终极大招

root破解思路

1重启系统
2出现内核选择界面时，按E进入
3将光标移动到Linux处，然后按键盘中的end键（不知道在哪儿的自己在键盘上面找），此时光标已经移动到开头为Linux那一行的末尾，在此处3加上rd.break，然后按Ctrl+x执行。
4switch_root:/#时，输入mount -o remount,rw /sysroot，然后敲击回车
5下一行还会出现switch_root:/#，此时输入chroot /sysroot，然后敲击回车
6此时输入新密码，待输入密码后在输入一次密码确认密码
7待输入密码完成后，输入touch /.autorelabel
8输入exit退出（此处需要连续输入两个exit才能完成退出），待重启后，密码修改成功。

什么是pod

Pod是Kubernetes中最小的逻辑单元，它由一组、一个或多个容器组成，本质上是以应用为中心的基础设施，每个Pod还包含了一个Pause容器，Pause容器是Pod的父容器，主要负责僵尸进程的回收管理，通过Pause容器可以使同一个Pod里面的多个容器共享存储、网络、PID、IPC等，本质上是共享名称空间以及存储资源的容器集，模拟不可变基础设施，删除后可通过资源清单重建，在设计上将具有超亲密关系的应用分别以不同容器形式运行在同一pod内部。
Pod也是一组容器组成的集合并包含这些容器的管理机制，k8s不调用容器 但是调用pod，k8s把用户期望的容器抽象为pod
init初始化容器，业务容器等

如何使k8s集群平稳运行/保证集群平稳运行策略

1、使用精简基础镜像
2、使用提供最佳正常运行时间的注册表
3、使用ImagePullSecrets对您的注册表进行身份验证
4、使用命名空间隔离环境
5、通过Labels 管理您的集群
6、使用annotation注释来跟踪重要的系统更改等
7、使用RBAC实施访问控制
8、使用Pod安全策略防止危险行为
9、使用网络策略实施网络控制/防火墙
10、使用Secrets来存储和管理必要的敏感信息
11、使用镜像扫clair描识别和修复镜像漏洞
12、遵循CI/CD方法基于不可变基础发布声明
13、使用Canary方法进行更新
14、实施监控并将其与SIEM集成
15、使用服务网格管理服务间通信
16、使用准入控制器解锁Kubernetes中的高级功能

k8s的通信方式

按照南北流量分为内部通信和外部通信
内部通信
pod to pod
service to service
service to pod
外部通信
node to node
cluster to node
cluster to cluster

pod间的亲和性

硬亲和：必须满足pod向scheduler的调度请求
软亲和：尽量满足pod向scheduler的调度请求
反亲和：不满住，或与之相反的调度请求，避免单点情况，使pod分布在不同节点上

docke 网络模式

应用程序层的报警
Kubernetes上运行的service服务的报警
Kubernetes基础设施的报警
master or node上的报警

K8s的镜像下载策略

Always：镜像标签为latest时，总是从指定的仓库中获取镜像；

Never：禁止从仓库中下载镜像，也就是说只能使用本地镜像；

IfNotPresent：仅当本地没有对应镜像时，才从目标仓库中下载；

6.基础报警类型

Kubernetes基础设施的报警

在容器编排层面的监控和报警有两个层面。一方面，我们需要监控Kubernetes所处理的服务是否符合所定义的要求。另一方面，我们需要确保Kubernetes的所有组件都正常运行。
由Kubernetes处理的服务
1.1 是否有足够的Pod/Container给每个应用程序运行？

Kubernetes可以通过Deployments、Replica Sets和Replication Controllers来处理具有多个Pod的应用程序。它们之间区别比较小，可以使用它们来维护运行相同应用程序的多个实例。运行实例的数量可以动态地进行伸缩，甚至可以通过自动缩放来实现自动化。

运行容器数量可能发生变化的原因有多种：因为节点失败或资源不足将容器重新调度到不同主机或者进行新版本的滚动部署等。如果在延长的时间段内运行的副本数或实例的数量低于我们所需的副本数，则表示某些组件工作不正常（缺少足够的节点或资源、Kubernetes或Docker Engine故障、Docker镜像损坏等等）。

在Kubernetes部署服务中，跨多个服务进行如下报警策略对比是非常有必要的：

```
timeAvg(kubernetes.replicaSet.replicas.running) <
timeAvg(kubernetes.replicaSet.replicas.desired)
```

正如之前所说，在重新调度与迁移过程运行中的实例副本数少于预期是可接受的，所以对每个容器需要关注配置项 .spec.minReadySeconds （表示容器从启动到可用状态的时间）。你可能也需要检查 .spec.strategy.rollingUpdate.maxUnavailable 配置项，它定义了在滚动部署期间有多少容器可以处于不可用状态。

下面是上述报警策略的一个示例，在 wordpress 命名空间中集群 kubernetes-dev 的一个deployment wordpress-wordpress。

1.2 是否有给定应用程序的任何Pod/Container？

与之前的警报类似，但具有更高的优先级（例如，这个应用程序是半夜获取页面的备选对象），将提醒是否没有为给定应用程序运行容器。

以下示例，在相同的deployment中增加警报，但不同的是1分钟内运行Pod <1时触发

1.3 重启循环中是否有任何Pod/Container？

在部署新版本时，如果没有足够的可用资源，或者只是某些需求或依赖关系不存在，容器或Pod最终可能会在循环中持续重新启动。这种情况称为“CrashLoopBackOff”。发生这种情况时，Pod永远不会进入就绪状态，从而被视为不可用，并且不会处于运行状态，因此，这种场景已被报警覆盖。尽管如此，笔者仍希望设置一个警报，以便在整个基础架构中捕捉到这种行为，立即了解具体问题。这不是那种中断睡眠的报警，但会有不少有用的信息。

这是在整个基础架构中应用的一个示例，在过去的2分钟内检测到4次以上的重新启动

监控Kubernetes系统服务
除了确保Kubernetes能正常完成其工作之外，我们还希望监测Kubernetes的内部健康状况。这将取决于Kubernetes集群安装的不同组件，因为这些可能会根据不同部署选择而改变，但有一些基本组件是相同的。

2.1 etcd是否正常运行？

etcd是Kubernetes的分布式服务发现、通信、命令通道。监控etcd可以像监控任何分布式键值数据库一样深入，但会使事情变得简单。

我们可以进一步去监控设置命令失败或者节点数，但是我们将会把它留给未来的etcd监控文章来描述。

2.2 集群中有足够的节点吗？

节点故障在Kubernetes中不是问题，调度程序将在其他可用节点中生成容器。但是，如果我们耗尽节点呢？或者已部署的应用程序的资源需求超过了现有节点？或者我们达到配额限制？

在这种情况下发出警报并不容易，因为这取决于你想要在待机状态下有多少个节点，或者你想要在现有节点上推送多少超额配置。可以使用监控度量值kube_node_status_ready和
kube_node_spec_unschedulable进行节点状态警报。

如果要进行容量级别报警，则必须将每个已调度Pod请求的CPU和memory进行累加，然后检查是否会覆盖每个节点，使用监控度量值kube_node_status_capacity_cpu_cores和
kube_node_status_capacity_memory_bytes。
在主机/节点层上的报警

主机层的警报与监视虚拟机或物理机区别不大，主要是关于主机是启动还是关闭或不可访问状态，以及资源可用性（CPU、内存、磁盘等）。

主要区别是现在警报的严重程度。在此之前，系统服务宕机可能意味着你的应用程序停止运行并且要紧急处理事故（禁止有效的高可用性）。而对于Kubernetes来说当服务发生异常，服务可以在主机之间移动，主机警报不应该不受重视。

让我们看看我们应该考虑的几个方面：
1. 主机宕机
如果主机停机或无法访问，我们希望收到通知。我们将在整个基础架构中应用此单一警报。我们打算给它一个5分钟的等待时间，因为我们不想看到网络连接问题上的嘈杂警报。你可能希望将其降低到1或2分钟，具体取决于你希望接收通知的速度。

2. 磁盘利用率
这是稍微复杂一些的警报。我们在整个基础架构的所有文件系统中应用此警报。我们将范围设置为 everywhere，并针对每个 fs.mountDir 设置单独的评估/警报。

以下是超过80％使用率的通用警报，但你可能需要不同的策略，如具有更高阈值（95％）或不同阈值的更高优先级警报（具体取决于文件系统）。

如果要为不同的服务或主机创建不同的阈值，只需更改要应用特定阈值的范围。
3. 一些其他资源
此类别中的常见资源是有关负载、CPU使用率、内存和交换空间使用情况的警报。如果在一定的时间内这些数据中的任何一个显著提升，可能就需要警报提醒您。我们需要在阈值、等待时间以及如何判定噪声警报之间进行折中。

如果您想为这些资源设置指标，请参考以下指标：

```
负载：load.average.1m，load.average.5m 和 load.average.15m
CPU：cpu.used.percent
memory：memory.used.percent 或 memory.bytes.used
swap：memory.swap.used.percent 或 memory.swap.bytes.used
```

一些人同样使用这个类别监控他们部分基础架构所在云服务提供商的资源。
Sysdig额外监控：监控系统调用
从警报触发并收到通知的那一刻起，真正的工作就开始为DevOps值班成员开展工作。有时运行手册就像检查工作负载中的轻微异常一样简单。这可能是云提供商的一个事件，但希望Kubernetes正在处理这个问题，并且只需要花费一些时间来应对负载。

但是如果一些更棘手的问题出现在我们面前，我们可以看到我们拥有的所有武器：提供者状态页面、日志（希望来自中心位置）、任何形式的APM或分布式追踪（如果开发人员安装了代码或者外部综合监测）。然后，我们祈祷这一事件在这些地方留下了一些线索，以便我们可以追溯到问题出现的多种来源原因。

我们怎么能够在警报被解除时自动dump所有系统调用？系统调用是所发生事情的真实来源，并将包含我们可以获得的所有信息。通过系统调用有可能发现触发警报的根本问题所在。Sysdig Monitor允许在警报触发时自动开始捕获所有系统调用。

例如，可以配置CrashLoopBackOff场景

只有当你安装代码后，Sysdig Monitor代理才能从系统调用拦截中计算出来相应的指标。考虑HTTP响应时间或SQL响应时间。Sysdig Monitor代理程序在套接字上捕获read()和write()系统调用，解码应用程序协议并计算转发到我们时间序列数据库中的指标。这样做的好处是可以在不触及开发人员代码的情况下获得仪表盘数据和警报

服务相应时间
服务可用性
SLA合规性
每秒请求的成功/失败数

HTTP请求数
数据库连接数、副本数
线程数、文件描述符数、连接数
中间件特定指标：Python uwsgi worker数量，JVM堆大小等

7.service与ingress的区别

service做服务发现
ingress做精准流量调度

8.各版本发行时间

Version	Release date	End of Life date[70]
1.0	10 July 2015	
1.1	9 November 2015	
1.2	16 March 2016	23 October 2016
1.3	1 July 2016	1 November 2016
1.4	26 September 2016	21 April 2017
1.5	12 December 2016	1 October 2017
1.6	28 March 2017	23 November 2017
1.7	30 June 2017	4 April 2018
1.8	28 August 2017	12 July 2018
1.9	15 December 2017	29 September 2018
1.10	28 March 2018	13 February 2019
1.11	3 July 2018	1 May 2019
1.12	27 September 2018	8 July 2019
1.13	3 December 2018	15 October 2019
1.14	25 March 2019	11 December 2019
1.15	20 June 2019	6 May 2020
1.16	22 October 2019	2 September 2020

1.16	22 October 2019	2 September 2020
1.17	9 December 2019	13 January 2021
1.18	25 March 2020	18 June 2021
1.19	26 August 2020[71]	28 October 2021
1.20	8 December 2020	28 February 2022
1.21	8 April 2021	28 June 2022
1.22	4 August 2021	28 October 2022
1.23	7 December 2021	28 February 2023
1.24	3 May 2022	29 September 2023

Legend: Old version | Older version, still maintained | **Latest version** | Latest preview version | Future release

sysdig概述

Sysdig捕获通过系统调用包含所有信息，包括网络流量、文件系统I/O和进程行为，提供故障排除所需的所有数据。
在容器环境中进行故障排除时，能够过滤和添加容器上下文信息（如Docker容器名称或Kubernetes元数据）使我们的排查过程更加轻松。
Sysdis在所有主流Linux发行版、OSX以及Windows上都能使用，下载它获得最新版本。Sysdig是一个开源工具，但该项目背后的公司也提供了一个商业产品来监视和排除多个主机上的容器和微服务

kubernetes node节点notready排错思路

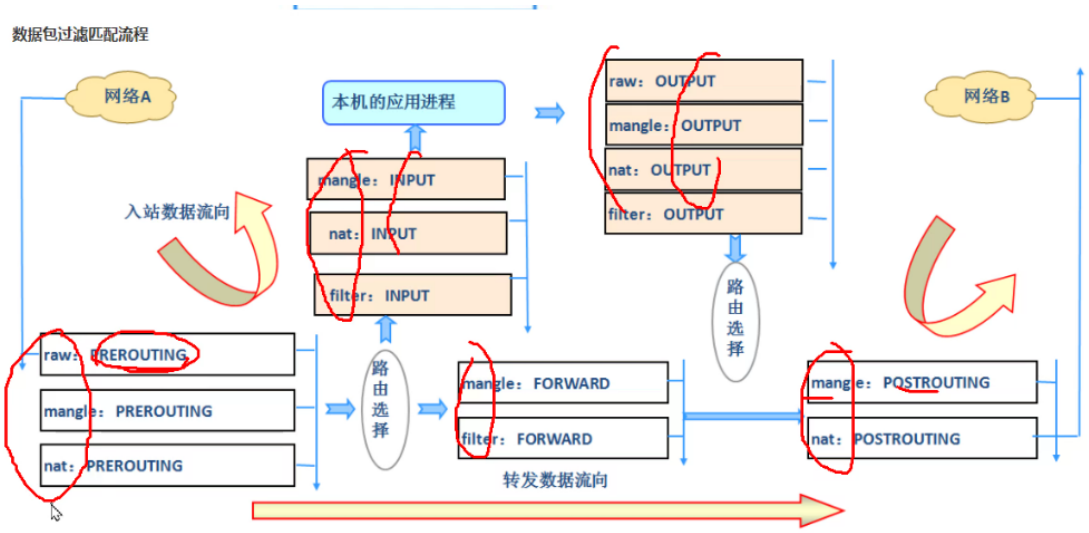
kubelet是运行于节点之上的主代理程序，需要向Master上报自身运行状态(心跳信息)。
节点心跳通过NodeStatus信息每10秒发送一次。如果在参数node-monitor-grace-period的指定时长内(默认40秒)没有收到心跳信息，节点控制器将把相应节点标记为NotReady，如果在参数pod-eviction-timeout指定时长内仍然没有收到节点信息，则节点控制器将会开始从该节点驱逐Pod对象。
在Kubernetes1.13版本以后，引入了与NodeStatus协同工作的节点租约(node lease)。每个节点的kubelet负责在专用的名称空间kube-node-lease中创建一个与节点同名的Lease对象以表示心跳信息，该资源隶属于coordination.k8s.io的新内置API群组。
重启kuberlet 服务

五表五链，流量运行规则

五链也是五个钩子函数，对外开放的，可以被外部应用调用
进入第一个链 PRE_ROUTING(预路由)然后根据路由表进行判决看是否是访问的内部IP，如果是则进入input链,如果规则不通过则穿过路由器流向FORWARD（IP forward 允许穿过路由）链然后进入另一端路由表判决看从哪个网卡流出，到达下一个目标地址并经过output，最后经由POST_ROUTING（路由后）

流量流向
流入服务：PRE_ROUTING > INPUT > kernal
流出服务：kernal > output > POSTROUTING
转发：PRE_ROUTING > FORWARD > POST_ROUTING

五表
filter:过滤规则表会自上而下逐一调用数据帧头部与rules进行比对;支持input,forward,output三链
nat:进行网络地址转换为分别为SNAT，DNAT支持除FORWARD链外其余4链
mangle:修改数据报文标记使其具有特征全部支持
raw:关闭启用的连接跟踪机制，加快封包穿越防火墙速度支持PRE_ROUTING，output
security:用于强制访问MAC控制网络规则，由Linux安全模块实现如selinux支持input，output，forward



追加规则

iptables -A(append 追加) input -s(source 源地址) 0.0.0.0 -j(跳跃) drop（抛弃）

tcpdump进行抓包 -i eth0 -nn icmp

来去，目标，

-v(详细的包内容)n(转换数据)L(查看规则)

某链内某个源地址抛弃的包数

合并规则(检查次数变少，效率提高)：支持不连续端口，-m multiport

```
[root@rocky8 ~]#iptables -RINPUT  2 -p tcp -m multiport --dports 22,80 -j ACCEPT
```

地址连续区间 iprange --src -range 10.0.0.0-10.0.0.6 -j accept

```
[root@rocky8 ~]#iptables -I INPUT 2 -m iprange --src-range 10.0.0.6-10.0.0.10 -j ACCEPT
```

mac地址扩展 只会收到给自己的报文

```
iptables -A INPUT -s 172.16.0.100 -m mac  --mac-source  00:50:56:12:34:56 -j ACCEPT
iptables -A INPUT -s 172.16.0.100  -j REJECT
```

arp -n 查看访问MAC地址缓存

地址解析协议，即ARP（Address Resolution Protocol），是根据IP地址获取物理地址的一个TCP/IP协议。主机发送信息时将包含目标IP地址的ARP请求广播到局域网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。地址解析协议是建立在网络中各个主机互相信任的基础上的，局域网络上的主机可以自主发送ARP应答消息，其他主机收到应答报文时不会检测该报文的真实性就会将其记入本机ARP缓存；由此攻击者就可以向某一主机发送伪ARP应答报文，使其发送的信息无法到达预期的主机或到达错误的主机，这就构成了一个ARP欺骗。ARP命令可用于查询本机ARP缓存中IP地址和MAC地址的对应关系、添加或删除静态对应关系等。相关协议有RARP、代理ARP。NDP用于在IPv6中代替地址解析协议。

工作过程
主机A的IP地址为192.168.1.1，MAC地址为0A-11-22-33-44-01；
主机B的IP地址为192.168.1.2，MAC地址为0A-11-22-33-44-02；
当主机A要与主机B通信时，地址解析协议可以将主机B的IP地址（192.168.1.2）解析成主机B的MAC地址，以下为工作流程：
第1步：根据主机A上的路由表内容，IP确定用于访问主机B的转发IP地址是192.168.1.2。然后A主机在自己的本地ARP缓存中检查主机B的匹配MAC地址。
第2步：如果主机A在ARP缓存中没有找到映射，它将询问192.168.1.2的硬件地址，从而将ARP请求帧广播到本地网络上的所有主机。源主机A的IP地址和MAC地址都包括在ARP请求中。本地网络上的每台主机都接收到ARP请求并且检查是否与自己的IP地址匹配。如果主机发现请求的IP地址与自己的IP地址不匹配，它将丢弃ARP请求。
第3步：主机B确定ARP请求中的IP地址与自己的IP地址匹配，则将主机A的IP地址和MAC地址映射添加到本地ARP缓存中。
第4步：主机B将包含其MAC地址的ARP回复消息直接发送回主机A。
第5步：当主机A收到从主机B发来的ARP回复消息时，会用主机B的IP和MAC地址映射更新ARP缓存。本机缓存是有生存期的，生存期结束后，将再次重复上面的过程。主机B的MAC地址一旦确定，主机A就能向主机B发送IP通信了。

string扩展

字符串检测

```
[root@rocky8 ~]#iptables -A OUTPUT -p tcp --sport 80 -m string --algo bm --from 62  --string   "google" -j REJECT
```

time模块 centos8无法使用

控制时间段访问，基于UTC时间 -u格林尼治时间 差8个小时 按天，月，周，时，分 都可控制

```
[root@centos7 ~]#iptables -A INPUT -m time --timestart 01:00 --timestop 10:00 -j REJECT
```

connlimit扩展

流量并发控制，根据每个客户端IP做并发连接数匹配，可以防止dos攻击

ss -nt 查看连接数

```
iptables -A INPUT -p tcp --dport 80 -m connlimit --connlimit-above 2 -j REJECT
```

当你访问我的80端口，并发连接数超过俩个就连接拒绝

limit 扩展

限速限流

```
[root@rocky8 ~]#iptables -I INPUT  -p icmp --icmp-type 8 -m limit --limit 20/minute --limit-burst 10 -j ACCEPT
[root@rocky8 ~]#iptables -A INPUT -p icmp -j REJECT
```

三过一

3.4.2.8 state扩展

state 扩展模块，可以根据"连接追踪机制"去检查连接的状态，较耗资源

conntrack机制：追踪本机上的请求和响应之间的关系

状态类型:

- NEW：新发出请求，连接追踪信息库中不存在此连接的相关信息条目，因此，将其识别为第一次发出的请求
- ESTABLISHED：NEW状态之后，连接追踪信息库中为其建立的条目失效之前期间内所进行的通信状态
- RELATED：新发起的但与已有连接相关联的连接，如：ftp协议中的数据连接与命令连接之间的关系
- INVALID：无效的连接，如flag标记不正确
- UNTRACKED：未进行追踪的连接，如：raw表中关闭追踪

鸡兔同笼

```
#!/bin/bash
HEAD=$1
FOOT=$2
RABBIT=$(((FOOT-HEAD-HEAD)/2))
CHOOK=$[HEAD-RABBIT]
echo RABBIT:$RABBIT
echo CHOOK:$CHOOK
###
[root@Rocky8 test_scripts]#bash chook_rabbit.sh 10 30
RABBIT:5
CHOOK:5
```

斐波那契数列求和

```
#!/bin/bash
list=(0 1)
for i in `seq 2 11`
do
      list[$i]=`expr ${list[-1]} + ${list[-2]}`
done
echo ${list[@]}

0 1 1 2 3 5 8 13 21 34 55 89
```

使用tcpdump 监听主机为10.0.0.1,tcp端口为80的数据,同时将输出结果保存成文件?

```
tcpdump tcp port 80 and host 10.0.0.1 -w ./target.cap
```

实现判断某网段内,当前在线的IP有哪些，能ping 通则认为在线

```
#!/bin/bash
for ip in `seq 1 255` ; do
{
ping -c 1 172.29.1.$ip > /dev/null 2>&1
if [ $? -eq 0 ]; then
   echo 172.29.1.$ip UP
fi
 }&
done
sleep 1
```

扫描主机端口状态

```
#!/bin/bash
yum install -y tcping
read -p"请输入IP地址:" IP
for PORT in `seq 1 65535` ; do
{
tcping $IP $PORT > /dev/null
if [ $? -eq 0 ]; then
   echo $IP:$PORT UP
fi
 }&
done
sleep 1
```

```
1+100

seq 10 |paste -s -d+|bc
awk -v i=1 -v sum=0 'BEGIN{while(i<=100){sum+=i;i++};print sum}'
awk 'BEGIN{ total=0;i=1;while(i<=100){total+=i;i++};print total}'
awk 'BEGIN{ total=0;i=1;do{ total+=i;i++;}while(i<=100);print total}'
awk 'BEGIN{sum=0;for(i=1;i<=100;i++){sum+=i};print sum}'
for((i=1,sum=0;i<=100;i++));do let sum+=i;done;echo $sum
awk 'BEGIN{total=0;for(i=1;i<=100;i++){total+=i};print total}'
sum=0;for i in {1..100};do let sum+=i;done ;echo sum=$sum
seq -s+ 100|bc
echo {1..100}|tr ' ' +|bc
seq 100|paste -sd +|bc
```

查看进程中的线程

```
grep -i threads /proc/PID/status
```

解释top命令

工作日时间，每10分钟执行一次磁盘空间检查，一旦发现任何分区利用率高于80%，就执行wall警报

```
#!/bin/bash
[ -d /data ] || mkdir /data
cat >> /var/spool/cron/root << EOF
* /10 0 * * /data/check_disk.sh
EOF
cat >/data/check_disk.sh<< EOF
#!/bin/bash
WARNING=80
df | sed -En '/^\/dev\/sd/s@^([^ ]+).* ([0-9]+)%.*@\1 \2@p'| while read
DEVICE
USE;do
   [ $USE -gt $WARNING ] && echo "$DEVICE will be full,USE:$USE" | wall
done
EOF
```

什么是硬链接和软链接?

1）硬链接

由于 Linux 下的文件是通过索引节点(inode)来识别文件，硬链接可以认为是一个指针，指向文件索引节点的指针，系统并不为它重新分配 inode 。每添加一个一个硬链接，文件的链接数就加 1 。

不足：1）不可以在不同文件系统的文件间建立链接；2）只有超级用户才可以为目录创建硬链接。

2）软链接

软链接克服了硬链接的不足，没有任何文件系统的限制，任何用户可以创建指向目录的符号链接。因而现在更为广泛使用，它具有更大的灵活性，甚至可以跨越不同机器、不同网络对文件进行链接。

不足：因为链接文件包含有原文件的路径信息，所以当原文件从一个目录下移到其他目录中，再访问链接文件，系统就找不到了，而硬链接就没有这个缺陷，你想怎么移就怎么移；还有它要系统分

配额外的空间用于建立新的索引节点和保存原文件的路径。

实际场景下，基本是使用软链接。总结区别如下：

硬链接不可以跨分区，软件链可以跨分区。

硬链接指向一个 inode 节点，而软链接则是创建一个新的 inode 节点。

删除硬链接文件，不会删除原文件，删除软链接文件，会把原文件删除。

etcd通过什么工作机制为k8s提供服务

etcd为其存储的数据提供了监听(watch)机制，用于监视和推送变更。API Server是Kubernetes集群中唯一能够与etcd通信的组件，它封装了这种监听机制，并借此同其他各组件高效协同。

容器运行原理

容器运行的根本：文件系统是静态的，容器运行真正靠的是文件系统树映射之下的我们所要启动的进程树为根本骨架进行运行的。系统向PID发送终止信，PID号为1的进程结束，容器退出运行转为停滞状态。

docker出现的原因

交付时面对的都是异构环境，平台异构，操作系统所支持的程序包异构形成一个交付时面向客户的复杂的产品矩阵，因此出现了docker images；

容器其实就是基于操作系统内核的名称空间隔离机制，docker是简化容器创建运行等一应管理运行的管理接口并且是CS架构

声明式API工作的基本逻辑

声明：API server是终态声明式API 向APIserver提交创建数据对象，如果声明的是pod ,controller 把声明提交的对象保存在etcd当中的数据进行调用；而真正将pod跑起来是由kubelet 负责

实现：由控制器controller从apiserver读取声明，并对声明中对应的具体实例进行实现，并确保期望状态和实际状态始终一致，由reconciltion loop调谐循环（始终持续执行）实现

多个pod运行后需要流量分发，直接分发优于跃点分发，对于每一个应用来讲，在客户端（存在于集群任何一个上）所在的内核上把应用定义成虚拟服务，将其对应的后端服务器pod ip地址都作为realserver 并关联起来。由service 定义一个虚拟IP地址 和端口，进行动态发现，由kv标签和标签选择器进行过滤和分发，到达后端realserver。

什么是容器编排技术：单个容器没有生产力，必须考虑到容器和容器所依赖的底层基础设施协同起来和其他容器共同组成信息系统的过程叫容器编排技术，要管理整个容器的生命周期，和依赖到的外部基础设施的周期，要使其联动起来，不需要太多的人为干预，需要配备的资源由系统自动配备到位。

虚拟服务器及上游对应端点流量分发实现过程：k8s自带servers插件，为每个应用创建负载均衡器；由servers 创建labelseleter(端口和标签选择器)，提供给kube-apiserver，随后由server控制器 结合kube-proxy进行工作，proxy将apiserver当中的service读出来，转化为当前节点内核当中的IPVS和realserver定义;转化由laberselecter挑选出的pod在apiserver中都有定义。

三次握手四次挥手

由客户端发起链接请求，如果双方从未建立服务链接，那么彼此端口都处于（closed）关闭状态.接下来由客户端主动向服务端发送带有SYN=1的确认标记位报文，在其发送后 客户端状态由closed 立即转变为sent（同步发送）状态，第一次握手结束。接下来服务端收到建立连接请求后，端口打开，由closed变为listen（监听状态），接受到客户端的SYN后，状态立即转变为RCVD（同步收到）状态，并向客户端回复ACK=1，SYN=1 的回复报文，表示同意建立链接第二次握手结束。最后由客户端向服务端回复ACK=1的报文，确认建立链接。双方状态都变成ESTAB 状态，第三次握手结束，开始进行数据发送。

握手的目的主要是确认通信安全，保证信息有来有回，双方同步确认。

数据传输完成后，由客户端发起释放链接请求，向服务端发送带有FIN=1的报文，状态由ESTAB转变为FIN-WAIT（终止等待）状态，第一次挥手结束；接下来服务端收到请求报文后，状态会由ESTAB转变为（closed-wait）关闭等待状态，并向客户端回复ACK=1的报文，表示收到释放请求，完成第二次挥手；然后服务端会向客户端发送ASK=1,FIN=1的报文，状态由closed-wait 转变为last-ack 半关闭状态，表示同意释放链接，完成第三次挥手；在第二次和第三次挥手过程中如果服务端有未传输完成的数据，会进行单向传输。最后，客户端会向服务端回复ACK=1的报文表示收到同意释放确认报文，双方断开链接，都处于closed状态完成第四次挥手。

HTTP 与 HTTPS 的区别

HTTP（HyperText Transfer Protocol：超文本传输协议）是一种用于分布式、协作式和超媒体信息系统的应用层协议。 简单来说就是一种发布和接收 HTML 页面的方法，被用于在 Web 浏览器和网站服务器之间传递信息。

HTTP 默认工作在 TCP 协议 80 端口，用户访问网站 **http://** 打头的都是标准 HTTP 服务。

HTTP 协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此，HTTP协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。

HTTPS（Hypertext Transfer Protocol Secure：超文本传输安全协议）是一种透过计算机网络进行安全通信的传输协议。HTTPS 经由 HTTP 进行通信，但利用 SSL/TLS 来加密数据包。HTTPS 开发的主要目的，是提供对网站服务器的身份认证，保护交换数据的隐私与完整性。

HTTPS 默认工作在 TCP 协议443端口，它的工作流程一般如以下方式：

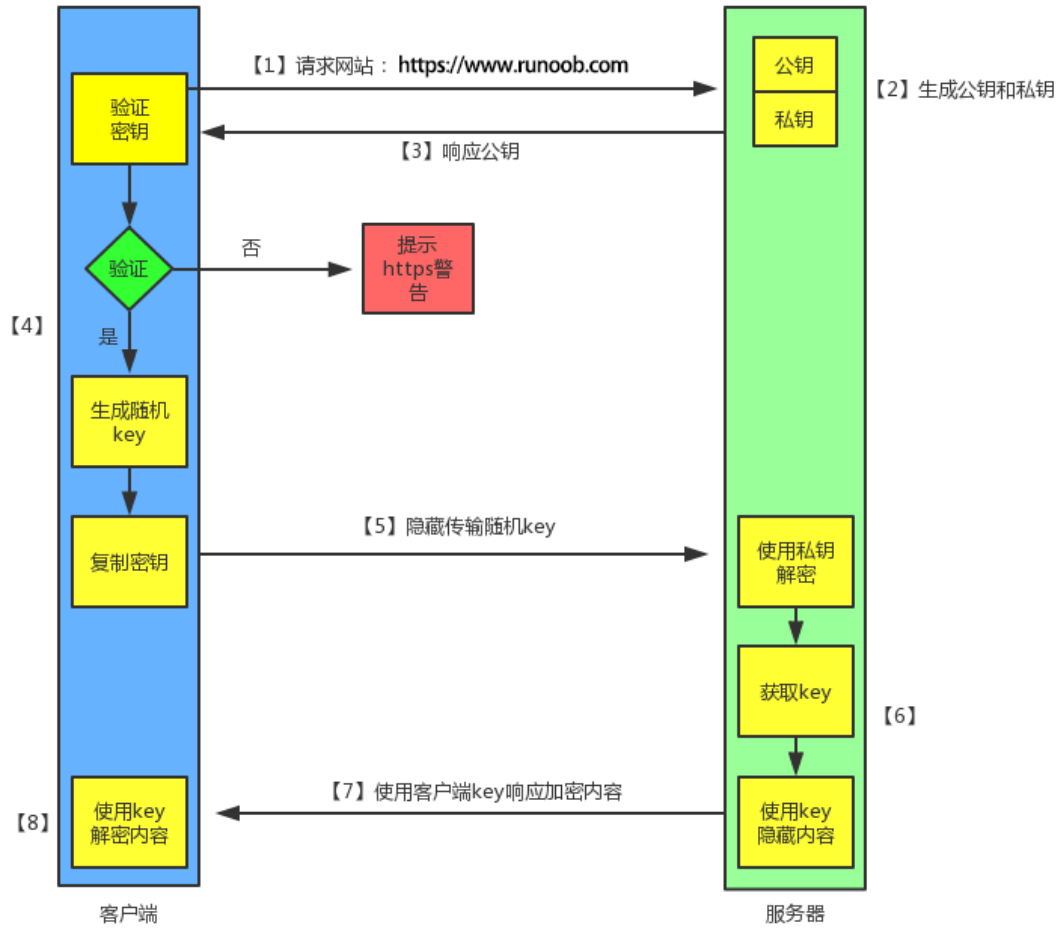
- 1、TCP 三次同步握手
- 2、客户端验证服务器数字证书
- 3、DH 算法协商对称加密算法的密钥、hash 算法的密钥
- 4、SSL 安全加密隧道协商完成
- 5、网页以加密的方式传输，用协商的对称加密算法和密钥加密，保证数据机密性；用协商的hash算法进行数据完整性保护，保证数据不被篡改。

- HTTP 明文传输，数据都是未加密的，安全性较差，HTTPS（SSL+HTTP） 数据传输过程是加密的，安全性较好。
- 使用 HTTPS 协议需要到 CA（Certificate Authority，数字证书认证机构） 申请证书，一般免费证书较少，因而需要一定费用。证书颁发机构如：Symantec、Comodo、GoDaddy 和 GlobalSign 等。

- HTTP 页面响应速度比 HTTPS 快，主要是因为 HTTP 使用 TCP 三次握手建立连接，客户端和服务器需要交换 3 个包，而 HTTPS除了 TCP 的三个包，还要加上 ssl 握手需要的 9 个包，所以一共是 12 个包。
- http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。
- HTTPS 其实就是建构在 SSL/TLS 之上的 HTTP 协议，所以，要比较 HTTPS 比 HTTP 要更耗费服务器资源。

HTTPS 的工作原理

我们都知道 HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。



1、客户端发起 HTTPS 请求

这个没什么好说的，就是用户在浏览器里输入一个 https 网址，然后连接到 server 的 443 端口。

2、服务端的配置

采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面(startssl 就是个不错的选择，有 1 年的免费服务)。

这套证书其实就是一对公钥和私钥，如果对公钥和私钥不太理解，可以想象成一把钥匙和一个锁头，只是全世界只有你一个人有这把钥匙，你可以把锁头给别人，别人可以用这个锁把重要的东西锁起来，然后发给你，因为只有你一个人有这把钥匙，所以只有你才能看到被这把锁锁起来的东西。

3、传送证书

这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间等等。

4、客户端解析证书

这部分工作是有客户端的TLS来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。

如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密，就好像上面说的，把随机值用锁头锁起来，这样除非有钥匙，不然看不到被锁住的内容。

5、传送加密信息

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。

6、服务端解密信息

服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密，所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

7、传输加密后的信息

这部分信息是服务段用私钥加密后的信息，可以在客户端被还原。

8、客户端解密信息

客户端用之前生成的私钥解密服务段传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听到了数据，也束手无策。

开机自检程序

```
1  开机自检（POST），初始化部分硬件

2  搜素可用于引导的启动设备（如磁盘的MBR）

3  读取并将控制权交给系统启动加载器（grub2）

4  启动加载器加载器配置，显示可用配置菜单

5  启动加载器加载内核及initramfs，置入内存

6  启动加载器将控制权交给内核

7  由内核查找initramfs中的硬件驱动，作为PID=1从initramfs执行/sbin/init

（在RHEL7中，为systemd，并包含udev守护进程

8  Systemd执行initrd.target的所有单元（包含将文件系统挂载到/sysroot）

9  内核root文件系统从initramfsroot文件系统切换到/sysroot上的系统root文件系统

10  Systemd查找默认目标（target），然后启动该target的所有单元
```

四层与七层代理区别

四层是TCP层，使用IP+端口的方式。类似路由器，只是修改下IP地址，然后转发给后端服务器，TCP三次握手是直接和后端连接的。只不过在后端机器上看到的都是与代理机的IP的established而已。

7层代理则必须要先和代理机三次握手后，才能得到7层（HTT层）的具体内容，然后再转发。意思就是代理机必须要与client和后端的机器都要建立连接。显然性能不行了，但胜在于七层，能写更多的转发规则

具体内容可参考：https://www.jianshu.com/p/fa937b8e6712

一、简介

** 所谓四层就是基于IP+端口的负载均衡；七层就是基于URL等应用层信息的负载均衡；**同理，还有基于MAC地址的二层负载均衡和基于IP地址的三层负载均衡。 换句换说，二层负载均衡会通过一个虚拟MAC地址接收请求，然后再分配到真实的MAC地址；三层负载均衡会通过一个虚拟IP地址接收请求，然后再分配到真实的IP地址；四层通过虚拟IP+端口接收请求，然后再分配到真实的服务器；七层通过虚拟的URL或主机名接收请求，然后再分配到真实的服务器。

** 所谓的四到七层负载均衡，就是在对后台的服务器进行负载均衡时，依据四层的信息或七层的信息来决定怎么样转发流量。** 比如四层的负载均衡，就是通过发布三层的IP地址（VIP），然后加四层的端口号，来决定哪些流量需要做负载均衡，对需要处理的流量进行NAT处理，转发至后台服务器，并记录下这个TCP或者UDP的流量是由哪台服务器处理的，后续这个连接的所有流量都同样转发到同一台服务器处理。七层的负载均衡，就是在四层的基础上（没有四层是绝对不可能有七层的），再考虑应用层的特征，比如同一个Web服务器的负载均衡，除了根据VIP加80端口辨别是否需要处理的流量，还可根据七层的URL、浏览器类别、语言来决定是否要进行负载均衡。举个例子，如果你的Web服务器分成两组，一组是中文语言的，一组是英文语言的，那么七层负载均衡就可以当用户来访问你的域名时，自动辨别用户语言，然后选择对应的语言服务器组进行负载均衡处理。

负载均衡器通常称为四层交换机或七层交换机。四层交换机主要分析IP层及TCP/UDP层，实现四层流量负载均衡。七层交换机除了支持四层负载均衡以外，还有分析应用层的信息，如HTTP协议URI或Cookie信息。

负载均衡分为L4 switch（四层交换），即在OSI第4层工作，就是TCP层啦。此种Load Balance不理解应用协议（如HTTP/FTP/MySQL等等）。例子：LVS，F5。

另一种叫做L7 switch（七层交换），OSI的最高层，应用层。此时，该Load Balancer能理解应用协议。例子： haproxy，MySQL Proxy。

注意：上面的很多Load Balancer既可以做四层交换，也可以做七层交换。

二、区别

技术原理上

所谓四层负载均衡，也就是主要通过报文中的目标地址和端口，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。

以常见的TCP为例，负载均衡设备在接收到第一个来自客户端的SYN 请求时，即通过上述方式选择一个最佳的服务器，并对报文中目标IP地址进行修改(改为后端服务器IP），直接转发给该服务器。TCP的连接建立，即三次握手是客户端和服务器直接建立的，负载均衡设备只是起到一个类似路由器的转发动作。在某些部署情况下，为保证服务器回包可以正确返回给负载均衡设备，在转发报文的同时可能还会对报文原来的源地址进行修改。

区别

所谓七层负载均衡，也称为“内容交换”，也就是主要通过报文中的真正有意义的应用层内容，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。

以常见的TCP为例，负载均衡设备如果要根据真正的应用层内容再选择服务器，只能先代理最终的服务器和客户端建立连接(三次握手)后，才可能接受到客户端发送的真正应用层内容的报文，然后再根据该报文中的特定字段，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。负载均衡设备在这种情况下，更类似于一个代理服务器。负载均衡和前端的客户端以及后端的服务器会分别建立TCP连接。所以从这个技术原理上来看，七层负载均衡明显的对负载均衡设备的要求更高，处理七层的能力也必然会低于四层模式的部署方式。

应用场景

七层应用负载的好处，是使得整个网络更智能化。例如访问一个网站的用户流量，可以通过七层的方式，将对图片类的请求转发到特定的图片服务器并可以使用缓存技术；将对文字类的请求可以转发到特定的文字服务器并可以使用压缩技术。当然这只是七层应用的一个小案例，从技术原理上，这种方式可以对客户端的请求和服务器的响应进行任意意义上的修改，极大的提升了应用系统在网络层的灵活性。很多在后台，例如Nginx或者Apache上部署的功能可以前移到负载均衡设备上，例如客户请求中的Header重写，服务器响应中的关键字过滤或者内容插入等功能。

另外一个常常被提到功能就是安全性。网络中最常见的SYN Flood攻击，即黑客控制众多源客户端，使用虚假IP地址对同一目标发送SYN攻击，通常这种攻击会大量发送SYN报文，耗尽服务器上的相关资源，以达到Denial of Service(DoS)的目的。从技术原理上也可以看出，四层模式下这些SYN攻击都会被转发到后端的服务器上；而七层模式下这些SYN攻击自然在负载均衡设备上就截止，不会影响后台服务器的正常运营。另外负载均衡设备可以在七层层面设定多种策略，过滤特定报文，例如SQL Injection等应用层面的特定攻击手段，从应用层面进一步提高系统整体安全。

现在的7层负载均衡，主要还是着重于应用HTTP协议，所以其应用范围主要是众多的网站或者内部信息平台等基于B/S开发的系统。 4层负载均衡则对应其他TCP应用，例如基于C/S开发的ERP等系统。

三、Nginx、LVS及HAProxy负载均衡软件的优缺点

负载均衡 （Load Balancing） 建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力，同时能够提高网络的灵活性和可用性。

Nginx/LVS/HAProxy是目前使用最广泛的三种负载均衡软件。

一般对负载均衡的使用是随着网站规模的提升根据不同的阶段来使用不同的技术。具体的应用需求还得具体分析，如果是中小型的Web应用，比如日PV小于1000万，用Nginx就完全可以了；如果机器不少，可以用DNS轮询，LVS所耗费的机器还是比较多的；大型网站或重要的服务，且服务器比较多时，可以考虑用LVS。

一种是通过硬件来进行，常见的硬件有比较昂贵的F5和Array等商用的负载均衡器，它的优点就是有专业的维护团队来对这些服务进行维护、缺点就是花销太大，所以对于规模较小的网络服务来说暂时还没有需要使用；另外一种就是类似于Nginx/LVS/HAProxy的基于 Linux的开源免费的负载均衡软件，这些都是通过软件级别来实现，所以费用非常低廉。

目前关于网站架构一般比较合理流行的架构方案：Web前端采用Nginx/HAProxy+ Keepalived作负载均衡器；后端采用 MySQL数据库一主多从和读写分离，采用LVS+Keepalived的架构。当然要根据项目具体需求制定方案。

下面说说各自的特点和适用场合。

Nginx的优点是：

工作在网络的7层之上，可以针对http应用做一些分流的策略，比如针对域名、目录结构，它的正则规则比HAProxy更为强大和灵活，这也是它目前广泛流行的主要原因之一，Nginx单凭这点可利用的场合就远多于LVS了。

Nginx对网络稳定性的依赖非常小，理论上能ping通就就能进行负载功能，这个也是它的优势之一；相反LVS对网络稳定性依赖比较大。

Nginx安装和配置比较简单，测试起来比较方便，它基本能把错误用日志打印出来。LVS的配置、测试就要花比较长的时间了，LVS对网络依赖比较大。

可以承担高负载压力且稳定，在硬件不差的情况下一般能支撑几万次的并发量，负载度比LVS相对小些。

Nginx可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点，不过其中缺点就是不支持url来检测。比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，Nginx会把上传切到另一台服务器重新处理，而LVS就直接断掉了，如果是上传一个很大的文件或者很重要的文件的话，用户可能会因此而不满。

Nginx不仅仅是一款优秀的负载均衡器/反向代理软件，它同时也是功能强大的Web应用服务器。LNMP也是近几年非常流行的web架构，在高流量的环境中稳定性也很好。

Nginx现在作为Web反向加速缓存越来越成熟了，速度比传统的Squid服务器更快，可以考虑用其作为反向代理加速器。

Nginx可作为中层反向代理使用，这一层面Nginx基本上无对手，唯一可以对比Nginx的就只有 lighttpd了，不过 lighttpd目前还没有做到Nginx完全的功能，配置也不那么清晰易读，社区资料也远远没Nginx活跃。

Nginx也可作为静态网页和图片服务器，这方面的性能也无对手。还有Nginx社区非常活跃，第三方模块也很多。

Nginx的缺点是：

Nginx仅能支持http、https和Email协议，这样就在适用范围上面小些，这个是它的缺点。

对后端服务器的健康检查，只支持通过端口来检测，不支持通过url来检测。不支持Session的直接保持，但能通过ip_hash来解决。

LVS：使用Linux内核集群实现一个高性能、高可用的负载均衡服务器，它具有很好的可伸缩性（Scalability)、可靠性（Reliability)和可管理性（Manageability)。

LVS的优点是：

抗负载能力强、是工作在网络4层之上仅作分发之用，没有流量的产生，这个特点也决定了它在负载均衡软件里的性能最强的，对内存和cpu资源消耗比较低。

配置性比较低，这是一个缺点也是一个优点，因为没有可太多配置的东西，所以并不需要太多接触，大大减少了人为出错的几率。

工作稳定，因为其本身抗负载能力很强，自身有完整的双机热备方案，如LVS+Keepalived。

无流量，LVS只分发请求，而流量并不从它本身出去，这点保证了均衡器IO的性能不会受到大流量的影响。

应用范围比较广，因为LVS工作在4层，所以它几乎可以对所有应用做负载均衡，包括http、数据库、在线聊天室等等。

LVS的缺点是：

软件本身不支持正则表达式处理，不能做动静分离；而现在许多网站在这方面都有较强的需求，这个是Nginx/HAProxy+Keepalived的优势所在。

如果是网站应用比较庞大的话，LVS/DR+Keepalived实施起来就比较复杂了，特别后面有 Windows Server的机器的话，如果实施及配置还有维护过程就比较复杂了，相对而言，Nginx/HAProxy+Keepalived就简单多了。

HAProxy的特点是：

HAProxy也是支持虚拟主机的。

HAProxy的优点能够补充Nginx的一些缺点，比如支持Session的保持，Cookie的引导；同时支持通过获取指定的url来检测后端服务器的状态。

HAProxy跟LVS类似，本身就只是一款负载均衡软件；单纯从效率上来讲HAProxy会比Nginx有更出色的负载均衡速度，在并发处理上也是优于Nginx的。

HAProxy支持TCP协议的负载均衡转发，可以对MySQL读进行负载均衡，对后端的MySQL节点进行检测和负载均衡，大家可以用LVS+Keepalived对MySQL主从做负载均衡。

HAProxy负载均衡策略非常多，HAProxy的负载均衡算法现在具体有如下8种：

① roundrobin，表示简单的轮询，这个不多说，这个是负载均衡基本都具备的；

② static-rr，表示根据权重，建议关注；

③ leastconn，表示最少连接者先处理，建议关注；

④ source，表示根据请求源IP，这个跟Nginx的IP_hash机制类似，我们用其作为解决session问题的一种方法，建议关注；

⑤ ri，表示根据请求的URI；

⑥ rl_param，表示根据请求的URl参数’balance url_param’ requires an URL parameter name；

⑦ hdr(name)，表示根据HTTP请求头来锁定每一次HTTP请求；

⑧ rdp-cookie(name)，表示根据据cookie(name)来锁定并哈希每一次TCP请求。

Nginx和LVS对比的总结：

Nginx工作在网络的7层，所以它可以针对http应用本身来做分流策略，比如针对域名、目录结构等，相比之下LVS并不具备这样的功能，所以Nginx单凭这点可利用的场合就远多于LVS了；但Nginx有用的这些功能使其可调整度要高于LVS，所以经常要去触碰触碰，触碰多了，人为出问题的几率也就会大。

Nginx对网络稳定性的依赖较小，理论上只要ping得通，网页访问正常，Nginx就能连得通，这是Nginx的一大优势！Nginx同时还能区分内外网，如果是同时拥有内外网的节点，就相当于单机拥有了备份线路；LVS就比较依赖于网络环境，目前来看服务器在同一网段内并且LVS使用direct方式分流，效果较能得到保证。另外注意，LVS需要向托管商至少申请多一个ip来做Visual IP，貌似是不能用本身的IP来做VIP的。要做好LVS管理员，确实得跟进学习很多有关网络通信方面的知识，就不再是一个HTTP那么简单了。

Nginx安装和配置比较简单，测试起来也很方便，因为它基本能把错误用日志打印出来。LVS的安装和配置、测试就要花比较长的时间了；LVS对网络依赖比较大，很多时候不能配置成功都是因为网络问题而不是配置问题，出了问题要解决也相应的会麻烦得多。

Nginx也同样能承受很高负载且稳定，但负载度和稳定度差LVS还有几个等级：Nginx处理所有流量所以受限于机器IO和配置；本身的bug也还是难以避免的。

Nginx可以检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点。目前LVS中 ldirectd也能支持针对服务器内部的情况来监控，但LVS的原理使其不能重发请求。比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，Nginx会把上传切到另一台服务器重新处理，而LVS就直接断掉了，如果是上传一个很大的文件或者很重要的文件的话，用户可能会因此而恼火。

Nginx对请求的异步处理可以帮助节点服务器减轻负载，假如使用 apache直接对外服务，那么出现很多的窄带链接时apache服务器将会占用大 量内存而不能释放，使用多一个Nginx做apache代理的话，这些窄带链接会被Nginx挡住，apache上就不会堆积过多的请求，这样就减少了相当多的资源占用。这点使用squid也有相同的作用，即使squid本身配置为不缓存，对apache还是有很大帮助的。

Nginx能支持http、https和email（email的功能比较少用），LVS所支持的应用在这点上会比Nginx更多。在使用上，一般最前端所采取的策略应是LVS，也就是DNS的指向应为LVS均衡器，LVS的优点令它非常适合做这个任务。重要的ip地址，最好交由LVS托管，比如数据库的 ip、webservice服务器的ip等等，这些ip地址随着时间推移，使用面会越来越大，如果更换ip则故障会接踵而至。所以将这些重要ip交给 LVS托管是最为稳妥的，这样做的唯一缺点是需要的VIP数量会比较多。Nginx可作为LVS节点机器使用，一是可以利用Nginx的功能，二是可以利用Nginx的性能。当然这一层面也可以直接使用squid，squid的功能方面就比Nginx弱不少了，性能上也有所逊色于Nginx。Nginx也可作为中层代理使用，这一层面Nginx基本上无对手，唯一可以撼动Nginx的就只有lighttpd了，不过lighttpd目前还没有能做到 Nginx完全的功能，配置也不那么清晰易读。另外，中层代理的IP也是重要的，所以中层代理也拥有一个VIP和LVS是最完美的方案了。具体的应用还得具体分析，如果是比较小的网站（日PV小于1000万），用Nginx就完全可以了，如果机器也不少，可以用DNS轮询，LVS所耗费的机器还是比较多的；大型网站或者重要的服务，机器不发愁的时候，要多多考虑利用LVS。

dockerfile

docker网络模式

bridge网络模式
本模式是docker的默认模式，即不指定任何模式就是bridge模式，也是使用比较多的模式，此模式创建的容器会为每一个容器分配自己的网络 IP 等信息，并将容器连接到一个虚拟网桥与外界通信
可以和外部网络之间进行通信，通过SNAT访问外网，使用DNAT可以让容器被外部主机访问，所以此模

式也称为NAT模式
此模式宿主机需要启动ip_forward功能
bridge网络模式特点
网络资源隔离：不同宿主机的容器无法直接通信，各自使用独立网络
无需手动配置：容器默认自动获取172.17.0.0/16的IP地址，此地址可以修改
可访问外网：利用宿主机的物理网卡，SNAT连接外网
外部主机无法直接访问容器：可以通过配置DNAT接受外网的访问
低性能较低：因为可通过NAT，网络转换带来更的损耗
端口管理繁琐：每个容器必须手动指定唯一的端口，容器产生端口冲容

Host 网络模式
如果指定host模式启动的容器，那么新创建的容器不会创建自己的虚拟网卡，而是直接使用宿主机的网卡和IP地址，因此在容器里面查看到的IP信息就是宿主机的信息，访问容器的时候直接使用宿主机IP+容器端口即可，不过容器内除网络以外的其它资源，如：文件系统、系统进程等仍然和宿主机保持隔离
此模式由于直接使用宿主机的网络无需转换，网络性能最高，但是各容器内使用的端口不能相同，适用
Host 网络模式特点：
使用参数 --network host 指定
共享宿主机网络
网络性能无损耗
网络故障排除相对简单
各容器网络无隔离
网络资源无法分别统计
端口管理困难：容易产生端口冲突
不支持端口映射
于运行容器端口比较固定的业务

none 模式
在使用none 模式后，Docker 容器不会进行任何网络配置，没有网卡、没有IP也没有路由，因此默认无法与外界通信，需要手动添加网卡配置IP等，所以极少使用 none模式特点
使用参数 --network none 指定
默认无网络功能，无法和外部通信
无法实现端口映射
适用于测试环境

Container 模式
使用此模式创建的容器需指定和一个已经存在的容器共享一个网络，而不是和宿主机共享网络，新创建的容器不会创建自己的网卡也不会配置自己的IP，而是和一个被指定的已经存在的容器共享IP和端口范围，因此这个容器的端口不能和被指定容器的端口冲突，除了网络之外的文件系统、进程信息等仍然保持相互隔离，两个容器的进程可以通过lo网卡进行通信
Container 模式特点
使用参数 --network container:名称或ID 指定
与宿主机网络空间隔离
空器间共享网络空间
适合频繁的容器间的网络通信
直接使用对方的网络，较少使用

自定义模式
除了以上的网络模式，也可以自定义网络，使用自定义的网段地址，网关等信息
注意：自定义网络内的容器可以直接通过容器名进行相互的访问,而无需使用 --link
可以使用自定义网络模式,实现不同集群应用的独立网络管理,而互不影响,而且在网一个网络内,可以直接利用容器名相互访问，非常便利

ZooKeeper 集群选举过程

2.4.1.3 选举过程

ZooKeeper 集群选举过程：
当集群中的 zookeeper 节点启动以后，会根据配置文件中指定的 zookeeper节点地址进行leader 选择操作，过程如下：
每个zookeeper 都会发出投票，由于是第一次选举leader，因此每个节点都会把自己当做leader 角色进行选举，每个zookeeper 的投票中都会包含自己的myid和zxid，此时zookeeper 1 的投票为myid 为 1，初始zxid有一个初始值0x0，后期会随着数据更新而自动变化，zookeeper2 的投票为myid 为2，初始zxid 为初始生成的值。
每个节点接受并检查对方的投票信息，比如投票时间、是否状态为LOOKING状态的投票。
对比投票，优先检查zxid，如果zxid 不一样则 zxid 大的为leader，如果zxid相同则继续对比 myid，myid 大的一方为 leader
成为 Leader 的必要条件： Leader 要具有最高的zxid；当集群的规模是 n 时，集群中大多数的机器（至少n/2+1）得到响应并follower 选出的 Leader。
心跳机制：Leader 与 Follower 利用 PING 来感知对方的是否存活，当 Leader无法响应PING 时，将重新发起 Leader 选举。当 Leader 服务器出现网络中断、崩溃退出与重启等异常情况时，
ZAB(Zookeeper Atomic Broadcast) 协议就会进入恢复模式并选举产生新的Leader服务器。这个过程

大致如下：
Leader Election（选举阶段）：节点在一开始都处于选举阶段，只要有一个节点得到超半数节点的票数，它就可以当选准 leader。
Discovery（发现阶段）：在这个阶段，followers 跟准 leader 进行通信，同步 followers 最近接收的事务提议。
Synchronization（同步阶段）:同步阶段主要是利用 leader 前一阶段获得的最新提议历史，同步集群中所有的副本。同步完成之后 准 leader 才会成为真正的 leader。
Broadcast（广播阶段） ：到了这个阶段，Zookeeper 集群才能正式对外提供事务服务，并且 leader 可以进行消息广播。同时如果有新的节点加入，还需要对新节点进行同步
ZAB 协议介绍
2.4.1.4 ZooKeeper 集群特性
整个集群中只要有超过集群数量一半的 zookeeper工作是正常的，那么整个集群对外就是可用的
假如有 2 台服务器做了一个 Zookeeper 集群，只要有任何一台故障或宕机，那么这个 ZooKeeper集群就不可用了，因为剩下的一台没有超过集群一半的数量，但是假如有三台zookeeper 组成一个集群， 那么损坏一台就还剩两台，大于 3台的一半，所以损坏一台还是可以正常运行的，但是再损坏一台就只剩一台集群就不可用了。那么要是 4 台组成一个zookeeper集群，损坏一台集群肯定是正常的，那么损坏两台就还剩两台，那么2台不大于集群数量的一半，所以 3 台的 zookeeper 集群和 4 台的 zookeeper 集
群损坏两台的结果都是集群不可用，以此类推 5 台和 6 台以及 7 台和 8台都是同理
另外偶数节点可以会造成"脑裂"现象,所以这也就是为什么集群一般都是奇数的原因。

消息队列

消息队列主要有以下应用场景
削峰填谷
诸如电商业务中的秒杀、抢红包、企业开门红等大型活动时皆会带来较高的流量脉冲，或因没做相 应的保护而导致系统超负荷甚至崩溃，或因限制太过导致请求大量失败而影响用户体验，消息队列 可提供削峰填谷的服务来解决该问题。
异步解耦
交易系统作为淘宝等电商的最核心的系统，每笔交易订单数据的产生会引起几百个下游业务系统的 关注，包括物流、购物车、积分、流计算分析等等，整体业务系统庞大而且复杂，消息队列可实现 异步通信和应用解耦，确保主站业务的连续性。细数日常中需要保证顺序的应用场景非常多，例如证券交易过程时间优先原则，交易系统中的订单 创建、支付、退款等流程，航班中的旅客登机消息处理等等。与先进先出FIFO（First In First Out）原理类似，消息队列提供的顺序消息即保证消息FIFO。
分布式事务一致性
交易系统、支付红包等场景需要确保数据的最终一致性，大量引入消息队列的分布式事务，既可以 实现系统之间的解耦，又可以保证最终的数据一致性。 大数据分析 数据在“流动”中产生价值，传统数据分析大多是基于批量计算模型，而无法做到实时的数据分析， 利用消息队列与流式计算引擎相结合，可以很方便的实现业务数据的实时分析。
分布式缓存同步
电商的大促，各个分会场琳琅满目的商品需要实时感知价格变化，大量并发访问数据库导致会场页 面响应时间长，集中式缓存因带宽瓶颈，限制了商品变更的访问流量，通过消息队列构建分布式缓 存，实时通知商品数据的变化 蓄流压测 线上有些链路不方便做压力测试，可以通过堆积一定量消息再放开来压测

kafak的优点

分布式：多机实现,不允许单机
分区：一个消息.可以拆分出多个，分别存储在多个位置
多副本：防止信息丢失，可以多来几个备份
多订阅者：可以有很多应用连接kafka
Zookeeper：早期版本的Kafka依赖于zookeeper， 2021年4月19日Kafka 2.8.0正式发布，此版本包括了很多重要改动，最主要的是kafka通过自我管理的仲裁来替代ZooKeeper，即Kafka将不再需要ZooKeeper!
优势
Kafka 通过 O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以 TB 级别以上的消息存储也能够保持长时间的稳定性能。
高吞吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。支持通过Kafka 服务器分区消息。
分布式： Kafka 基于分布式集群实现高可用的容错机制，可以实现自动的故障转移
顺序保证：在大多数使用场景下，数据处理的顺序都很重要。大部分消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。 Kafka保证一个Partiton内的消息的有序性（分区间数据是无序的，如果对数据的顺序有要求，应将在创建主题时将分区数partitions设置为1）
支持 Hadoop 并行数据加载
通常用于大数据场合,传递单条消息比较大，而Rabbitmq 消息主要是传输业务的指令数据,单条数据较小

常用基础题

多个IP去重，取前几的IP
awk '{print $1}' test|sort | unip -c | socr -nr |head -10
awk -F" +|:" '{print $(NF-2)}' test |sort |unip -c |sort -nr |hesd -n10
提取分区利用率
df | awk '{print $1,$5}'
提取IP
1.hostname -I | cat -A
2.ifconfig eth0 | awk '/netwark/{print $2}'
3.ifconfig eth0 | sed -rn '2s/^[^0-9]+([0-9.]+) .*$/\1/p'
1-10 按列排序
seq 10 | awk 'n++'
NF 是列 NR是行

awk 函数变量
NF 行的字段个数（列数）
FS 分隔符，默认为空格
OFS 定义输出字段分隔符，默认空格
RS 输入记录分隔符，默认换行
ORS 输出记录分隔符，默认换行
$0 当前处理行的所有记录
$1,$2… 文件中每行以间隔符号分隔的不同字符

监控项目

zabbix监控Mysql:
监控Mysql3306连接线程
监控Mysql慢SQL数量
监控Mysql3306连接线程
监控IO/SQL线程状态
监控TPS/QPS平均多少

ningx优化

最大连接数
1.gizp压缩，减少网络消耗
2.nginxd的work进程于cpu的亲缘性绑定；让nginx的worker进程和cpu进行结亲绑定，这就极大减少了nginx的工作进程在不同的cpu核心上的来回跳转，减少了CPU对进程的资源分配与回收以及内存管理等，因此可以有效的提升nginx服务器的性能
3.更改nginx的worker进程的优先级；
数字越小优先级越高；普通用户能改优先级，只能改大，不能改小，root不限；
避免nginx惊群；
accept_mutex on; #on为同一时刻一个请求轮流由worker进程处理,而防止被同时唤醒所有worker,避免多个睡眠进程被唤醒的设置，默认为off，新请求会唤醒所有worker进程,此过程也称为"惊群"，因此nginx刚安装完以后要进行适当的优化。建议设置为on 在全局配置里面设置

4.开启nginx服务器的每个worker进程可以接收多个新的网络连接
multi_accept on; #on时Nginx服务器的每个工作进程可以同时接受多个新的网络连接，此指令默认为off，即默认为一个工作进程只能一次接受一个新的网络连接，打开后几个同时接受多个。建议设置为on
}

优化nginx连接超时用的什么参数

当客户端与服务器端三次握手正式建立tcp以后，默认情况下，除非客户端或服务器端关闭上层socket，否则tcp会始终保持连接，如果这个时候网络断掉，这个链接就会变成一个死链接，会占用服务器资源。
那如何解决这种问题呢？
1、大多数的上游应用会通过心跳机制来检测对方是否存活，不存会则由上游应用程序关闭socket释放链接。
2、通过tcp带的keeplive机制来检测，已Java为例，在socket编程中需要显示开启tcp的keeplive 。
————————————————
so_keepalive=on   //表示开启tcp探活，并且使用系统内核的参数。
————————————————

so_keepalive=off
so_keepalive=30s::10   //表示开启tcp探活，30s后nginx会发送探活包，时间间隔使用系统默认的，发送10次探活包

centos7添加开机启动服务或脚本

1.由于在centos7中/etc/rc.d/rc.local的权限被降低了，所以需要赋予其可执行权
chmod +x /etc/rc.d/rc.local
2./etc/init.d是/etc/rc.d/init.d的软链接，可以通过ll命令查看。当Linux启动时，会寻找这些目录中的服务脚本，并根据脚本的运行级别确定不同的启动级别。

将脚本移动到/etc/rc.d/init.d目录下
mv /usr/local/app/start.sh /etc/rc.d/init.d

增加脚本的可执行权限
chmod +x /usr/local/app/start.sh

添加脚本到开机自动启动项目中
cd /etc/rc.d/init.d
chkconfig --add start.sh 添加为系统服务
chkconfig start.sh on 开机自启动
service start.sh start 启动服务

开机自启动会创建一个软连接，这个软连接，指向哪里，开机启动在那个目录

有时候我们需要Linux系统在开机的时候自动加载某些脚本或系统服务主要用三种方式进行这一操作：
第一种方式：ln -s 建立启动软连接
在Linux中有7种运行级别（可在/etc/inittab文件设置），每种运行级别分别对应着/etc/rc.d/rc[0~6].d这7个目录
ln -s   在/etc/rc.d/rc*.d目录中建立/etc/init.d/服务的软链接(*代表0～6七个运行级别之一）
这7个目录中，每个目录分别存放着对应运行级别加载时需要关闭或启动的服务
由详细信息可以知道，其实每个脚本文件都对应着/etc/init.d/目录下具体的服务
K开头的脚本文件代表运行级别加载时需要关闭的，S开头的代表需要执行

chkonfig   命令行运行级别设置
第二种方式：chkconfig
如果需要自启动某些服务，只需使用chkconfig 服务名 on即可，若想关闭，将on改为off
在默认情况下，chkconfig会自启动2345这四个级别，如果想自定义可以加上--level选项

上面我们先将sshd服务的所有启动级别关闭，然后使用--level选项启动自定义级别

Tips：--list选项可查看指定服务的启动状态，chkconfig不带任何选项则查看所有服务状态
第三种
ntsysv　　伪图形运行级别设置

介绍一下mysql主从复制

（mysql主节点和从节点的serverID不同）

在每个事物更新数据完成之前，主节点在二进制日志记录这些变化。写入二进制日志完成后，主节点通知存储引擎提交事务

从节点开启i/o线程，并在主节点上打开一个连接

Binlog日志 dump 线程从主节点的二进制日志中读取事件

mysql从节点的i/o线程接收bin.log重放到中继日志relay.log中；

sql线程从中继日志读取事件存放到数据库中

2.mysql有哪些优化?

（1）创建索引，提高查询效率（类比字典）

（2）事务日志优化

优化主和从节点服务器的性能

```
MariaDB [hellodb]> set global innodb_flush_log_at_trx_commit=2
MariaDB [hellodb]> set global sync_binlog=0
```

innodb_flush_log_at_trx_commit=2

```
innodb_flush_log_at_trx_commit=2    0|1|2
0每秒write os cache & flush到磁盘
1每次事务提交write os cache & flush到磁盘
2每次事务提交write os cache    每秒flush到磁盘
set global sync_binlog=0
1 此为默认值，日志缓冲区将写入日志文件，并在每次事务后执行刷新到磁盘。 这是完全遵守ACID特性
0 提交时没有写磁盘的操作；而是每秒执行一次将日志缓冲区的提交的事务写入刷新到磁盘。 这样可提供更好的性能，但服务器崩溃可能丢失最后一秒的事务
2 每次提交后都会写入OS的缓冲区，但每秒才会进行一次刷新到磁盘文件中。 性能比0略差一些，但操作系统或停电可能导致最后一秒的交易丢失
说明：
1.配置为2和配置为0，性能差异并不大，因为将数据从Log Buffer拷贝到OS cache，虽然跨越用户态与内核态，但毕竟只是内存的数据拷贝，速度很快
2.配置为2和配置为0，安全性差异巨大，操作系统崩溃的概率相比MySQL应用程序崩溃的概率，小很多，设置为2，只要操作系统不奔溃，也绝对不会丢数据
0是mysql的日志缓存区，而2是内存缓存区，操作系统相较于mysql还是比较稳定的
```

set global sync_binlog=0

```
sync_binlog=1|0：#设定是否启动二进制日志即时同步磁盘功能，默认0，由操作系统负责同步日志到磁盘
开启后表示最高级别容错，但是会影响性能
```

读/写分离优化

主库负责写，从库负责读；

mysql的备份用哪些参数，备份的命令是什么

```
格式：mysqldump -h主机名 -P端口 -u用户名 -p密码 --database 数据库名 > 文件名.sql

例如： mysqldump -h 10.0.0.100 -p 3306 -uroot -p123456 --database cmdb >
/data/backup/cmdb.sql
```

```
2、备份压缩      导出的数据有可能比较大，不好备份到远程，这时候就需要进行压缩
格式：mysqldump -h主机名 -P端口 -u用户名 -p密码 --database 数据库名 | gzip > 文件
名.sql.gz
例如： mysqldump -h10.0.0.100 -p 3306 -uroot -p123456 --database cmdb | gzip >
/data/backup/cmdb.sql.gz
```

tcpdump工具的使用

```
tcpdump -i eth0  抓取eth0网卡的包
tcpdump -i eth0 host 172.16.160.215 使用 host 参数指定监听本机与指定的主机之间（双向）的
通信包。不指定默认监听与所有主机的通信,172.16.160.215是目标主机。
tcpdump -i eth0 port 3306
上述语句监听本机 eth0网口与所有主机之间（双向）使用 3306 端口的通信包
tcpdump -i ens192 src port 3306
上述语句监听所有经过本机 ens192 网口并且发出方端口为 3306 的通信包。
tcpdump -i ens192 tcp
只监听 TCP 通信包。
tcpdump -i ens192 -w data.cap
使用 -w 指定输出文件
```

怎样将一个文件打成包

```
tar -cvf test-14-09-12.tar /home/lfqy/test
其中，-c选项代表创建新的tar文件(也就是不删除原文件)；-v表示显示创建的过程；-f表示指定新创建的
tar文件的名字，-f后面必须要紧跟文件名，因此，f选项放在各个选项的最后面。
```

怎样查看网络流量

```
ip -s -h link
```