нетология

Компьютерные сети

лекция 3

Андрей
Вахутинский

Андрей Вахутинский

Зам.начальника ИТ отдела в АО "ИНТЕКО"

Андрей Вахутинский

План модуля

1. Работа в терминале, лекция 1
2. Работа в терминале, лекция 2
3. Операционные системы, лекция 1
4. Операционные системы, лекция 2
5. Файловые системы
6. Компьютерные сети, лекция 1
7. Компьютерные сети, лекция 2
8. **Компьютерные сети, лекция 3**
9. Элементы безопасности информационных систем

План занятия

1. Сетевые неймспейсы (пространства имен)
2. ipvs (LVS) + keepalived (VRRP)
3. Итоги
4. Домашнее задание

Сетевые неймспейсы (пространства имен)

ip, netns – инструмент управления сетевыми неймспейсами

Современное ядро Linux поддерживает изоляцию многих своих подсистем. Среди них – сетевая подсистема, это распространенная практика для приложений в контейнерах.
При старте контейнеров от пользователя все скрыто, но на самом деле происходит следующее:

```
root@netology1:~# ip netns add demo_app
root@netology1:~# ip netns list # lsns -t net не покажет т.к. нет процессов
demo_app
root@netology1:~# ip netns exec demo_app ip link list # nsenter тоже, нет процессов
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
root@netology1:~# ls /var/run/netns
demo_app
```

При создании неймспейса руками, в нем присутствует только loopback, причем он не активен – сеть в неймспейсе вообще не работает. Один из способов поднять сеть в ns – создать пару Virtual Ethernet устройств, добавив одно из них в неймспейс:

```
root@netology1:~# ip link add veth0 type veth peer name veth1 # создаем пару
root@netology1:~# ip link set veth1 netns demo_app # добавляем один veth в ns
root@netology1:~# ip netns exec demo_app ip link set veth1 up # поднимаем интерфейс ns
root@netology1:~# ip link set veth0 up # поднимаем парный ему интерфейс на хосте
root@netology1:~# ip netns exec demo_app ip link list | grep veth
9: veth1@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode
DEFAULT group default qlen 1000
```

ip, netns – назначение адресов

Результат работы данной последовательности команд можно сравнить с двумя хостами, соединенными напрямую витой парой: линк установлен, связность по на уровне L2 есть. Осталось назначить адреса устройству на хосте (veth0) и в неймспейсе (veth1):

```
root@netology1:~# ip addr add 169.254.1.1/30 dev veth0
root@netology1:~# ip netns exec demo_app ip addr add 169.254.1.2/30 dev veth1
root@netology1:~# ping -c1 169.254.1.2
PING 169.254.1.2 (169.254.1.2) 56(84) bytes of data.
64 bytes from 169.254.1.2: icmp_seq=1 ttl=64 time=0.061 ms …
root@netology1:~# ip netns exec demo_app ping -c1 169.254.1.1
PING 169.254.1.1 (169.254.1.1) 56(84) bytes of data.
64 bytes from 169.254.1.1: icmp_seq=1 ttl=64 time=0.117 ms ...
```

Обратите внимание, что в соответствии со стандартом мы выбрали подсеть в диапазоне из link-local для виртуальных устройств.

Необходимо понимать, что пара Virtual Ethernet – только один из возможных вариантов, который, например, не предусматривает взаимодействие нескольких контейнеров между собой на канальном уровне. Проблем организовать взаимодействие как таковое нет, но тогда хост-система будет вынуждена выступать L3 роутером, что избыточно.

Поэтому, например, Docker вместо Virtual Ethernet пары создает стандартный Linux bridge с помощью **`brctl`**, добавляя в него простые (непарные) виртуальные интерфейсы.

Пример: добавление внешней связности

Маршрут по-умолчанию в namespace:

```
root@netology1:~# ip netns exec demo_app ip route add default via 169.254.1.1
```

Разрешить хосту пересылать пакеты между разными L3 сетями:

```
root@netology1:~# sysctl  -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Форвардинг пакетов между интерфейсами:

```
root@netology1:~# iptables -A FORWARD -o eth0 -i veth0 -j ACCEPT
root@netology1:~# iptables -A FORWARD -i eth0 -o veth0 -j ACCEPT
```

NAT из сети неймспейса:

```
root@netology1:~# iptables -t nat -A POSTROUTING -s 169.254.1.0/30 -o eth0 -j
MASQUERADE
```

Проверка:

```
root@netology1:~# iptables -L -v | grep anywhere
   16  1344 ACCEPT     all  --  veth0  eth0    anywhere             anywhere
    4   336 ACCEPT     all  --  eth0   veth0   anywhere             anywhere
root@netology1:~# iptables -t nat -L | grep MASQ
MASQUERADE  all  --  169.254.1.0/30       anywhere
root@netology1:~# ip netns exec demo_app ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=334 ms
...
```

netns, стиль Docker по-умолчанию

Docker не следует стандартной схеме, не создавая необходимые файлы в `/var/run/netns`. Поэтому они не видны при вызове ip netns list.

```
root@netology1:~# ID=e63e3c13d249
root@netology1:~# ID_PID=$(docker inspect -f '{{.State.Pid}}' $ID)
root@netology1:~# ln -s /proc/$ID_PID/ns/net /var/run/netns/$ID
root@netology1:~# ip netns exec $ID ip a s eth0 | grep inet
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0

root@netology1:~# brctl show docker0
bridge name     bridge id       STP enabled     interfaces
docker0         8000.0242168ef9fb   no          vethd14e177

root@netology1:~# ip -4 a s docker0 | grep inet
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
```

Стоит отметить, что существуют другие варианты построения сетей для контейнеров, от macvlan [1] до slirp4netns [2], [3], но рассмотреть их все за разумное время просто невозможно.

1. Docker macvlan
2. Podman Rootless (21 страница)
3. slirp4netns

ipvs (LVS) + keepalived (VRRP)*

ipvs – IP Virtual Server = LVS – Linux Virtual Server
VRRP – Virtual Router Redundancy Protocol

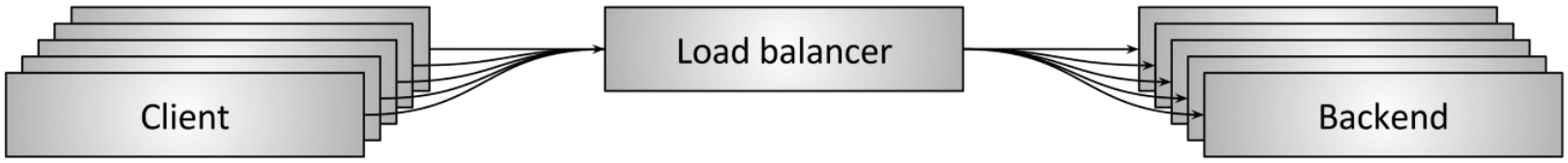
Балансировка и отказоустойчивость

Одна из задач любого серьезного проекта – построить такую систему, чтобы при выходе из строя ее части общее функционирование не нарушилось.
Иными словами – не должны существовать единые точки отказа.

Не слишком важно, используем ли мы физические, виртуальные машины или приложения в контейнерах, – необходима избыточность, а, значит, и механизмы доставки трафика нескольким инстансам приложений. Важно, чтобы и сам балансировщик не был точкой отказа.

Подходов к балансировке множество. Часто, используя конкретные решения, вы будете ограничены определенными возможностями этих решений. Например, Kubernetes реализует конечный набор техник балансировки, и не меняя исходного кода его компонент вы не сможете воспользоваться произвольной схемой.

Но принципы, а также конкретные реализации, знать полезно. При работе вне облачных провайдеров данные знания пригодятся вам наверняка.

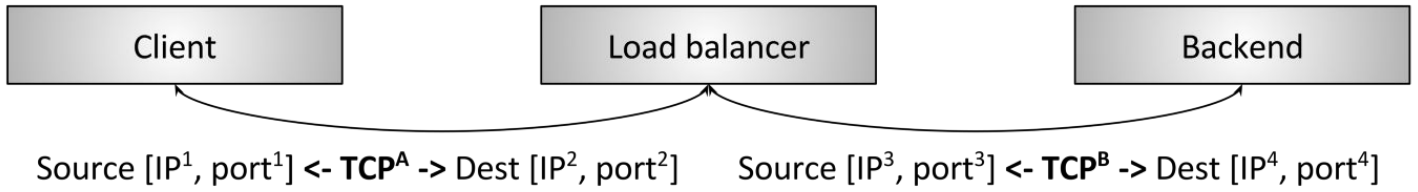


взято с сайта: envoyproxy.io

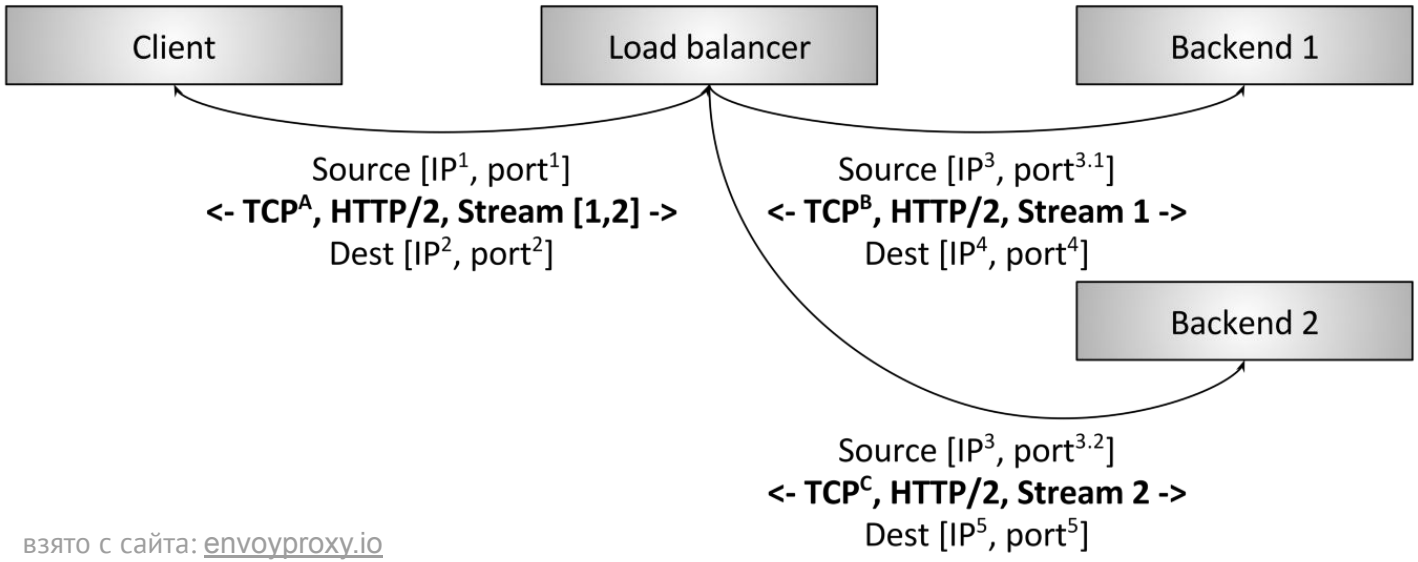
L4 vs. L7

Один из первых вопросов про балансировку, на который необходимо ответить, – на каком уровне ее необходимо реализовать. Наиболее частый выбор (не единственный, см. ECMP и т.д.) – между транспортным уровнем и уровнем приложений.

Балансировка на транспортном уровне L4 (TCP сессии или UDP датаграммы):



Балансировка на уровне приложений L7 (балансировщик знает о протоколе):



взято с сайта: envoyproxy.io

Сильные стороны L4 и L7

Нет правильного и неправильного выбора для балансировщика – у обоих подходов есть свои сильные стороны. L7 reverse-proxy сможет “разобрать” высокоуровневый протокол (например, HTTP), и по каким-то признакам, даже принять решение о дальнейшем роутинге запроса. На L4 балансировщик по определению ничего такого делать не может, так как самый высокий уровень, на котором он разбирает заголовки, – транспортный.
С другой стороны, на L7 вы используете заточенное под определенный протокол приложение, то есть данное решение не является универсальным – что подходило для роутинга HTTP запросов не будет работать для балансировки над MySQL-слейвами.

L4

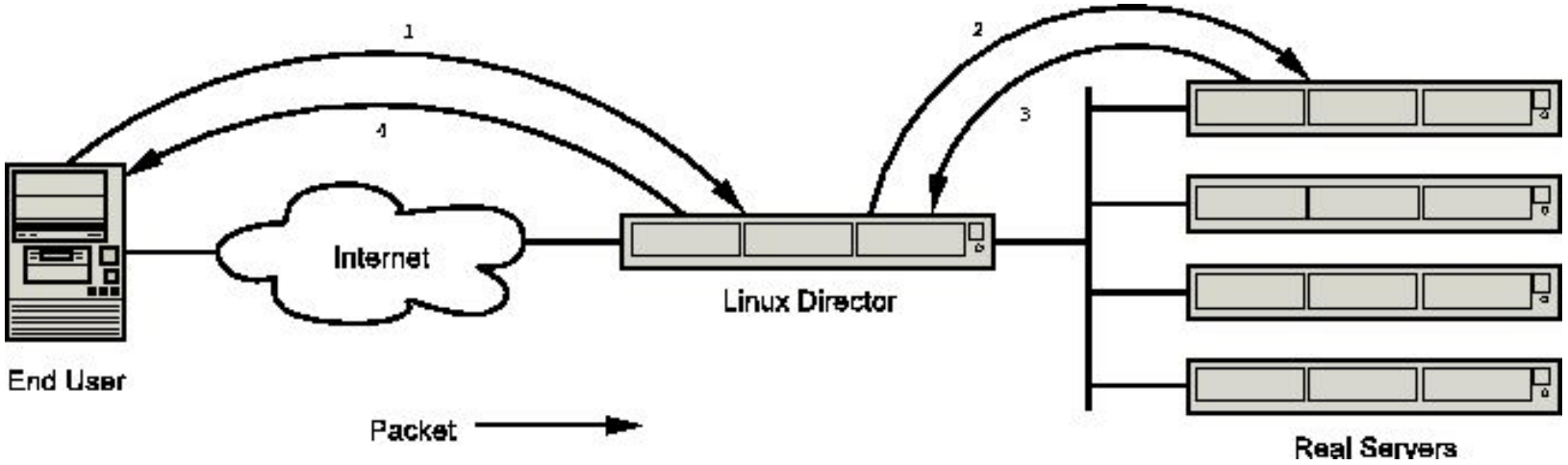
- универсальность
- производительность (10 Гбит/с при ~2 млн pps на 10 ядрах в NAT режиме), работает в kernel-space

L7

- понимание протоколов приложений
- возможность переложить часть работы с приложения, пример – терминирование TLS на балансере
- возможность сбора данных (метрики, логи с запросов) на балансере
- больше возможностей по “прикреплению” пользователя к конкретному экземпляру приложения, опять же, – за счет понимания протокола

L4 балансер на примере ipvs

L7 балансировщикам в курсе посвящен целый раздел, L4 же разумно затронуть в этом модуле. Встроенный в ядро Linux L4 балансировщик – ipvs, или LVS. У ipvs есть 3 режима работы. Начнем с NAT (Network Address Translation, режим трансляция сетевых адресов):



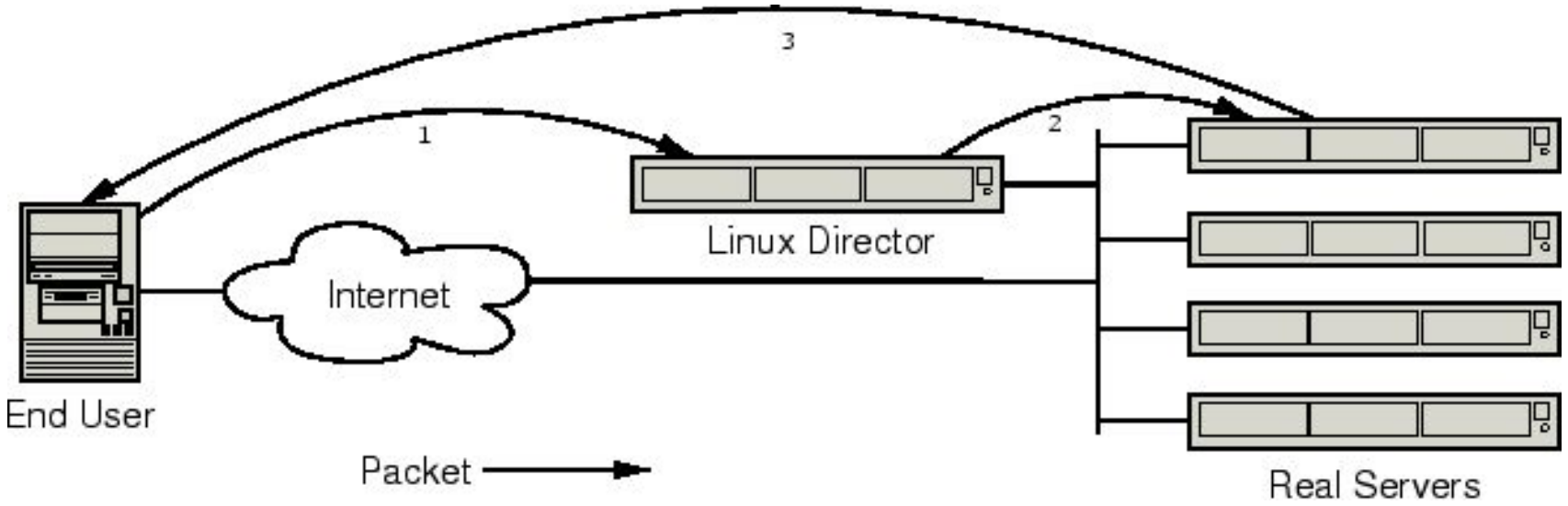
В IPVS есть концепция виртуального сервиса с собственным адресом **– VIP** (Virtual IP). В NAT режиме адрес VIP заменяется DST одного из “real” серверов, где отвечает приложение. Так как для возврата ответа клиенту требуется обратная трансляция, в NAT режиме ответ роутится через балансировщик. Отметим, что системная таблица conntrack не используется ipvs для NAT-режима.

взято с сайта: ultramonkey.org

L4 балансер на примере ipvs, DR/Tunnel

При прямом роутинге (DR, direct routing) трансляции не происходит. ipvs перенаправляет пакет с адресом виртуального сервера на один из real серверов, чтобы обработать такой пакет на последнем должен быть поднят адрес VIP.

Соответственно, отправить ответ пользователю при DR можно уже напрямую клиенту, минуя балансер.



Особенность режима DR – между балансировщиком и real'ам должна быть L2 связность, т.к. в DR ipvs манипулирует Ethernet заголовками напрямую. Третий режим туннелирования схож с DR во всем кроме этого момента – он применяет дополнительную инкапсуляцию IP in IP для доставки пакетов до риалов, поэтому требование L2 связности снимается.

взято с сайта: ultramonkey.org

ipvsadm DR демо, часть 1

Чтобы сделать демо-конфигурацию максимально простой и не путаться в интерфейсах, воспользуемся тремя хостами и режимом по-умолчанию DR.
netology{1,2} будут риалами, при этом на netology1 будет расположен ipvs балансер, с netology3 же мы будем тестировать подключение с клиента к VIP.

В предложенном Vagrant файле на наших тестовых серверах уже установлен и запущен nginx со стандартным конфигурационным файлом.

```
vagrant@netology3:~$ curl -I -s 172.28.128.{10,60}:80 | grep HTTP
HTTP/1.1 200 OK
HTTP/1.1 200 OK
```

При первом вызове ipvs модуль ядра будет загружен автоматически:

```
root@netology1:~# lsmod | grep -c ip_vs
0
root@netology1:~# apt -y install ipvsadm; ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
```

ipvsadm DR демо, часть 2

Так как мы используем DR режим, выберем любой свободный адрес из сети 172.28.128.0/24 (интерфейсы eth1). Например, 172.28.128.200.

```
root@netology1:~# ip -4 addr show eth1 | grep inet
    inet 172.28.128.10/24 scope global eth1
root@netology2:~# ip -4 addr show eth1 | grep inet
    inet 172.28.128.60/24 scope global eth1
```

```
root@netology1:~# ipvsadm -A -t 172.28.128.200:80 -s rr
root@netology1:~# ipvsadm -a -t 172.28.128.200:80 -r 172.28.128.10:80 -g -w 1
root@netology1:~# ipvsadm -a -t 172.28.128.200:80 -r 172.28.128.60:80 -g -w 1
root@netology1:~# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  172.28.128.200:80 rr
  -> 172.28.128.10:80             Masq    1      0          0
  -> 172.28.128.60:80             Masq    1      0          0
```

-A – добавить сервис (VIP)
-t – с протоколом TCP
-s – scheduler, планировщик = алгоритм балансировки, где rr – round robin или последовательное чередование
-a – добавить риал к сервису
-g – direct route режим
-w – вес, при равных весах запросы будут чередоваться

ipvsadm DR демо, часть 3

Что еще надо сделать, чтобы на адрес 172.28.128.200:80 можно было обратиться? Этот адрес не поднят ни на одном из интерфейсов netology{1,2}. Исправим это:

```
root@netology1:~# ip addr add 172.28.128.200/32 dev eth1 label eth1:200
root@netology1:~# ip -4 addr show eth1 | grep inet
    inet 172.28.128.10/24 scope global eth1
    inet 172.28.128.200/32 scope global eth1:200
```

Используем alias сетевого интерфейса eth1:200 для удобства, необходимости чтобы он был равен последнему октету нет. Напомним, на netology1 находится сам балансировщик, то есть он должен обеспечить внешние обращения к 172.28.128.200. netology2 же только должен обрабатывать пересланные балансером пакеты, поэтому:

```
root@netology2:~# ip addr add 172.28.128.200/32 dev lo label lo:200
root@netology2:~# ip -4 addr show lo | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.28.128.200/32 scope global lo:200
```

Если проверить балансировку в данный момент, нас ждут неутешительный результат:

```
root@netology1:~# wc -l /var/log/nginx/access.log
0 /var/log/nginx/access.log
root@netology2:~# wc -l /var/log/nginx/access.log
100 /var/log/nginx/access.log
```

ipvsadm DR демо, часть 4

Так происходит потому, что с настройками ядра по-умолчанию Linux отвечает на ARP запросы вне зависимости от того, с какого интерфейса был получен запрос, а на каком интерфейсе присутствует адрес. У нас же адрес назначен нескольким хостам, и lo отвечает на ARP, что приводит к нехорошей ситуации:

```
root@netology3:~# arping -c1 -I eth1 172.28.128.200
ARPING 172.28.128.200
60 bytes from 08:00:27:ea:ee:52 (172.28.128.200): index=0 time=631.412 usec
60 bytes from 08:00:27:73:ff:5c (172.28.128.200): index=1 time=654.008 usec
---
```

Поменять это поведение:

```
root@netology2:~# sysctl -w net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_ignore = 1
root@netology2:~# sysctl -w net.ipv4.conf.all.arp_announce=2
net.ipv4.conf.all.arp_announce = 2
---
root@netology3:~# arping -c1 -I eth1 172.28.128.200
ARPING 172.28.128.200
60 bytes from 08:00:27:73:ff:5c (172.28.128.200): index=0 time=593.308 usec
```

Документация по sysctl arp_ignore / arp_announce.
Марсианские пакеты.

ipvsadm DR демо, часть 5

После данных изменений балансировка работает хорошо:

```
vagrant@netology3:~$ for i in {1..50}; do curl -I -s 172.28.128.200>/dev/null; done

root@netology1:~# ipvsadm -Ln --stats
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
TCP  172.28.128.200:80                 199     1127        0    74600        0
  -> 172.28.128.10:80                   99      594        0    39500        0
  -> 172.28.128.60:80                  100      533        0    35100        0
```

Это демо опускает нюансы работы ipvs и разные режимы (NAT/Tunnel), но для целей знакомства этого достаточно:

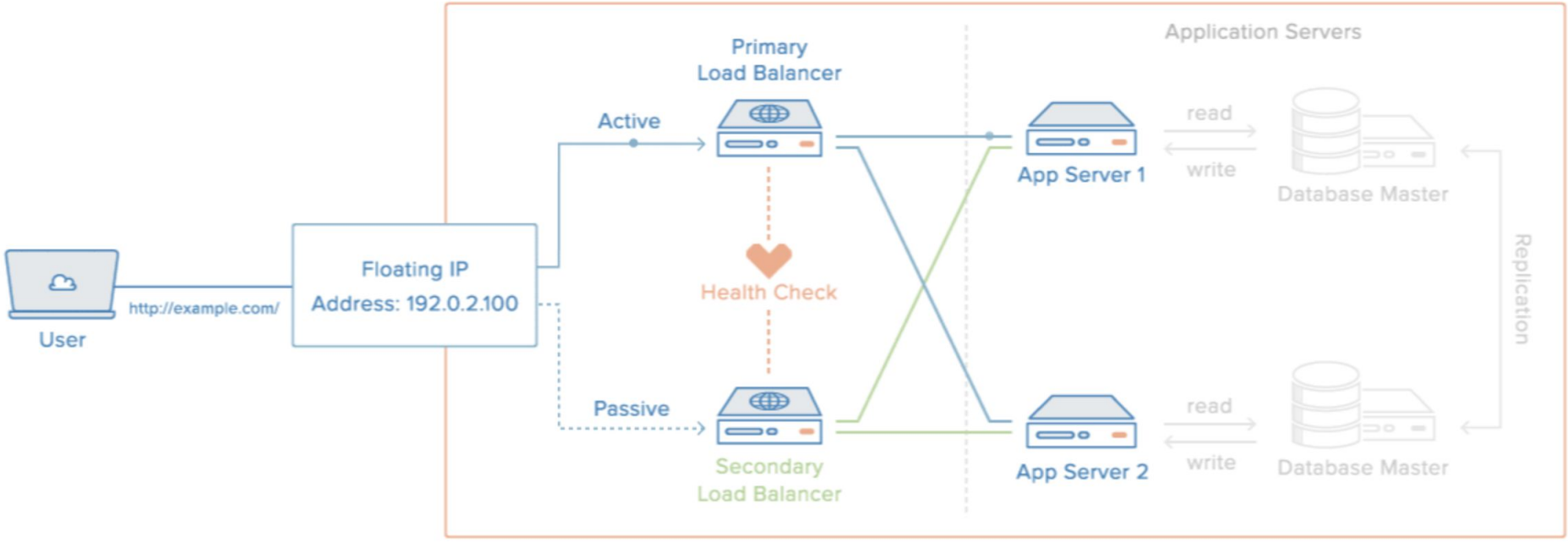
```
root@netology1:~# wc -l /var/log/nginx/access.log
25 /var/log/nginx/access.log
root@netology2:~# wc -l /var/log/nginx/access.log
125 /var/log/nginx/access.log
```

keepalived – реализация VRRP

Внимательные слушатели наверняка заметили, что для реализации ipvs в предыдущем примере мы *подняли внешний IP адрес на интерфейсе только лишь одной виртуальной машины*, значит – создали точку отказа в виде единственного балансера.

keepalived – программное обеспечение, работающее по протоколу VRRP, которое позволит устранить данную проблему (не имеет никакого отношения к HTTP connection keep-alive!). Практическая цель применения keepalived – перемещение адресов VIP между нодами. Например, в DNS могут быть прописаны 2 виртуальных IP, которые в штатном режиме находятся на двух хостах. В случае проблем на одном из хостов, оставшийся “забирает” у него адрес, переводя весь трафик на себя. Когда вторая нода восстанавливается, трафик вновь перетекает в равных частях между машинами.
Такой механизм позволяет не только реагировать на аварии, но и проводить плановые работы без каких-либо проблем для пользователей.

keepalived – реализация VRRP (иллюстрация)



взято с сайта: DO

keepalived, протокол

Следует понимать, что keepalived в сочетании с ipvs – одна из возможных связок. VIP может обслуживаться любым другим ПО, включая балансировщики L7: nginx, envoy и т.д.

Хосты, между которыми необходимо перемещать виртуальный адрес, объединяются в группу. В ней присутствует ведущий **master** и ведомые **backup**. Ведущий хост посылает сообщения о своем штатном состоянии (heartbeat) участникам своей группы. В случае, если участники за определенный период времени не дождались сообщения, старший по номеру участник становится master и поднимает виртуальный IP.

VRRP использует IP multicast для определения master/backup.

```
root@netology2:~# tcpdump -vvv -n -i eth1 host 224.0.0.18
13:57:07.093742 IP (tos 0xc0, ttl 255, id 707, offset 0, flags [none], proto VRRP
(112), length 40)
    172.28.128.60 > 224.0.0.18: vrrp 172.28.128.60 > 224.0.0.18: VRRPv2,
Advertisement, vrid 33, prio 100, authtype simple, intvl 1s, length 20, addrs:
172.28.128.210 auth "netology"
```

keepalived, протокол

```
root@netology{1,2}:~# apt -y install keepalived

/etc/keepalived/keepalived.conf:
vrrp_script chk_nginx {
    script "systemctl status nginx"
    interval 2
}

vrrp_instance VI_1 {
    state MASTER
    interface eth1
    virtual_router_id 33
    priority 100 / 50
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass netology_secret
    }
    virtual_ipaddress {
        172.28.128.210/24 dev eth1
    }
    track_script {
        chk_nginx
    }
}
```

systemctl status keepalived, systemctl status nginx
ip -4 a s eth1

iftop, iperf3, демо

```
root@netology1:~# wget
https://mirror.yandex.ru/ubuntu/dists/focal/main/installer-amd64/current/legacy-ima
ges/netboot/boot.img.gz
--2020-07-07 13:39:42--
https://mirror.yandex.ru/ubuntu/dists/focal/main/installer-amd64/current/legacy-ima
ges/netboot/boot.img.gz

iftop
```

iperf3: сервер + клиент:

```
root@netology1:~# iperf3 -s
Server listening on 5201
[  5] local 172.28.128.10 port 5201 connected to 172.28.128.60 port 53316
[ ID] Interval           Transfer     Bitrate[  5]   9.00-10.00  sec   330 MBytes
2.78 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[  5]   0.00-10.00  sec  3.14 GBytes  2.70 Gbits/sec                  receiver
___
root@netology2:~# iperf3 -c 172.28.128.10
[  5] local 172.28.128.60 port 53316 connected to 172.28.128.10 port 5201
[ ID] Interval           Transfer     Bitrate         Retr    Cwnd
[  5]   0.00-1.00   sec   321 MBytes  2.69 Gbits/sec  2118    175 KBytes
___
root@netology2:~# iperf3 -c 172.28.128.10 -w 10K
[  5] local 172.28.128.60 port 53348 connected to 172.28.128.10 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  30.7 MBytes   258 Mbits/sec    0   18.4 KBytes
```

Итоги

- Посмотрели на практике как работают виртуальные сетевые интерфейсы в контейнерах,
- разобрали балансировка L4 и L7, познакомились с L4 балансером ipvs,
- увидели в работе keepalived в базовой конфигурации.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Андрей Вахутинский

Андрей Вахутинский