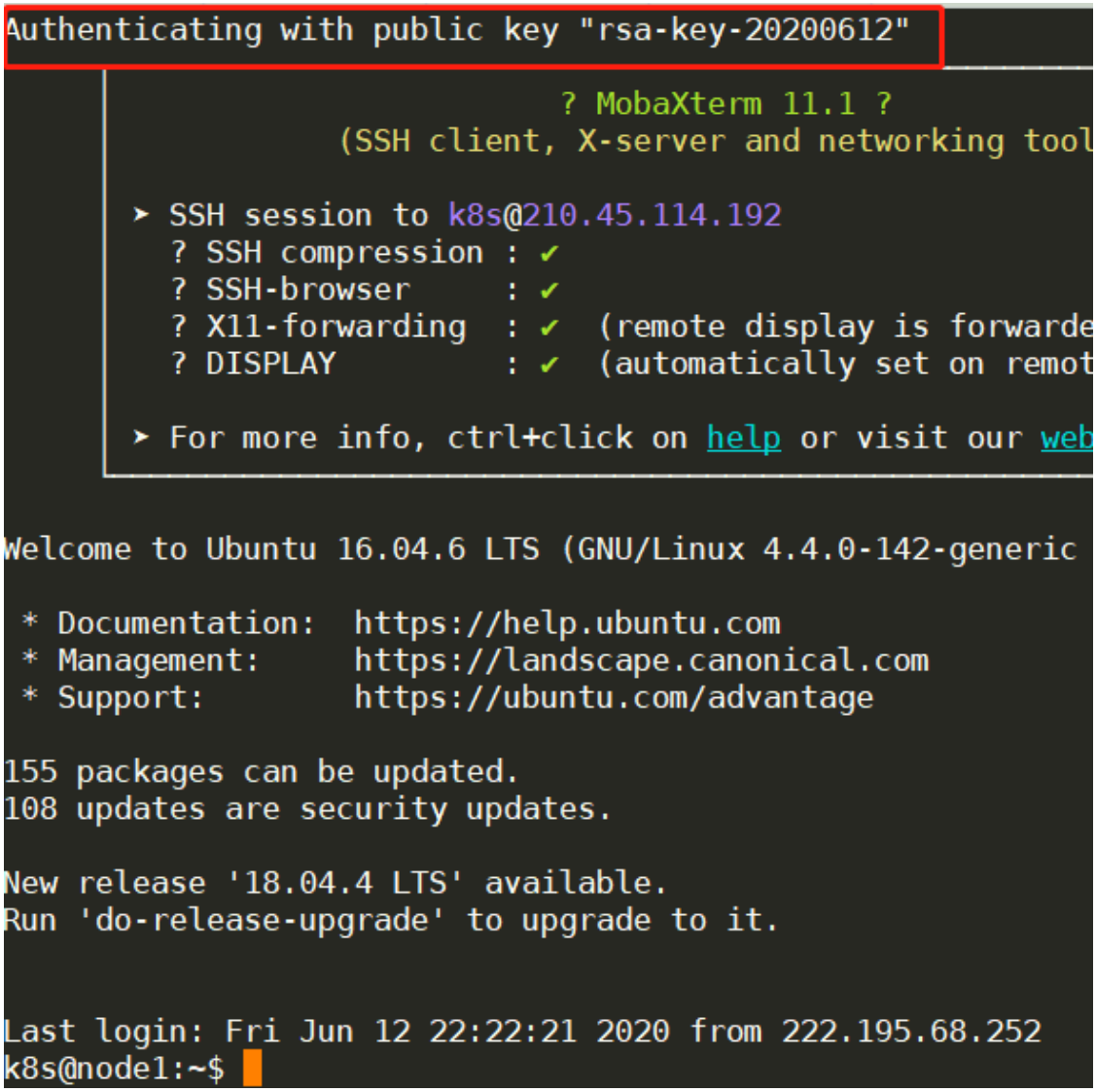
3. 生成完毕后的效果如下图。**首先**点击红框中的按钮，保存生成的公钥和私钥，比如可以将公钥（public key）保存成id_rsa.pub文件，保存的目录自己选择。私钥可以保存成id_rsa.ppk文件。**其次**，将下图绿色框线中的所有数据（可能需要拖动右侧滑动栏），拷贝并发给集群管理员，等待管理员的回复。



4. 收到管理员回复后，才可以在MobaXterm中登录Kuberntes集群。打开MobaXterm，登录的过程如下：



5. 登录后效果如下。

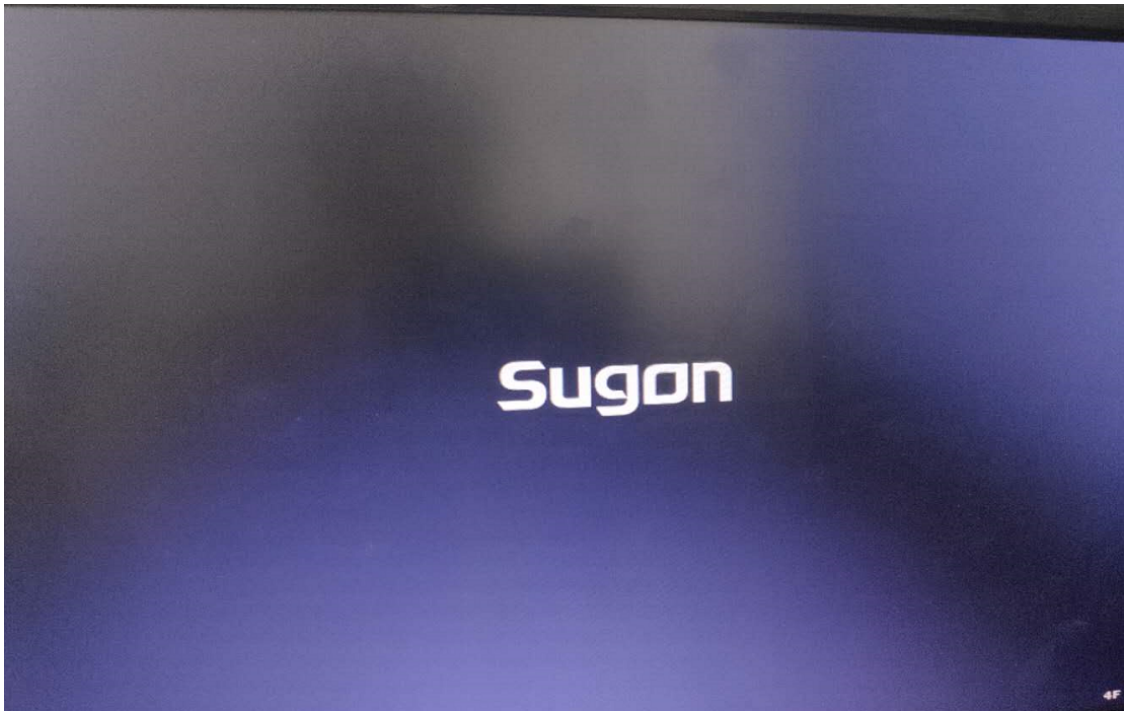


注：在Kubernetes集群内部，一个node进入到其他node可以直接使用ssh node编号(1-11，1号为master)，如在node1中进入node2可以使用`ssh node2`

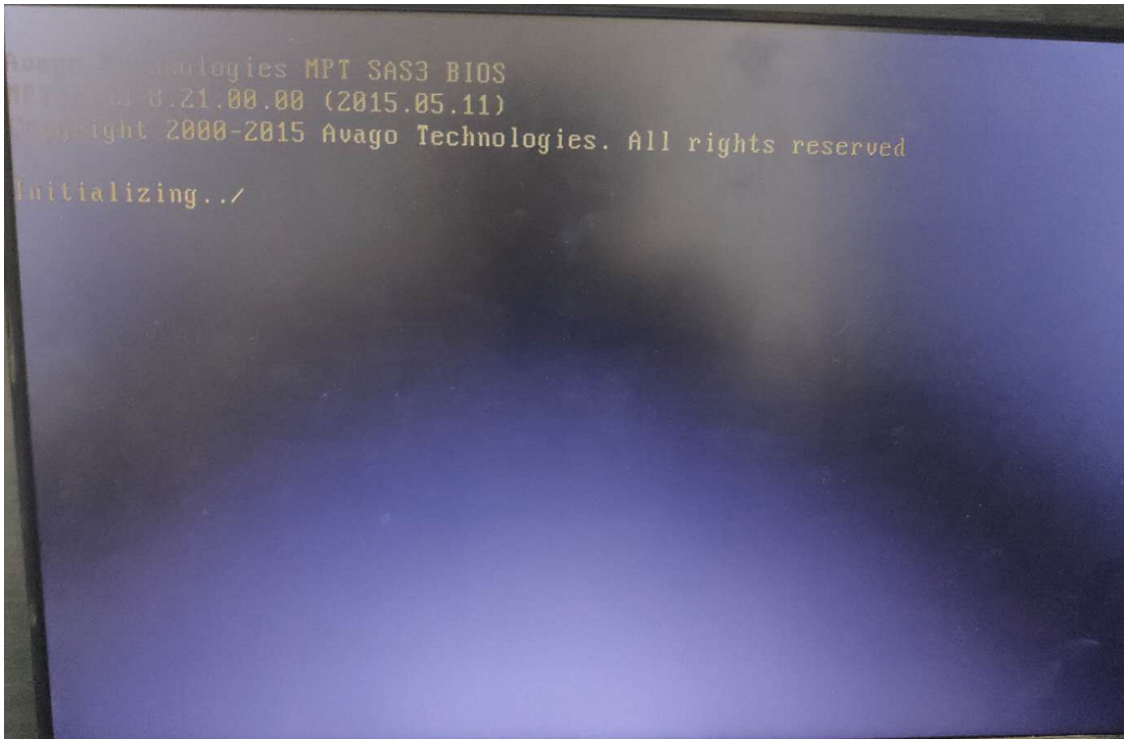
5 相关问题及解决方案

5.1 如何更换曙光I620-G20机器启动盘

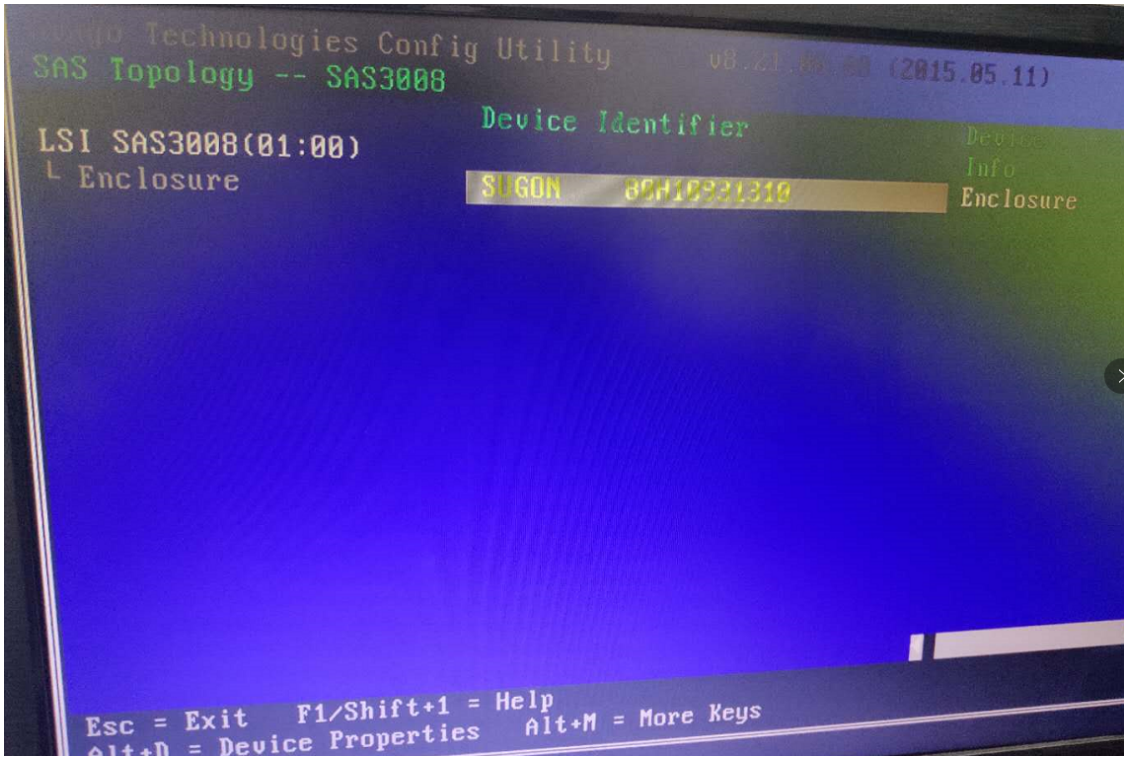
- 开机时首先进入Logo界面，如下：



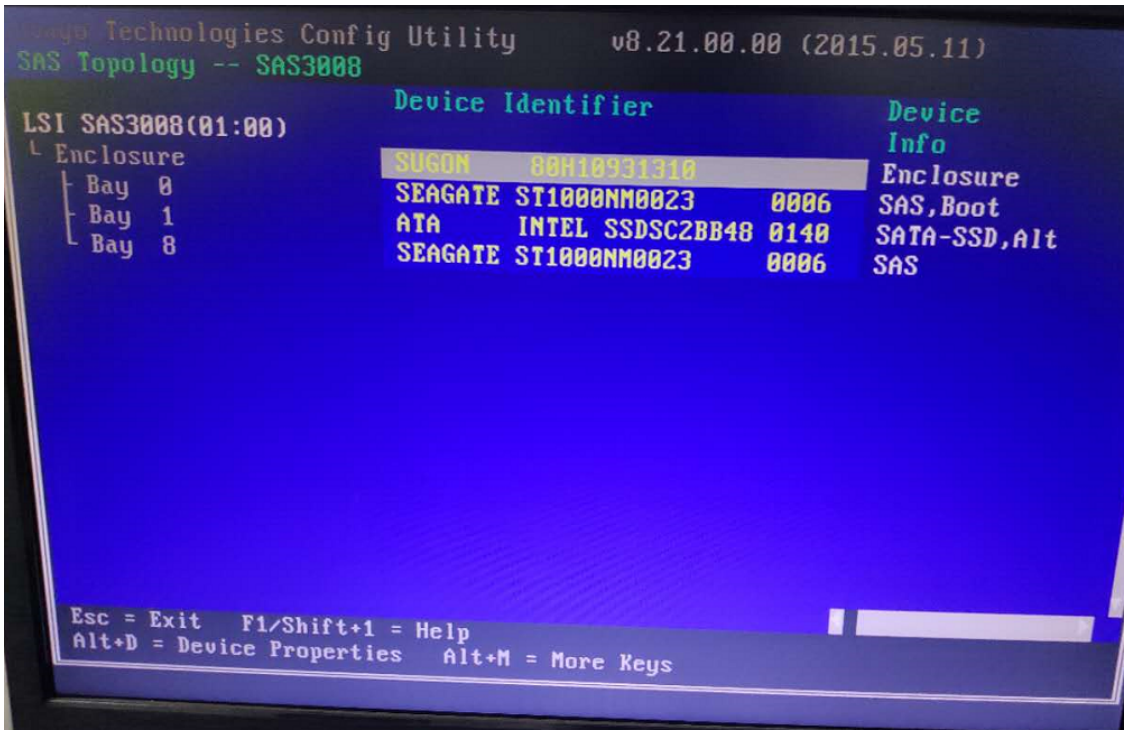
- 然后进入如下界面，并在此界面下按`ctrl + c`键。



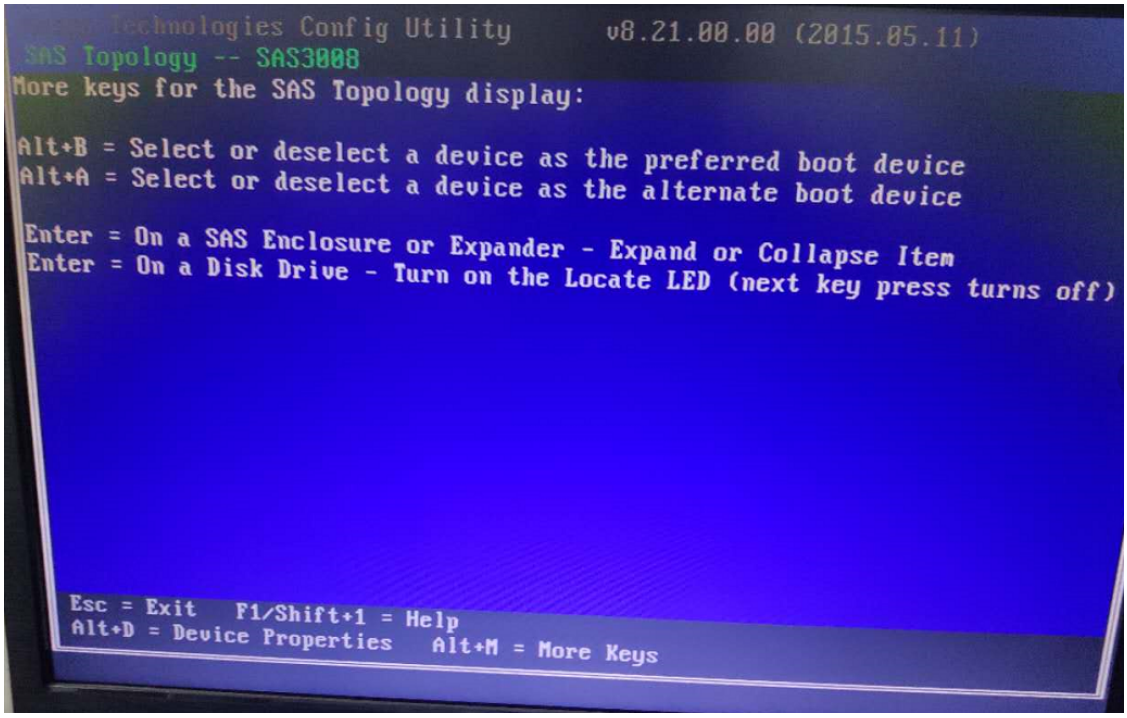
- 之后进入如下界面。



- 在上图界面中按`enter`键，并且需要多次，一直到如下界面。



- 上图中的Bay其实就是磁盘在机器上的插槽编号，选择自己想作为启动盘的条目，然后按`Alt + B`选择该条目作为启动项，也可以按`Alt + A`选择该条目作为可选的启动项，具体操作可以参考如下界面的说明。



- 选择好启动盘后，保存并退出。之后机器重新启动，然后之后在按`Del`键进入BIOS设置相应的磁盘作为首选引导盘。

  Dell机器也有类似的操作，不过需要在开机的时候按`ctrl + r`键进行设置，之后通过`F11`进入BIOS设置。

5.2 安装Ubuntu16.04 Server教程、

参考博客

5.3 Ubuntu 16.04 Server安装时网络自动化配置失败

Ubuntu16.04.6 Server版安装的过程中网络自动化配置失败，需要在安装成功后进行手动配置。配置步骤如下：

- 使用`ifconfig -a`打印出所有网卡的信息
- 使用`sudo dhclient + 网卡名`自动获取IP地址，执行成功后使用`ping`进行确认

  参考博客

5.4 kubectl启动失败

此时可以通过`journalctl -xefu kubectl`来查看相关的日志，可能的一个原因是swap没有关闭，可以使用`sudo swapoff -a`来关闭swap并使用`sudo systemctl restart kubelet`重启kubelet。相关参考

5.5 部分组件启动失败

- 在kubernetes安装的过程中，如果发现部分组件启动失败，可以使用`journalctl -xefu 组件名称`来查看组件最新启动的log日志，查看其出错原因。
- 在创建完成某些Deployment/ReplicaSet/ReplicationController/DaemonSet后，可以通过使用`kubectl get pods`或者`kubectl get pods -n Pod所在namespace名称`或者使用`--all-namespaces`来查看Pod的启动结果，如果发现Pod长时间停留在某一状态，则可以使用`kubectl describe pods pod名称 -n Pod所在namespace名称`来查看Pod未成功启动原因。或者使用`kubectl logs pod名称`

5.6 kubernetes网络报错相关问题

5.6.1 Pod启动异常问题

使用`kubectl get pods -n kube-system`发现coredns移除处于ContainerCreating状态。

- 查看Pod的详细情况`kubectl describe pod pod名称 -n kube-system`
- 如果看到了下面的信息：

```
Error syncing pod
Pod sandbox changed, it will be killed and re-created.
```

- 可以发现，该 Pod 的 Sandbox 容器无法正常启动，具体原因可以通过`journalctl -xefu kubelet`查看 Kubelet 日志，如果看到了类似与下面的问题：

```
RunPodSandbox from runtime service failed: rpc error: code = 2 desc =
NetworkPlugin cni failed to set up pod "kube-dns-86f4d74b45-ffwjf" network:
failed to set bridge addr: "cni0" already has an IP address different from
10.244.4.1/24
```

  这里的一个Pod中启动了多个容器，所以，使用kubectl logs 命令查看日志很有局限性，关于kubectl logs的使用，请参考kubernetes中的Pod简述与实践和kubernetes中文文档。

- 解决方案
  - 可以现在Master节点上将出问题Pod所在的Slave节点移除，移除方式可以参考2.1.4小节。然后在Pod被调度的节点上执行下面的操作：

```
sudo kubeadm reset
sudo systemctl stop kubelet
sudo systemctl stop docker
sudo rm -rf /var/lib/cni/
sudo rm -rf /var/lib/kubelet/*
sudo rm -rf /etc/cni/
sudo ifconfig cni0 down
sudo ifconfig flannel.1 down
sudo ifconfig docker0 down
sudo ip link delete cni0
sudo ip link delete flannel.1
##重启kubelet
sudo systemctl restart kubelet
##重启docker
sudo systemctl restart docker
```

  - 之后再次将给slave节点加入到集群中，加入方式参考2.1.3小节。如果上面的操作之后如果还是报类似的问题或者下面的错误：

```
"CreatePodSandbox for pod \" kube-dns-86f4d74b45-ffwjf
_default(78e796f5-e
b7c-11e7-b903-b827ebd42d30)\" failed: rpc error: code = Unknown desc =
N
etworkPlugin cni failed to set up pod \" kube-dns-86f4d74b45-ffwjf
_default\"
network: failed to allocate for range 0: no IP addresses available in
range set:
10.244.1.1-10.244.1.254"
```

- 则在Master节点上执行如下步骤：

```
#首先将集群中的节点都移除，移除方式参考2.1.4和2.1.5小节，然后执行下面的指令
sudo kubeadm reset
sudo systemctl stop kubelet
sudo systemctl stop docker
sudo rm -rf /var/lib/cni/
sudo rm -rf /var/lib/kubelet/*
sudo rm -rf /etc/cni/
sudo ifconfig cni0 down
sudo ifconfig flannel.1 down
sudo ifconfig docker0 down
sudo ip link delete cni0
sudo ip link delete flannel.1
##重启kubelet
sudo systemctl restart kubelet
##重启docker
sudo systemctl restart docker

##重新初始化集群
sudo kubeadm init --kubernetes-version=1.18.3 --apiserver-advertise-
address=192.168.1.107 --pod-network-cidr=10.244.0.0/16 --service-
cidr=10.96.0.0/12
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- 在Slave节点上执行如下步骤：

```
sudo kubeadm reset
sudo systemctl stop kubelet
sudo systemctl stop docker
sudo rm -rf /var/lib/cni/
sudo rm -rf /var/lib/kubelet/*
sudo rm -rf /etc/cni/
sudo ifconfig cni0 down
sudo ifconfig flannel.1 down
sudo ifconfig docker0 down
sudo ip link delete cni0
sudo ip link delete flannel.1
##重启kubelet
sudo systemctl restart kubelet
##重启docker
sudo systemctl restart docker

##使用kubeadm join将slave节点加入到集群中
```

- 如果上面的方法依然不能解决问题，执行下面的方法

```
#1.将所有的slave从集群中删除
#2.将Master删除
#3.在所有的节点中执行sudo kubeadm reset指令
#4.删除和cni相关的东西，使用如下指令：
sudo rm -rf /opt/cni
sudo rm -rf /run/flannel*
sudo rm -rf calcio*
sudo apt purge kubeadm kubelet kubectl
sudo apt purge kubernetes-cni
sudo apt autoremove
#5.重新安装kubeadm kubectl kubelet，并按照集群安装步骤继续操作
```

5.6.2 其它网络相关问题汇总

```
除了以上错误，其他可能的原因还有：
镜像拉取失败，比如：
（1）配置了错误的镜像
（2）Kubelet 无法访问镜像（国内环境访问 gcr.io 需要特殊处理
（3）私有镜像的密钥配置错误
（4）镜像太大，拉取超时（可以适当调整 kubelet 的 --image-pull-progress-deadline 和 --runtime-request-timeout 选项）
CNI 网络错误，一般需要检查 CNI 网络插件的配置，比如：
（1）无法配置 Pod 网络
（2）无法分配 IP 地址
容器无法启动，需要检查是否打包了正确的镜像或者是否配置了正确的容器参数等。
```

相关参考：

kubernetes中网络报错问题

Kubernetes-cni issue with 1.9.0

5.6.3 kubernetes未配置CNI时使用`kubectl get nodes`发现node处于Ready状态

这个可能是之前配置过CNI网络，想要重新更换网络，此时可以参考5.6.1中的解决方案。