Keepalived

讲师： 张士杰

http://www.magedu.com

本章内容

- 高可用集群
- KeepAlived 组成
- keepAlived 配置

- Keepalived 企业应用

高可用集群概念

- 集群类型：

  LB lvs/nginx（http/upstream, stream/upstream）

  HA 高可用性

  SPoF: Single Point of Failure，单点故障

  HPC 高性能集群(High Performance Computing) https://www.top500.org

- 系统可用性：SLA(Service-Level Agreement)

  95%=(60*24*30)*(1-0.9995)

  （指标）=99%, ..., 99.999%，99.9999%

- 系统故障：

  硬件故障：设计缺陷、wear out（损耗）、自然灾害……

  软件故障：设计缺陷

高可用集群概念

- 提升系统高用性的解决方案之降低MTTR(平均故障时间)

  解决方案：建立冗余机制

  active/passive　　主/备

  active/active　　双主

  active --> HEARTBEAT --> passive

  active <--> HEARTBEAT <--> active

- 高可用的是“服务”

  HA nginx service：

  　　vip/nginx process[/shared storage]

  资源：组成一个高可用服务的“组件”

  (1) passive node的数量

  (2) 资源切换

高可用集群概念

- shared storage：

NAS(Network Attached Storage)：网络附加存储，基于网络的共享文件系统。

SAN(Storage Area Network)：存储区域网络，基于网络的块级别的共享

- Network partition：网络分区

quorum：法定人数

with quorum：　> total/2

without quorum: <= total/2

隔离设备：fence

node：STONITH = Shooting The Other Node In The Head(强制下线/断电)

https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/7/html/high_availability_add-on_reference/s1-unfence-haar

高可用集群概念

- 双节点集群(TWO nodes Cluster)

  辅助设备：ping node, quorum disk(仲裁设备)

  - Failover：故障切换，即某资源的主节点故障时，将资源转移至其它节点的操作
  - Failback：故障移回，即某资源的主节点故障后重新修改上线后，将之前已转移至其它节点的资源重新切回的过程

- HA Cluster实现方案:

  AIS(Applicaiton Interface Specification)应用程序接口规范

  RHCS：Red Hat Cluster Suite红帽集群套件

  heartbeat：基于心跳监测实现服务高可用

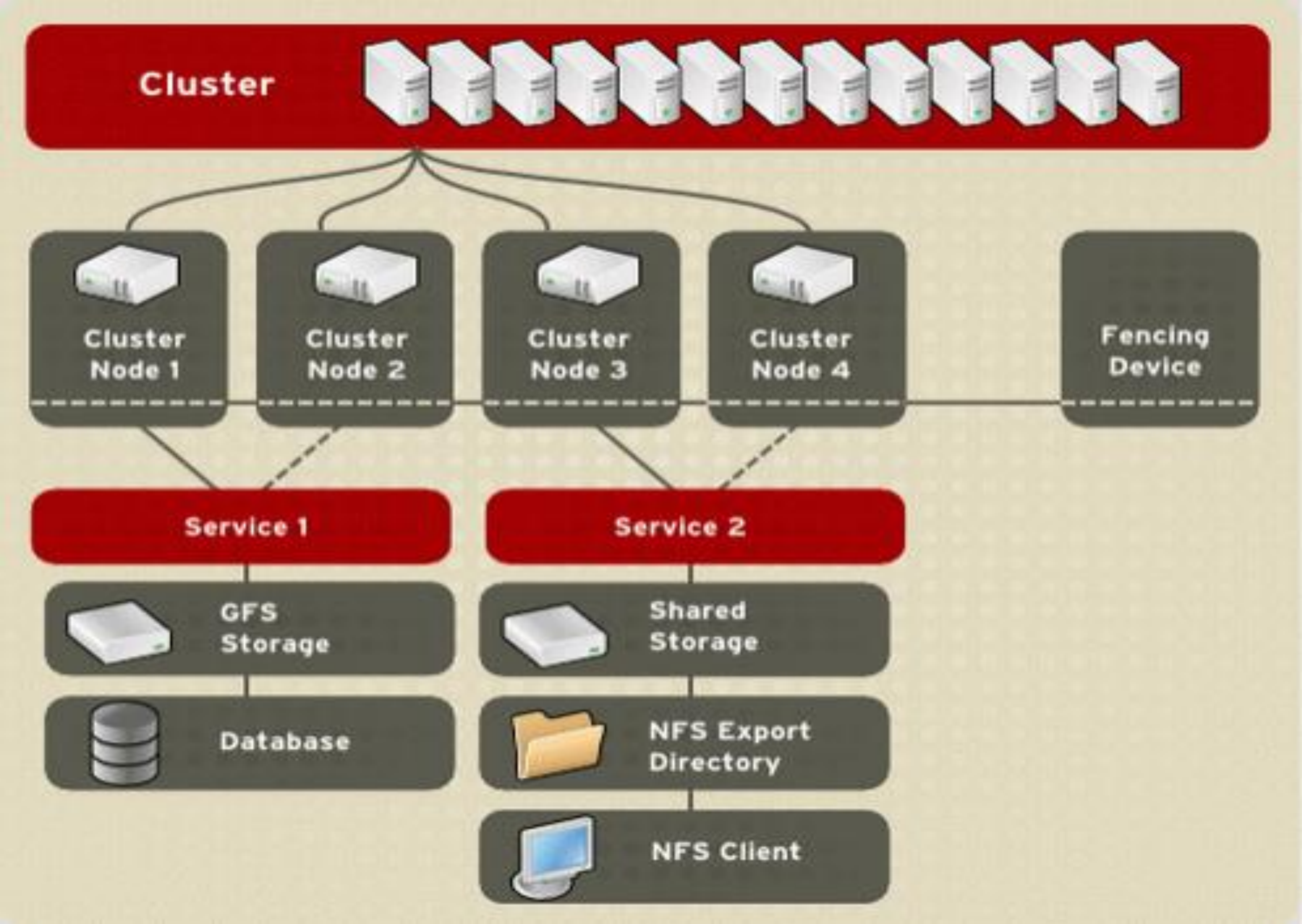
  pacemaker+corosync：资源管理与故障转移

  vrrp(Virtual Router Redundancy Protocol)：虚拟路由冗余协议,解决静态网关单点风险

  软件层—keepalived

  物理层—路由器、三层交换机

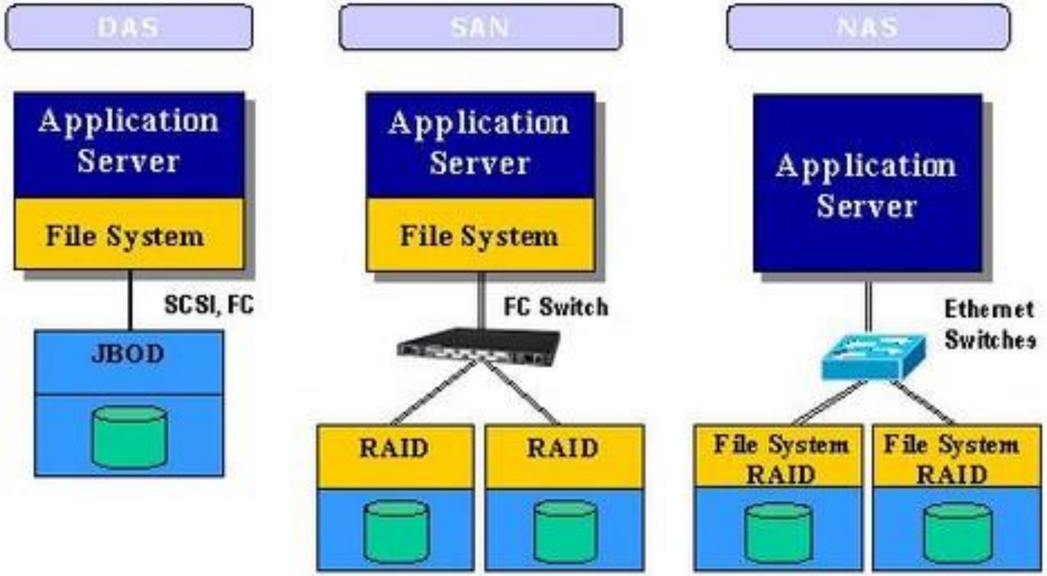
高可用集群-后端存储


Cluster
Cluster Node 1
Cluster Node 2
Cluster Node 3
Cluster Node 4
Fencing Device
Service 1
Service 2
GFS Storage
Shared Storage
Database
NFS Export Directory
NFS Client

高可用集群-后端存储

马哥教育
IT人的高薪职业学院

JBOD （Just a Bunch Of Disks）不是标准的 RAID 等级，它通常用来表示一个没有控制软件提供协调控制的磁盘集合，JBOD 将多个物理磁盘串联起来，提供一个巨大的逻辑磁盘，JBOD 的数据存放机制是由第一块磁盘开始按顺序往后存储，当前磁盘存储空间用完后，再依次往后面的磁盘存储数据，JBOD 存储性能完全等同于单块磁盘，而且也不提供数据安全保护，它只是简单提供一种扩展存储空间的机制，JBOD 可用存储容量等于所有成员磁盘的存储空间之和。

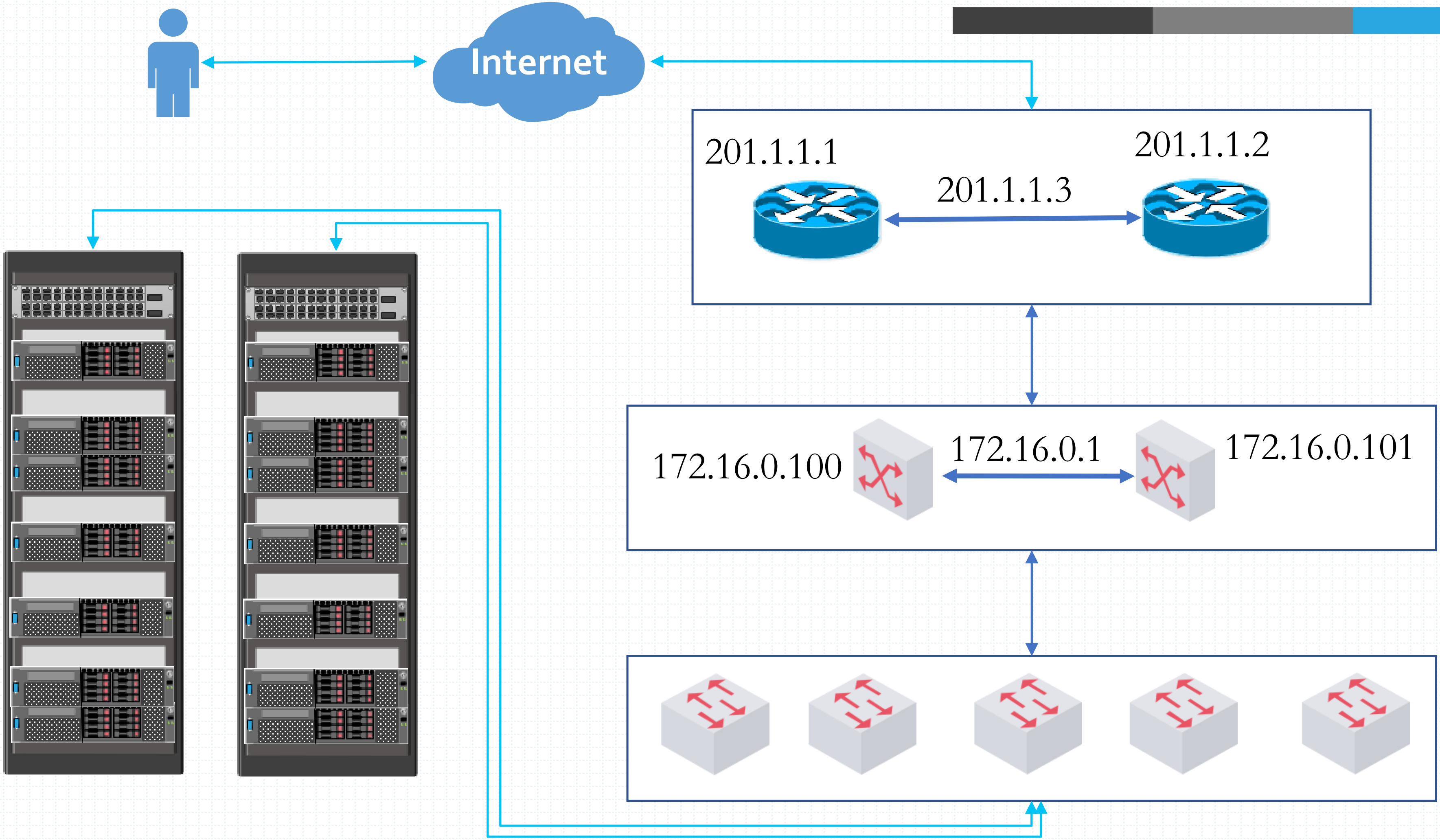


Keepalived简介

- keepalived:

  vrrp协议的软件实现，原生设计目的为了高可用ipvs服务
- 功能：
  - 基于vrrp协议完成地址流动
  - 为vip地址所在的节点生成ipvs规则(在配置文件中预先定义)
  - 为ipvs集群的各RS做健康状态检测
  - 基于脚本调用接口通过执行脚本完成脚本中定义的功能，进而影响集群事务，以此支持nginx、haproxy等服务

VRRP-网络层实现


Internet
201.1.1.1
201.1.1.2
201.1.1.3
172.16.0.100
172.16.0.1
172.16.0.101

Keepalived简介

- 组件：
  - 用户空间核心组件：

    vrrp stack-VIP消息通告

    checkers-监测real server

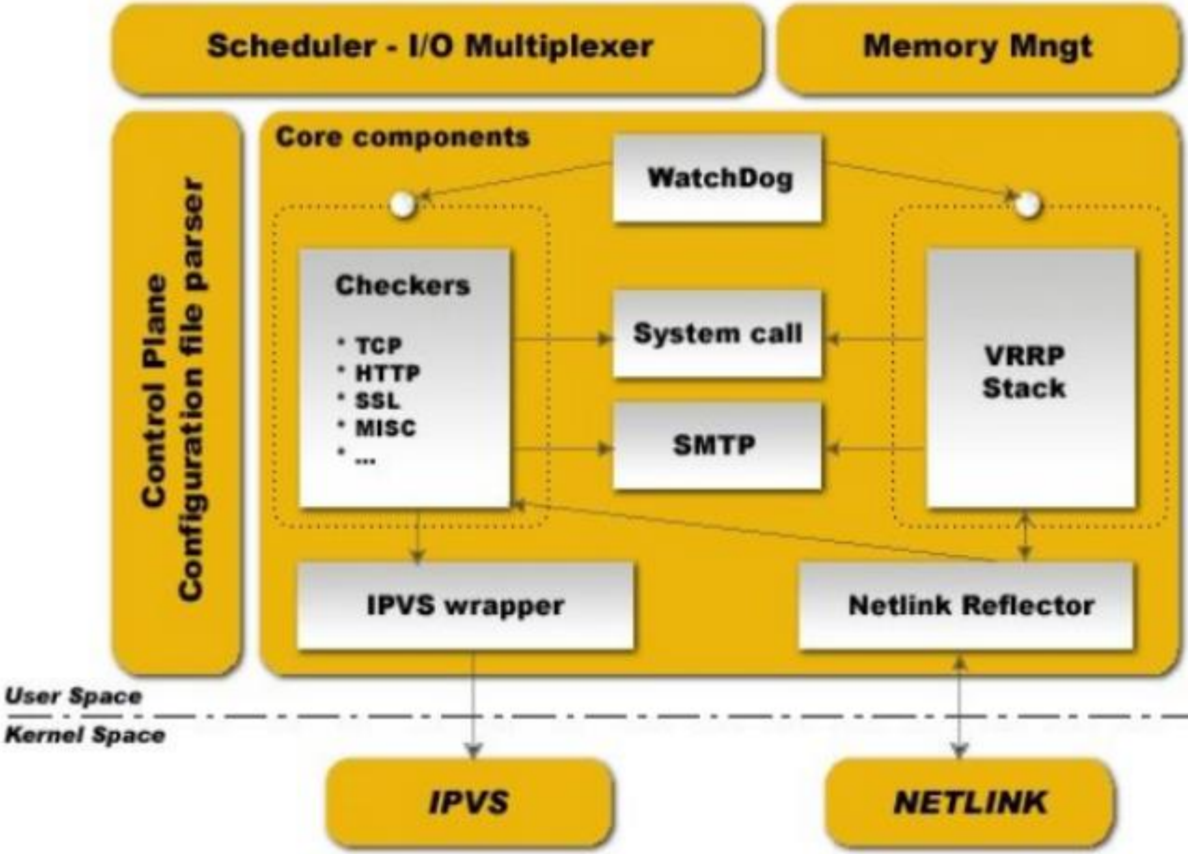
    system call-标记real server权重

    SMTP-邮件组件

    ipvs wrapper-生成IPVS规则

    Netlink Reflector-网络接口

    WatchDog-监控进程
  - 控制组件：配置文件分析器
  - IO复用器
  - 内存管理组件



- http://keepalived.org/documentation.html

Keepalived简介

- keepalived：

  vrrp协议：Virtual Router Redundancy Protocol
- 术语：

  虚拟路由器：Virtual Router

  虚拟路由器标识：VRID(0-255)，唯一标识虚拟路由器

  物理路由器：

  master：主设备

  backup：备用设备

  priority：优先级

  VIP：Virtual IP

  VMAC：Virutal MAC (00-00-5e-00-01-VRID)

Keepalived简介

- 通告：心跳，优先级等；周期性
- 工作方式：抢占式，非抢占式
- 安全工作：

  认证：

  无认证

  简单字符认证：预共享密钥

- 工作模式：

  主/备：单虚拟路由器

  主/主：主/备（虚拟路由器1），备/主（虚拟路由器2）

Keepalived环境准备

- 各节点时间必须同步
- 关闭selinux和防火墙

Keepalived安装

- Keepalived安装：

  # yum install keepalived (CentOS)

  # apt-get install keepalived (Ubuntu)

- 程序环境：
  - 主配置文件： /etc/keepalived/keepalived.conf
  - 主程序文件： /usr/sbin/keepalived
  - Unit File：
    - /usr/lib/systemd/system/keepalived.service (CentOS)
    - /lib/systemd/system/keepalived.service (Ubuntu)

  - Unit File的环境配置文件：
    - /etc/sysconfig/keepalived

KeepAlived配置

- 配置文件组件部分：
- TOP HIERACHY

  GLOBAL CONFIGURATION

  Global definitions

  VRRP CONFIGURATION

  VRRP instance(s)：即一个vrrp虚拟路由器

  LVS CONFIGURATION

  Virtual server group(s)

  Virtual server(s)：ipvs集群的vs和rs

Keepalived配置

- 配置语法：
- 配置虚拟路由器：

```
vrrp_instance <STRING> {
        ....
}
```

- 配置参数：

state MASTER|BACKUP：当前节点在此虚拟路由器上的初始状态，状态为MASTER或者BACKUP

interface IFACE_NAME：绑定为当前虚拟路由器使用的物理接口 ens32,eth0,bond0,br0

virtual_router_id VRID：当前虚拟路由器惟一标识，范围是0-255

priority 100：当前物理节点在此虚拟路由器中的优先级；范围1-254

advert_int 1：vrrp通告的时间间隔，默认1s

Keepalived配置

```
authentication { #认证机制
        auth_type AH|PASS
        auth_pass <PASSWORD>  仅前8位有效
}
virtual_ipaddress { #虚拟IP
        <IPADDR>/<MASK> brd <IPADDR> dev <STRING> scope <SCOPE> label <LABEL>
        192.168.200.17/24 dev eth1
        192.168.200.18/24 dev eth2 label eth2:1
}
track_interface { #配置监控网络接口，一旦出现故障，则转为FAULT状态实现地址转移
        eth0
        eth1
        …
}
```

组播配置示例-MASTER(上):

```
global_defs {
   notification_email {
     root@localhost #keepalived 发生故障切换时邮件发送的对象，可以按行区分写多个
   }
   notification_email_from keepalived@localhost
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id ha1.example.com
   vrrp_skip_check_adv_addr #所有报文都检查比较消耗性能，此配置为如果收到的报文和上一个报文是同一个路由器则跳过检查报文中的源地址
   vrrp_strict #严格遵守VRRP协议,不允许状况:1,没有VIP地址,2.单播邻居,3.在VRRP版本2中有IPv6地址.
   vrrp_garp_interval 0 #ARP报文发送延迟
   vrrp_gna_interval 0 #消息发送延迟
   vrrp_mcast_group4 224.0.0.18 #默认组播IP地址，224.0.0.0到239.255.255.255
   #vrrp_iptables
}
```

组播配置示例-MASTER(下):

```
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 80
    priority 100
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1111qwer
    }
    virtual_ipaddress {
        192.168.7.248 dev eth0 label eth0:0
    }
}
```

组播配置示例-BACKUP(上):

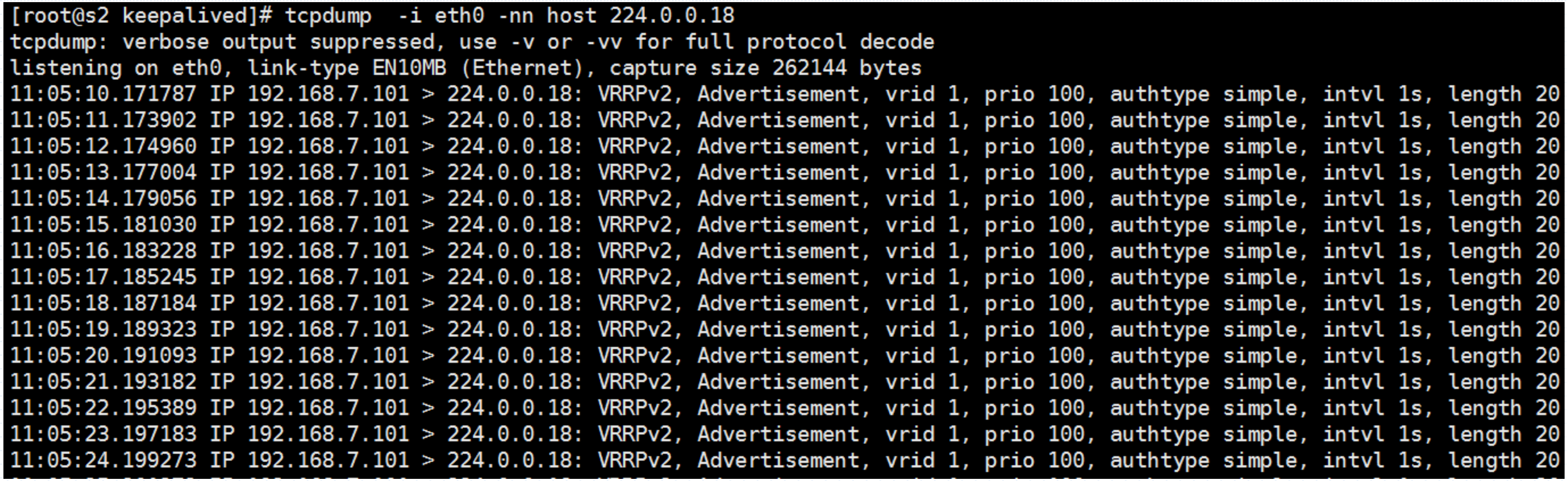
```
global_defs {
   notification_email {
     root@localhost
   }
   notification_email_from keepalived@localhost
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id ha2.example.com
   vrrp_skip_check_adv_addr #
   vrrp_strict #严格遵守VRRP协议。
   vrrp_garp_interval 0 #ARP报文发送延迟
   vrrp_gna_interval 0 #消息发送延迟
   vrrp_mcast_group4 224.0.0.18 #组播IP地址，224.0.0.0到239.255.255.255
   #vrrp_iptables
}
```

组播配置示例-BACKUP(下):

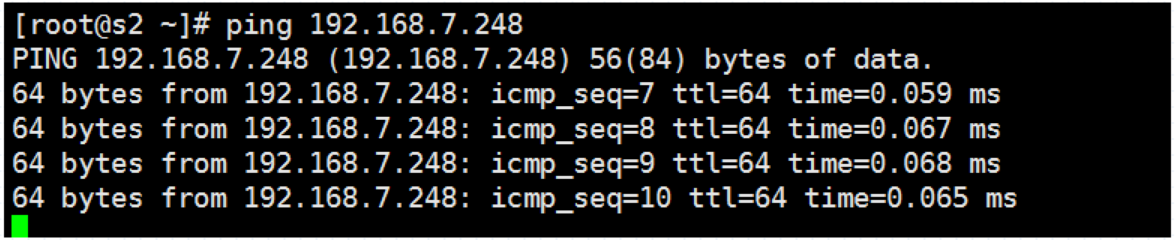
```
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 80
    priority 90
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1111qwer
    }
    virtual_ipaddress {
        192.168.7.248 dev eth0 label eth0:0
    }
}
```

VIP 测试

- # iptables -D INPUT   -s 0.0.0.0/0 -d 192.168.7.248 -j DROP #yum安装会自动生成防火墙策略，可以删除或禁止生成



```
[root@s2 keepalived]# tcpdump  -i eth0 -nn host 224.0.0.18
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:05:10.171787 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:11.173902 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:12.174960 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:13.177004 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:14.179056 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:15.181030 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:16.183228 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:17.185245 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:18.187184 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:19.189323 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:20.191093 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:21.193182 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:22.195389 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:23.197183 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
11:05:24.199273 IP 192.168.7.101 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio 100, authtype simple, intvl 1s, length 20
```



```
[root@s2 ~]# ping 192.168.7.248
PING 192.168.7.248 (192.168.7.248) 56(84) bytes of data.
64 bytes from 192.168.7.248: icmp_seq=7 ttl=64 time=0.059 ms
64 bytes from 192.168.7.248: icmp_seq=8 ttl=64 time=0.067 ms
64 bytes from 192.168.7.248: icmp_seq=9 ttl=64 time=0.068 ms
64 bytes from 192.168.7.248: icmp_seq=10 ttl=64 time=0.065 ms
```

非抢占

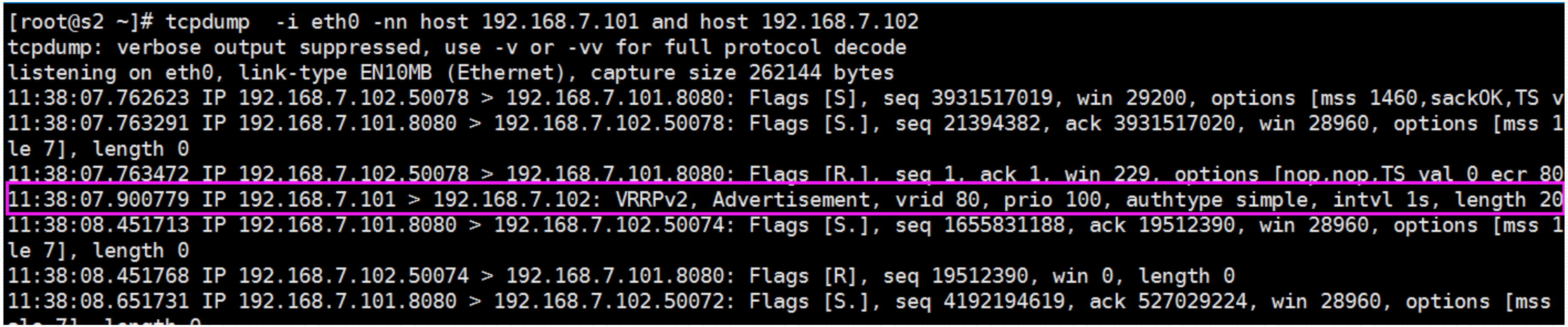
- nopreempt #关闭VIP抢占，需要VIP state都为BACKUP
  - vrrp_instance VI_1 {
  - state **BACKUP**
  - interface eth0
  - virtual_router_id 80
  - priority 100
  - advert_int 1
  - nopreempt

  - vrrp_instance VI_1 {
  - state **BACKUP**
  - interface eth0
  - virtual_router_id 80
  - priority 90
  - advert_int 1
  - nopreempt

VIP单播配置及示例

- unicast_src_ip 本机源IP
-     unicast_peer{
-         目标主机IP
-     }

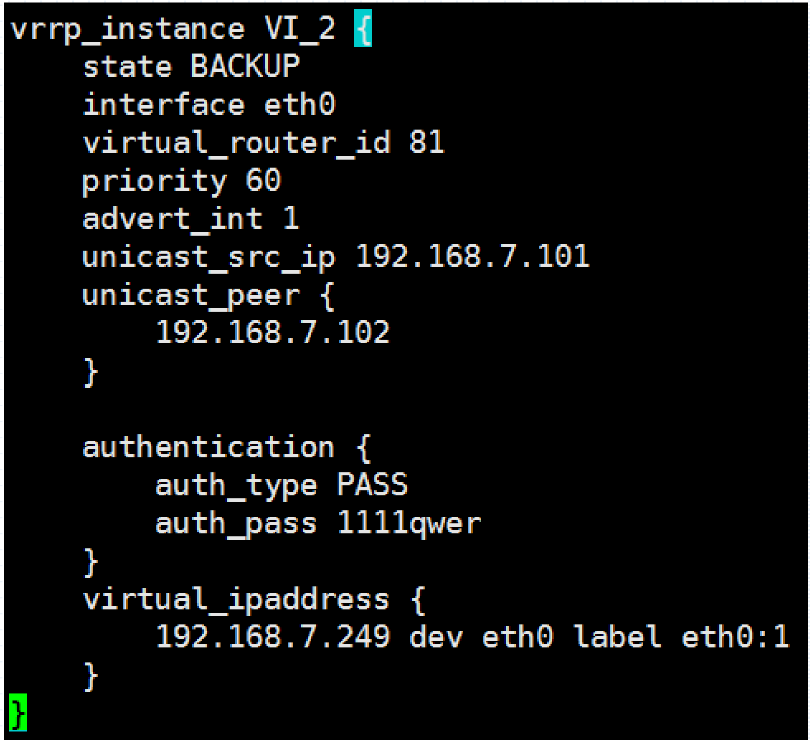
- [root@s2 ~]# tcpdump -i eth0 host -nn 172.18.200.101 and host 172.18.200.102



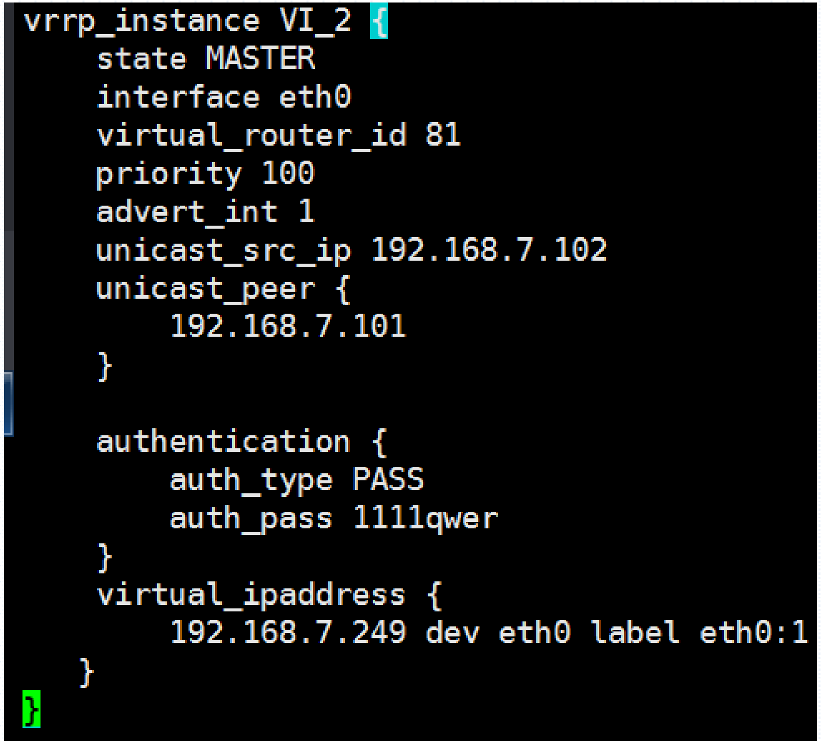
```
[root@s2 ~]# tcpdump  -i eth0 -nn host 192.168.7.101 and host 192.168.7.102
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:38:07.762623 IP 192.168.7.102.50078 > 192.168.7.101.8080: Flags [S], seq 3931517019, win 29200, options [mss 1460,sackOK,TS v
11:38:07.763291 IP 192.168.7.101.8080 > 192.168.7.102.50078: Flags [S.], seq 21394382, ack 3931517020, win 28960, options [mss 1
le 7], length 0
11:38:07.763472 IP 192.168.7.102.50078 > 192.168.7.101.8080: Flags [R.], seq 1, ack 1, win 229, options [nop,nop,TS val 0 ecr 80
11:38:07.900779 IP 192.168.7.101 > 192.168.7.102: VRRPv2, Advertisement, vrid 80, prio 100, authtype simple, intvl 1s, length 20
11:38:08.451713 IP 192.168.7.101.8080 > 192.168.7.102.50074: Flags [S.], seq 1655831188, ack 19512390, win 28960, options [mss 1
le 7], length 0
11:38:08.451768 IP 192.168.7.102.50074 > 192.168.7.101.8080: Flags [R], seq 19512390, win 0, length 0
11:38:08.651731 IP 192.168.7.101.8080 > 192.168.7.102.50072: Flags [S.], seq 4192194619, ack 527029224, win 28960, options [mss
ale 7], length 0
```

Keepalivde 双主配置

- 两个或以上VIP分别运行在不同的keepalived服务器，以实现服务器并行提供web访问的目的，提高服务器资源利用率。



```
vrrp_instance VI_2 {
    state BACKUP
    interface eth0
    virtual_router_id 81
    priority 60
    advert_int 1
    unicast_src_ip 192.168.7.101
    unicast_peer {
        192.168.7.102
    }

    authentication {
        auth_type PASS
        auth_pass 1111qwer
    }
    virtual_ipaddress {
        192.168.7.249 dev eth0 label eth0:1
    }
}
```



```
vrrp_instance VI_2 {
    state MASTER
    interface eth0
    virtual_router_id 81
    priority 100
    advert_int 1
    unicast_src_ip 192.168.7.102
    unicast_peer {
        192.168.7.101
    }

    authentication {
        auth_type PASS
        auth_pass 1111qwer
    }
    virtual_ipaddress {
        192.168.7.249 dev eth0 label eth0:1
   }
}
```

Keepalived通知配置

发件人配置：

```
[root@s2 ~]# vim /etc/mail.rc

set from=2973707860@qq.com
set smtp=smtp.qq.com
set smtp-auth-user=2973707860@qq.com
set smtp-auth-password=mfcjxxjezawgdgee
set smtp-auth=login
set ssl-verify=ignore
```

Keepalived通知配置

- nopreempt：定义工作模式为非抢占模式
- preempt_delay 300：抢占式模式，节点上线后触发新选举操作的延迟时长，默认模式
- 定义通知脚本：

notify_master <STRING>|<QUOTED-STRING>：

当前节点成为主节点时触发的脚本

notify_backup <STRING>|<QUOTED-STRING>：

当前节点转为备节点时触发的脚本

notify_fault <STRING>|<QUOTED-STRING>：

当前节点转为“失败”状态时触发的脚本

notify <STRING>|<QUOTED-STRING>：

通用格式的通知触发机制，一个脚本可完成以上三种状态的转换时的通知

Keepalived通知脚本

```
[root@localhost keepalived]# cat /etc/keepalived/notify.sh
#!/bin/bash
contact='2973707860@qq.com'
notify() {
 mailsubject="$(hostname) to be $1, vip 转移"
 mailbody="$(date +'%F %T'): vrrp transition, $(hostname) changed to be $1"
 echo "$mailbody" | mail -s "$mailsubject" $contact
}
case $1 in
master)
 notify master
 ;;
backup)
 notify backup
 ;;
fault)
 notify fault
 ;;
*)
 echo "Usage: $(basename $0) {master|backup|fault}"
 exit 1
 ;;
esac
```

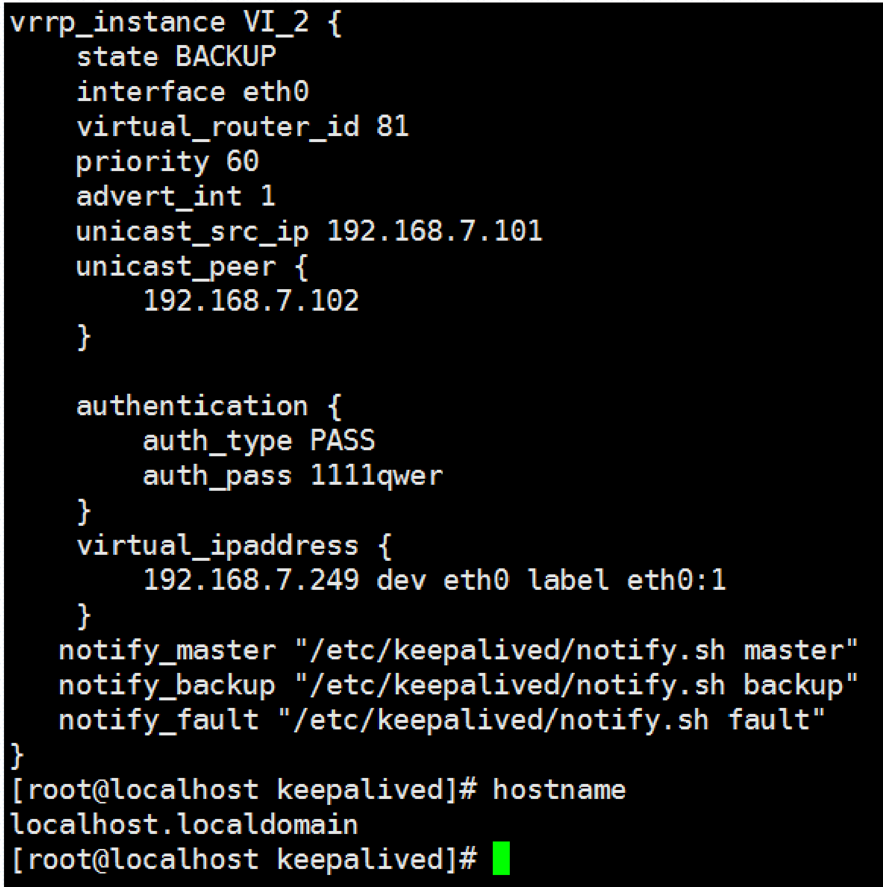
Keepalived通知配置

- 脚本的调用方法：

  notify_master "/etc/keepalived/notify.sh master"

  notify_backup "/etc/keepalived/notify.sh backup"

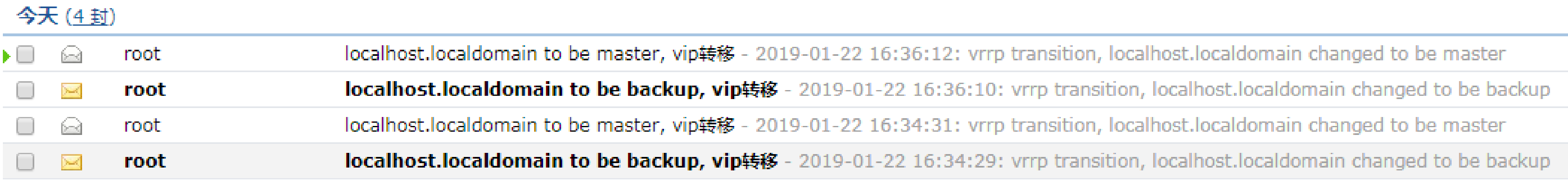
  notify_fault "/etc/keepalived/notify.sh fault"



```
vrrp_instance VI_2 {
    state BACKUP
    interface eth0
    virtual_router_id 81
    priority 60
    advert_int 1
    unicast_src_ip 192.168.7.101
    unicast_peer {
        192.168.7.102
    }

    authentication {
        auth_type PASS
        auth_pass 1111qwer
    }
    virtual_ipaddress {
        192.168.7.249 dev eth0 label eth0:1
    }
   notify_master "/etc/keepalived/notify.sh master"
   notify_backup "/etc/keepalived/notify.sh backup"
   notify_fault "/etc/keepalived/notify.sh fault"
}
[root@localhost keepalived]# hostname
localhost.localdomain
[root@localhost keepalived]# 
```

Keepalived通知验证

- 停止keepalived服务，验证IP 切换后是否收到通知邮件：



今天 (4 封)

root　localhost.localdomain to be master, vip转移 - 2019-01-22 16:36:12: vrrp transition, localhost.localdomain changed to be master

root　**localhost.localdomain to be backup, vip转移** - 2019-01-22 16:36:10: vrrp transition, localhost.localdomain changed to be backup

root　localhost.localdomain to be master, vip转移 - 2019-01-22 16:34:31: vrrp transition, localhost.localdomain changed to be master

root　**localhost.localdomain to be backup, vip转移** - 2019-01-22 16:34:29: vrrp transition, localhost.localdomain changed to be backup

KeepAlived与IPVS

虚拟服务器配置参数：

virtual server （虚拟服务）的定义：

virtual_server IP port #定义虚拟主机IP地址及其端口

virtual_server fwmark int #ipvs的防火墙打标，实现基于防火墙的负载均衡集群

virtual_server group string #将多个虚拟服务器定义成组，将组定义成虚拟服务

```
virtual_server IP port
{
      …
      real_server {
      …
      }
      …
}
```

KeepAlived与IPVS

delay_loop <INT>：检查后端服务器的时间间隔

lb_algo rr|wrr|lc|wlc|lblc|sh|dh：定义调度方法

lb_kind NAT|DR|TUN：集群的类型

persistence_timeout <INT>：持久连接时长

protocol TCP|UDP|SCTP：指定服务协议

sorry_server <IPADDR> <PORT>：所有RS故障时，备用服务器地址

real_server <IPADDR> <PORT>

```
 {
   weight <INT>    RS权重
   notify_up <STRING>|<QUOTED-STRING>    RS上线通知脚本
   notify_down <STRING>|<QUOTED-STRING> RS下线通知脚本
   HTTP_GET|SSL_GET|TCP_CHECK|SMTP_CHECK|MISC_CHEC K { ... }：定义当前主机的健康状态检测方法
 }
```

应用层监测

- HTTP_GET|SSL_GET：应用层检测

```
HTTP_GET|SSL_GET {
        url {
            path <URL_PATH>：定义要监控的URL
            status_code <INT>：判断上述检测机制为健康状态的响应码
        }
connect_timeout <INTEGER>：连接请求的超时时长
nb_get_retry  <INT>：重试次数
delay_before_retry <INT>：重试之前的延迟时长
connect_ip <IP ADDRESS>：向当前RS哪个IP地址发起健康状态检测请求
connect_port <PORT>：向当前RS的哪个PORT发起健康状态检测请求
bindto <IP ADDRESS>：发出健康状态检测请求时使用的源地址
bind_port <PORT>：发出健康状态检测请求时使用的源端口
}
```

TCP监测

- 传输层检测 TCP_CHECK

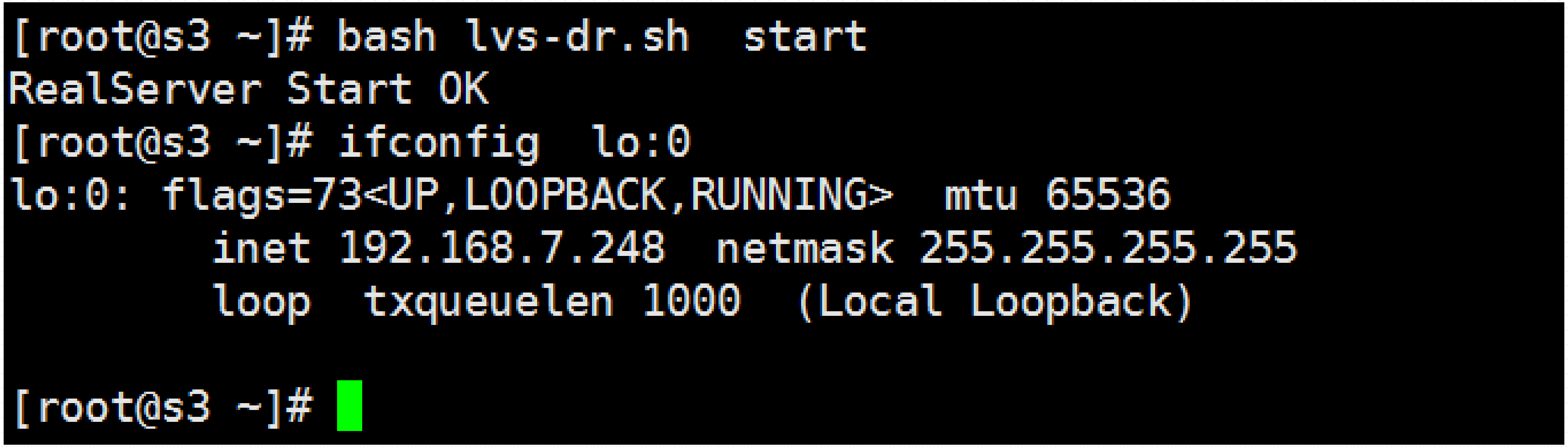
```
TCP_CHECK {
  connect_ip <IP ADDRESS>：向当前RS的哪个IP地址发起健康状态检测请求
  connect_port <PORT>：向当前RS的哪个PORT发起健康状态检测请求
  bindto <IP ADDRESS>：发出健康状态检测请求时使用的源地址
  bind_port <PORT>：发出健康状态检测请求时使用的源端口
  connect_timeout <INTEGER>：连接请求的超时时长
   }
```

Keepalived案例一：实现LVS-DR模式

规划图

Keepalived案例一： 实现LVS-DR模式

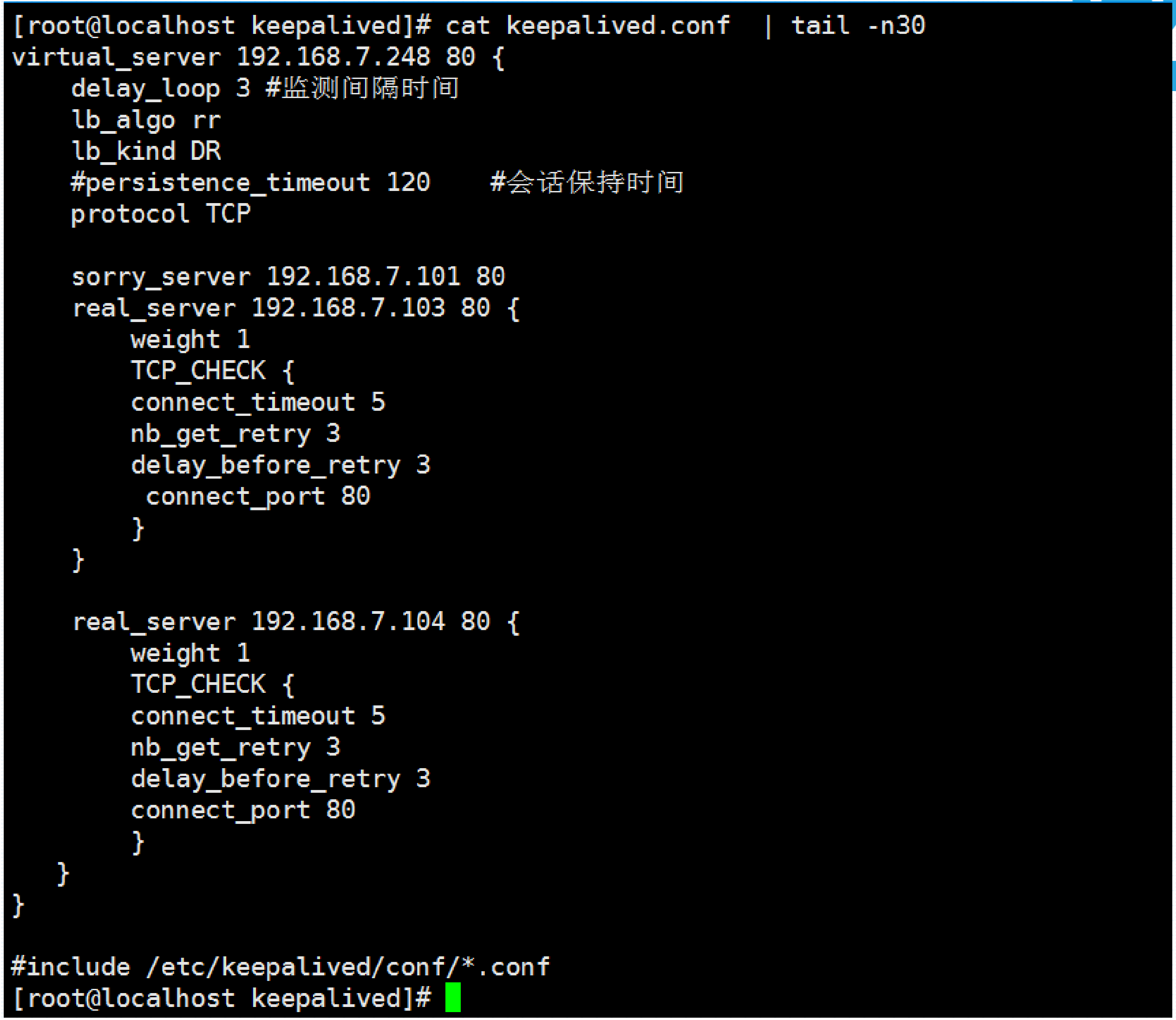
- 准备web服务器并使用脚本绑定VIP至web服务器lo网卡



```
[root@s3 ~]# bash lvs-dr.sh  start
RealServer Start OK
[root@s3 ~]# ifconfig  lo:0
lo:0: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 192.168.7.248  netmask 255.255.255.255
        loop  txqueuelen 1000  (Local Loopback)

[root@s3 ~]# 
```

Keepalived案例一： 实现LVS-DR模式

配置keepalived



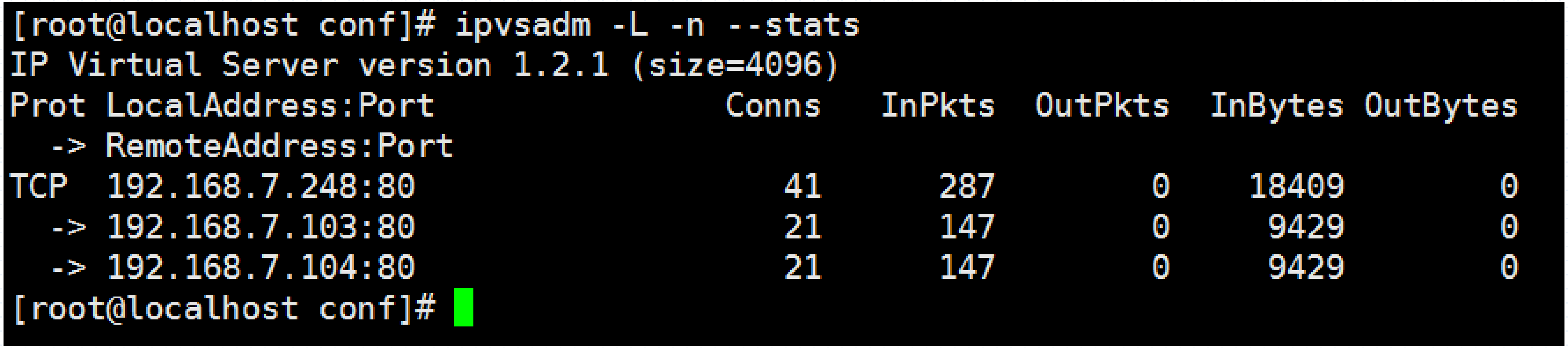
```
[root@localhost keepalived]# cat keepalived.conf  | tail -n30
virtual_server 192.168.7.248 80 {
    delay_loop 3 #监测间隔时间
    lb_algo rr
    lb_kind DR
    #persistence_timeout 120    #会话保持时间
    protocol TCP

    sorry_server 192.168.7.101 80
    real_server 192.168.7.103 80 {
        weight 1
        TCP_CHECK {
        connect_timeout 5
        nb_get_retry 3
        delay_before_retry 3
         connect_port 80
        }
    }

    real_server 192.168.7.104 80 {
        weight 1
        TCP_CHECK {
        connect_timeout 5
        nb_get_retry 3
        delay_before_retry 3
        connect_port 80
        }
   }
}

#include /etc/keepalived/conf/*.conf
[root@localhost keepalived]# 
```

Keepalived案例一： 实现LVS-DR模式

```
virtual_server 172.18.200.248 80 {
    delay_loop 6
    lb_algo wrr
    lb_kind DR
    #persistence_timeout 120    #会话保持时间
    protocol TCP
    sorry_server 172.18.200.105 80
    real_server 172.18.200.103 80 {
        weight 1
        TCP_CHECK {
        connect_timeout 5
        nb_get_retry 3
        delay_before_retry 3
         connect_port 80
        } }
    real_server 172.18.200.104 80 {
        weight 1
        TCP_CHECK {
        connect_timeout 5
        nb_get_retry 3
        delay_before_retry 3
         connect_port 80
        } } }
```

Keepalived案例一： 测试web访问

- # while true;do curl http://192.168.7.248 && sleep 1;done



```
[root@localhost conf]# ipvsadm -L -n --stats
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
TCP  192.168.7.248:80                   41      287        0    18409        0
  -> 192.168.7.103:80                   21      147        0     9429        0
  -> 192.168.7.104:80                   21      147        0     9429        0
[root@localhost conf]# 
```

real_server http监测

```
real_server 192.168.7.103 80 {
       weight 1
       HTTP_GET {
        url {
             path /index.html
             status_code 200
             }
       }
       connect_timeout 5
       nb_get_retry 3
       delay_before_retry 3
   }
```

VRRP script

- keepalived调用外部的辅助脚本进行资源监控，并根据监控的结果状态能实现优先动态调整
- vrrp_script: 自定义资源监控脚本，vrrp实例根据脚本返回值进行下一步操作，脚本可被多个实例调用。
  track_script: 调用vrrp_script定义的脚本去监控资源，定义在实例之内，调用事先定义的vrrp_script
- 分两步：(1) 先定义一个脚本；(2) 调用此脚本

```
vrrp_script <SCRIPT_NAME> {
    script <STRING>|<QUOTED-STRING>
    OPTIONS
}
track_script {
    SCRIPT_NAME_1
    SCRIPT_NAME_2
}
```

VRRP script

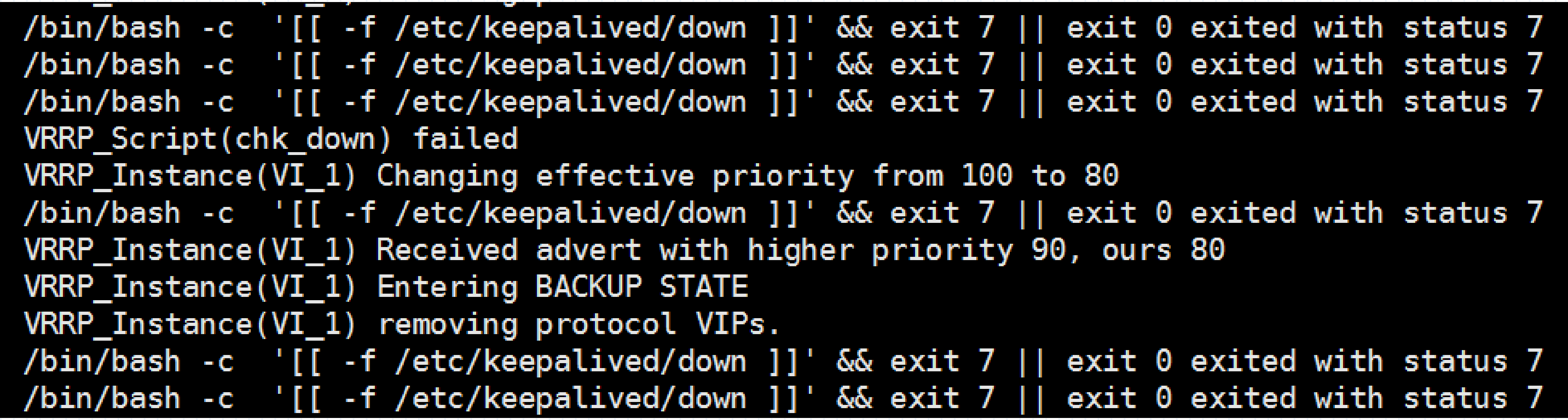
```
vrrp_script <SCRIPT_NAME> { #定义一个检测脚本，在global_defs 之外配置
        script <STRING>|<QUOTED-STRING> # shell命令或脚本路径
        interval <INTEGER>  # 间隔时间，单位为秒，默认1秒
        timeout <INTEGER>   # 超时时间
        weight <INTEGER:-254..254> # 权重，监测失败后会执行权重+操作
        fall <INTEGER>             #脚本几次失败转换为失败
        rise <INTEGER>             # 脚本连续监测成果后，把服务器从失败标记为成功的次数
         user USERNAME [GROUPNAME]  # 执行监测的用户或组
        init_fail              # 设置默认标记为失败状态，监测成功之后再转换为成功状态
      }
```

VRRP script

```
vrrp_script chk_down { #基于第三方仲裁设备
        script "/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0"
        interval 1
        weight -80
        fall 3
        rise 5
        timeout 2
}

vrrp_instance VI_1 {
…
 track_script {
    chk_down
   }
}
```

VRRP script

# touch /etc/keepalived/down



```
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
VRRP_Script(chk_down) failed
VRRP_Instance(VI_1) Changing effective priority from 100 to 80
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
VRRP_Instance(VI_1) Received advert with higher priority 90, ours 80
VRRP_Instance(VI_1) Entering BACKUP STATE
VRRP_Instance(VI_1) removing protocol VIPs.
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
/bin/bash -c  '[[ -f /etc/keepalived/down ]]' && exit 7 || exit 0 exited with status 7
```

高可用HAProxy

```
vrrp_script chk_haproxy {
        script "/etc/keepalived/chk_haproxy.sh"
        interval 1
        weight -80
        fall 3
        rise 5
        timeout 2
}

 track_script {
    chk_haproxy
   }


[root@s1 ~]# yum install psmisc -y
[root@s1 ~]# cat /etc/keepalived/chk_haproxy.sh
#!/bin/bash
/usr/bin/killall -0 haproxy
[root@s1 ~]# chmod a+x  /etc/keepalived/chk_haproxy.sh
```

高可用Nginx

```
vrrp_script chk_nginx {
        script "/etc/keepalived/chk_nginx.sh"
        interval 1
        weight -80
        fall 3
        rise 5
        timeout 2
}

 track_script {
    chk_haproxy
   }
```

```
[root@s1 ~]# yum install psmisc -y
[root@s1 ~]# cat /etc/keepalived/chk_nginx.sh
#!/bin/bash
/usr/bin/killall -0 nginx
[root@s1 ~]# chmod a+x  /etc/keepalived/chk_nginx.sh
```

作业

- keepalived 高可用性IPVS集群，IPVS集群提供php，如phpwind
- Keepalived 双主Nginx或者HAProxy
- vrrp_script 高可用性Nginx

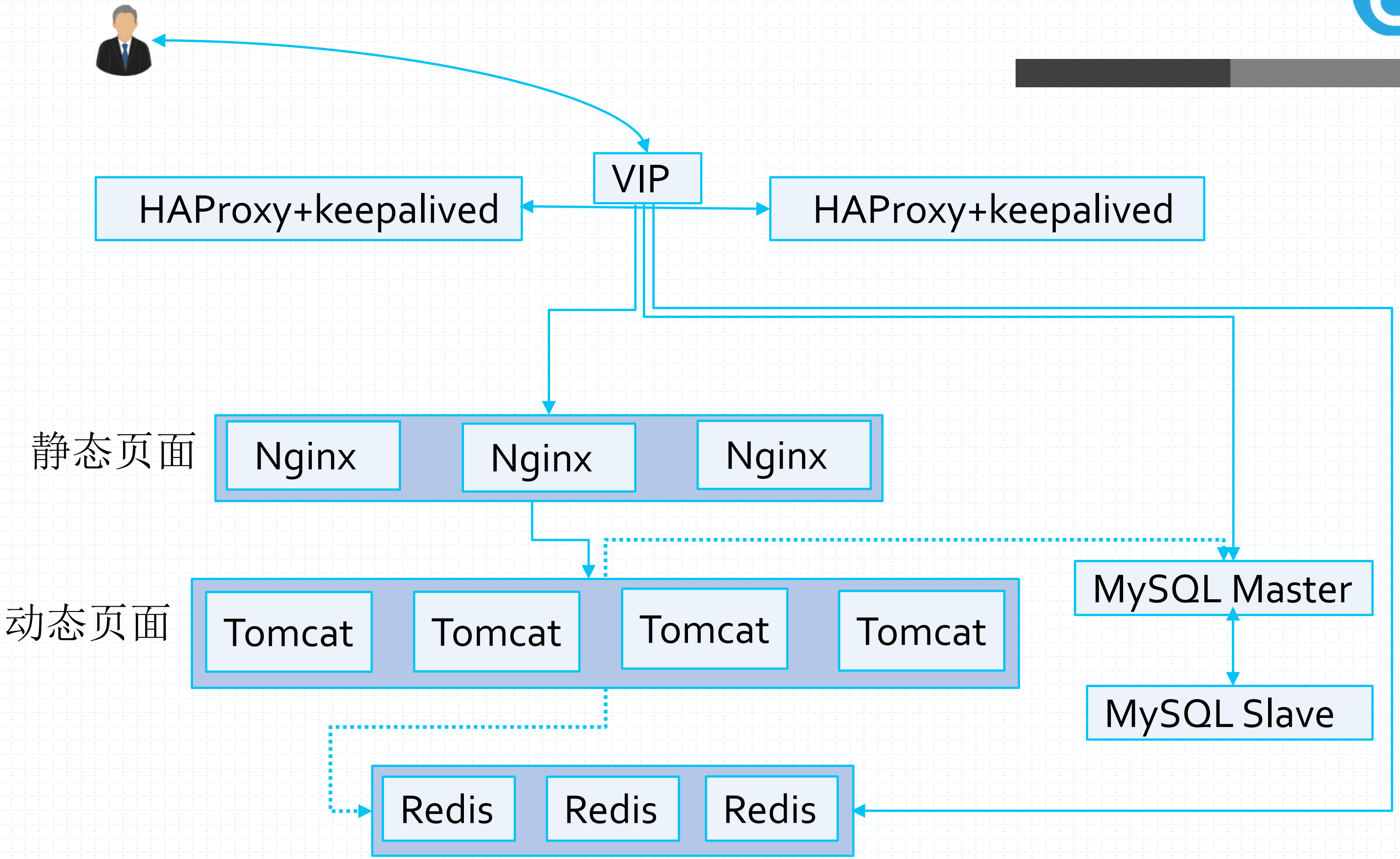
实战案例-1

- 1、 编译安装keepalived。
- 2、 实现keepalived(20+ vip)+nginx 双主高可用。
- 3、 实现keepalived(60+ vip)+haproxy 三服务器高可用。
- 4、 实现keepalived(100+ vip)+LVS高可用、real server状态监测及规则管理。

HAProxy+keepalived实战案例-2

- 1.编译安装HAProxy较新 LTS版本.,选择编译安装keepalived。
- 2.开启HAProxy多进程，进程数与CPU核心数保持一致并实现HAProxy进程绑定。
- 3.因业务较多避免配置文件误操作，需要按每业务一个配置文件并统一保存至/etc/haproxy/conf 目录中。
- 4.实现keepalived include导入配置文件功能，使用LVS-DR模型代理后端Nginxweb服务器。
- 5.基于ACL实现单IP多域名负载功能(适用于企业较少公网IP多域名场景)。
- 6.实现MySQL主从复制，并通过HAProxy对MySQL进行反向代理。
- 7.最终完成HAProxy+Nginx+Tomcat+ Redis，并实现session会话保持统一保存到Redis。

HAProxy+keepalived实战案例



关于马哥教育

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：1661815153, 113228115
- QQ群：203585050, 279599283

Thank You!

讲师： 张士杰（杰哥）

http://www.magedu.com